Challenges and Solutions for Opening Small and Medium-Scale Industrial Software Platforms

Christoph Seidl
Technical University of Braunschweig, Germany

Thorsten Berger
Chalmers | University of Gothenburg, Sweden

Christoph Elsner, Klaus-Benedikt Schultis
Siemens Corporate Technology, Germany

ABSTRACT

Establishing open software platforms is becoming increasingly important. Many vendors of large and well-known open platforms, such as Android or iOS, have successfully established huge ecosystems of platform extensions (*apps*). While such platforms are important role models, the practices and technologies employed by their vendors are often not applicable for smaller platform vendors, who have different goals and carry substantial legacy, such as an existing closed platform. Yet, many vendors start to open their platforms—for instance, when they alone cannot realize all incoming requirements anymore. Unfortunately, very few best practices exist to guide this opening process, especially for small and medium-scale industrial platforms with their specific solutions. We present a study of industrial organizations that successfully opened closed platforms. Using a survey, we identified 18 opened platforms, providing a broad picture, which is complemented with in-depth, qualitative insights from a case study of three organizations. We elicited the platforms' core characteristics, the organizations' opening strategies, as well as challenges and solutions. We believe that our results support practitioners seeking to open platforms, and researchers striving to build better methods and tools.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Software design tradeoffs**; *Software architectures*;

ACM Reference format:
Christoph Seidl, Thorsten Berger, Christoph Elsner, and Klaus-Benedikt Schultis. Challenges and Solutions for Opening Small and Medium-Scale Industrial Software Platforms. In *Proceedings of SPLC, Sevilla, Spain, 2017.*
DOI: 10.1145/3106195.3106203

1 INTRODUCTION

Software platforms constitute a cornerstone of software development by providing reusable core functionality to applications. While creating platforms has been an essential part of software engineering for many decades, they become increasingly relevant for establishing software product lines and software ecosystems [3, 4, 12]—portfolios of similar products in a particular domain. While product lines focus on intra-organizational reuse, software ecosystems build on platforms that are open to third-party contributions, enabling inter-organizational software development via *extensions*, to benefit from an increased innovation potential or larger product portfolios.

Many successful open platforms exist, such as the Android OS for mobile devices, the Eclipse platform, or the Linux kernel. All successfully established an ecosystem of third-party platform extensions, yet rely on different technologies and business models [3]. Many of these well-known open platforms have been studied, often with a focus on the organizational and business aspects, less on the technology that can be used [2, 16]. While such platforms are essential role models, the practices and technologies employed by their vendors are not necessarily applicable for smaller platform vendors, who often have different goals and carry substantial legacy, such as an existing closed platform which results in different solutions.

Opening a closed platform poses technical, social, and organizational challenges. For instance, new APIs may need to be created or existing ones exposed to external developers. Companies may also need to restructure development teams, raise awareness for the open platform, and improve maintenance strategies. Unfortunately, there are no best practices to guide this process. Prior research has mainly focused on the large, well-known platforms with large ecosystems—such as Android—that have been conceived as open platforms from the very beginning. Empirical data on opening and sustaining small and medium-scale industrial platforms is lacking.

We present a study of opening closed industrial software platforms. Our main focus is on technical aspects, largely sidestepping the organizational and business aspects, yet, reporting those that are caused by technological changes. Using a survey, we identified small and medium-scale platforms that have been opened up to establish an ecosystem, and we elicited core characteristics about the platforms and the opening processes. To better understand the challenges that companies face and the solutions they apply, we conducted an interview-based case study of three platforms identified in the survey, and then triangulated from both sources.

Our subjects are (i) a systems-engineering platform developed by a large platform vendor over decades, (ii) a platform for Internet of Things (IoT) applications that has been conceived only recently, and (iii) a variability-management platform that can, among others, be extended with adapters for interfacing with other engineering tools. The three platforms originate from diverse domains. For instance, the first one stems from a mature domain in a particularly conservative market, while the second one is from a rapidly changing market with fast innovation cycles. The individual opening processes of the platforms serve as landmarks, as they constitute extremes in platform age and market progressiveness as to which challenges are encountered and which solutions are deemed viable. For instance, backward compatibility is crucial in conservative markets and severely restricts the liberty for redesigning APIs.

SPLC'17, Sevilla, Spain

DOI: 10.1145/3106195.3106203

In summary, we contribute:

- quantitative data about the opening of 18 industrial, small- and medium-scale platforms;
- qualitative data about three selected platforms;
- synthesized challenges and solutions for opening;
- an online appendix with detailed survey results [1].

We believe that other organizations seeking to open closed platforms can benefit from our results. The synthesized challenges can be used as a check list of issues to be addressed, and the solutions for devising strategies to cope with the issues. Among researchers, our results create awareness of the challenges, encouraging conceiving better support for opening platforms. Practitioners can learn from the solutions applied in successfully opened industrial platforms.

2 METHODOLOGY

We conducted a survey with industrial participants and an interview-based case study of three particularly interesting platforms. We now describe our methodology.

2.1 Survey

We first conducted a survey to identify industrial companies that have opened a platform or are currently in the process of opening. The survey relied on a questionnaire that aimed at giving a broad overview of the participating companies' practices in opening their software platform. We designed the questionnaire to ask for specific details using closed questions (e.g., using a Likert scale), but also included open-ended questions whose answers permit further exploration. We included questions on the reasons, the process, the technical mechanisms, and the consequences of opening the platform. To assure quality and understandability of the questionnaire, we asked one of our industry partners to tentatively participate in the survey and provide feedback, which we then included in the final version of the questionnaire (cf. online appendix [1]).

To distribute the questionnaire, we assembled a list of potential participants. We consulted previous editions of established venues for software platforms, ecosystems, software product lines, and directly related areas that possess a track with industrial practitioners. In particular, we gathered e-mail addresses of contributors to previous editions of ICSE-SEIP, SPLC Industry Track, GPCE, IWSECO, and EWSECO. In summary, the resulting list comprised e-mail addresses of approximately 300 industry practitioners and academics with proven industry affiliation in the area of software platforms. We invited the potential participants with individualized e-mails and reminded them with a second e-mail.

Note that identifying such opened platforms is challenging. From the venues and the publications of the authors we invited, it was not obvious whether the described open platform originated from an opening process. Thus, despite the relatively high number of invitations, we did not expect a high response rate.

Responses. We received 22 responses. We filtered out four partial (i.e., started, but unsubmitted) responses, obtaining 18 complete ones. We analyzed the participants' responses to obtain quantitative data and to establish a first picture of the industrial opening efforts to select interesting cases for a subsequent case study.

Participants. Most of our participants were software architects (72 %), followed by developers (44 %), team leaders (33 %), and project managers (33 %). Note that multiple roles could be selected. We counted nine other roles (e.g., domain expert, modeler, product manager), which shows that our participants provide a diverse view on platform opening. The participants have substantial industrial experience: most (78 %) have more than 10 years, some (17 %) 5–10 years, and only one 3–5 years.

2.2 Case Study

To qualitatively understand the technical aspects of platform opening in more detail, we interviewed three survey participants.

Selection of Interviewees. To assure diversity of our subjects, we looked for small and medium-scale platforms that stem from different domains and where the opening was completed successfully. We narrowed down the list of survey participants by removing platforms whose opening process was still ongoing, and then invited the survey participants to a follow-up interview. The selection of our three case-study subjects matches our initial criteria for having platforms with a completed opening process from different domains and companies. The interviewees were knowledgeable employees who were involved in the platform opening process.

Interview Design. We conducted semi-structured interviews. Our interview guides focused on the challenges encountered by the platform vendors during the opening process and on the individual solutions they found to cope with these challenges. The guides also comprised specifically interesting answers provided in the questionnaire as well as areas for clarification. We further explored challenges mentioned in the questionnaire and allowed room for interviewees to proactively raise challenges and present solutions we did not predict. The interviews were recorded and transcribed.

Interviewees. Our interviewees were a senior software architect, a senior software developer, and a senior software architect who was also the CTO of the platform vendor. They have been working for their company for more than ten years, for four years, and for almost 15 years, respectively. Each interviewee belonged to a distinct platform vendor. We use quotations from the interviewees in the remainder, abbreviated with **A**, **B**, and **C**, respectively.

2.3 Data Analysis

We triangulated data from the survey and the case study. In the survey, we posed questions to confirm or refute challenges we predicted, such as maintaining backward compatibility. We analyzed the survey data by creating aggregate statistics for closed questions, creating violin plots for the Likert-scale questions (interpreting the levels of agreement as a continuous scale), and inspecting the responses to the open-ended questions. In the case study, we used open coding to assign reappearing key phrases related to challenges and solutions of platform opening to statements in the interview transcripts. From the full set of challenges and solutions, we created clusters to form two categories related to opening the platform and sustaining the open platform. In a further round, we clarified individual points with the interviewees, including challenges encountered in other platforms but not mentioned in the interview.

3 THE SOFTWARE PLATFORMS

We now introduce the platforms we identified in the survey and the three of these we focused in the case study.

Table 1: Overview of the case-study platforms

	Systems Engineering Platform	Internet of Things Platform	Variability Management Platform
domain	computer aided eng.	device mgmt. and control	software synthesis
size	5–20 MLOC	1–5 MLOC	1–5 MLOC
languages	C, C++, C#	C++, Java	Java, C++, C#, Prolog
no. developers	51-100	101–150	5-10
platform age	25 years	2 years	13 years
extension form	components	services	plug-ins, scripts
no. extensions	350	30–50	25

3.1 Platforms Identified in the Survey

The surveyed platforms stem from diverse domains and companies. Most frequently, they are used for software development and modeling for specific applications, such as web or IoT applications. All of our 18 respondents also provided the name of their platform. None is a well-known open-source platform. Instead, all are industrial, niche platforms. Note that every respondent reported on a unique platform, so that we identified 18 platforms.

Sizes. The size of the majority of platforms (33 %) lies between 150–500 KLOC. Many are even larger with 1–5 MLOC (28 %) or ultra-large with 5–20 MLOC (17 %). Furthermore, we identified two platforms (17 %) each with a size between 500 KLOC and 1 MLOC, and one particularly small platform with 50–150 KLOC.

Programming Languages. The most frequent programming language mentioned by our participants is C++ (50 %), followed by C and Java (39 % each), C# (28 %), as well as JavaScript (11 %). Single participants also mentioned Ruby, HTML, Prolog, Python, and "various scripting languages."

Developers. Most participants (33 %) stated moderate team sizes of 5–15 people. The team sizes went up via 16–50 (17 %), 51–100 (22 %), and 101-250 (6 %) to 251–500 (17 %) people. Only one participant (6 %) mentioned a team size of less than five people.

Users. Interestingly, the users of all identified platforms are not primarily end-users, but other developers, other departments in the same company, or other companies. Furthermore, the users of all platforms were described as technically skilled.

Extensions. Most frequently, the platform extensions are called components (42 %), followed by plugins (33 %), scripts (25 %), applications (22 %), and packages (11 %). Single participants also mentioned drivers, snap-ins, features, GUI widgets, services, and droplets (used in Cloud Foundry). Based on participants' estimates, seven platforms (39 %) have 26–100 extensions, followed by five platforms (28 %) with less than five extensions, four platforms (22 %) with 6–25 extensions, and two platforms (11 %) with 101–500 extensions.

3.2 Platforms Examined in the Case Study

The three selected case-study platforms represent a range of platform sizes, companies, and domains as summarized in Table 2. We now briefly describe each platform.

3.2.1 The Systems Engineering Platform (SE Platform). Our first subject is a platform for the companies' own engineering applications as well as for over 350 engineering applications of about 150 independent software companies. It is delivered with a comprehensive development kit including tools, developer resources, and documentation for integration into new and existing applications. Specifically, the platform provides over 800 C/C++ and C# API functions for developing comprehensive and robust system engineering applications. The platform has been sold for over 25 years.

3.2.2 The Internet of Things Platform (IoT Platform). Our second subject is a platform for developing applications for the innovation-driven IoT market—for instance, to manage devices in a manufacturing plant. For IoT development, the platform offers a wide variety of functionality, including business-rule and business-process management, or data processing and analysis.

3.2.3 The Variability Management Platform (VM Platform). Our third subject is a feature-modeling and software-synthesis tool. It supports customers in adopting and maintaining a software product line strategy, where a variable software system has to be managed in terms of features (modeled in a feature model). The tool needs to read and modify different kinds of implementation artifacts, including source code and design models. It also needs to be integrated into existing workflows of customers, interfacing with software development tools for the implementation artifacts, such as Sparx Enterprise Architect for design models or Doors for requirements.

4 OPENING THE PLATFORMS

In this section, we discuss the platform openings as experienced by the 18 survey and three case-study participants. We first present the intentions associated with the opening, including any specific problems that the preceding closed platforms faced, followed by the procedures for opening. Finally, we synthesize challenges and solutions we elicited in the survey and case study results.

It is worth noting that none of the 18 survey participants reported a failure of platform opening. The majority (72 %) either agreed or strongly agreed, and only less than a third (28 %) were neutral, with the statement that the opening was successful. Hence, we describe the results in terms of *how* to cope with challenges and not *whether* the platform vendors could cope with them. Identifying and studying failed attempts would be valuable future work.

4.1 Reasons for Opening

To understand the intentions behind platform opening, we discuss any potential problems that might have existed for the preceding closed platform (if it existed) and the explicit goals the platform vendors pursued for the opening.

4.1.1 Problems with Preceding Closed Platform. The majority (56 %) of participants stated that a previously closed platform existed. Two of the open platforms (20 %) are a complete re-implementation of the respective closed platform. Interestingly, 31 % of the open platforms we identified were conceived and developed as open platforms from their inception.

We asked whether there were any specific problems with the closed platform. Most participants stated too many new requirements (44 %), followed by strong competition in the market (39 %).

Interestingly, many (22 %) faced no problems before opening. Further problems mentioned include maintenance difficulties (17 %), lack of compatibility with other platforms (17 %), and difficult integration of extensions. Single participants expressed conflicting requirements and development scale effects.

The **SE Platform** was initially designed to be supplied to one specific customer. During the course of platform development, potential for opening was identified and it was incrementally opened to other customers. A specific problem to be addressed was the usability of existing APIs in the closed platform.

For the **IoT Platform**, the preceding closed platform consisted of a collection of tools connected via technology bridges to form a tool chain. These tools were acquired by the platform vendor by buying other enterprises. However, this integrated tool suite was difficult to evolve and extend due to different technologies of individual tools: **B:** *They are not that comfortable as they should be so we took several means to connect them together. But in the end it was quite hard to do so.* Furthermore, the applications created by customers are relatively distinct, which was hard to accommodate for, due to coarse-grained and inflexible functionality of individual tools: **B:** *Another big issue was that the old tools were monolithic in their architecture.* Finally, customers requested additional, specialized functionality, which was inefficient to realize with the old infrastructure.

For the **VM Platform**, before opening, a challenge was that it needed to interface with a large number of external tools. Thus, the platform vendor was faced with substantial efforts for acquiring knowledge about the tools in order to develop and maintain respective tool adapters. Consequently, the platform vendor decided to open up the platform to allow external developers (usually developers of external tools) to create adapters themselves.

4.1.2 Pursued Opening Goals. We first asked about any business intentions associated with the opening. Most frequently, respondents mentioned fostering innovation (61 %) and sharing cost of innovation (50 %), followed by increasing the number of users (39 %), establishing a value chain (33 %), and increasing platform attractiveness for new users (33 %). Many other business intentions were also given, such as increasing the value for existing users, increasing the user binding, and establishing a unique selling point (28 % each).

We also asked about any technical intentions that were associated with the platform opening. Our participants most frequently mentioned the external realization of requirements beyond the company's capacity (83 %) and of specialized requirements (78 %). Platform modernization was also mentioned by five participants (28 %). More intentions, including facilitating (6 %) and improving (11 %) platform compatibility, improving development efficiency (6 %), and "adapting to future open-source components" (6 %), were given, but do not seem to be general intentions associated with platform opening among our participants.

For the **SE Platform**, one of the main objectives of open platform design was increased revenue generation by leveraging cooperative business models. Moreover, the platform organization pursued the objective to increase domain knowledge by close relationships to external customers, which resulted in improved architecture and functionality for the internal customers.

For the **IoT Platform**, the overall goal of the opening was to allow customers to develop extensions for all areas required to build IoT applications. On a technical level, the interoperability of the functionalities provided by the platform should be increased. The existing monolithic tools should be re-engineered into smaller micro-services and then be lifted to the cloud environment so that they can be used to compose complex IoT applications.

For the **VM Platform**, opening aimed at two goals: allowing the integration of external-tool adapters as plugins, and incorporating a scripting mechanism (using JavaScript) to allow adding smaller functionality, such as specific UI actions. The business intent was to shift efforts from platform developers to its users.

4.2 Opening Procedure

From the 13 participants who provided details on the opening procedure they applied, the majority (54 %) stated that they used an incremental strategy. The more general steps consisted of scoping functionality provided via APIs (62 %) and implementing the respective APIs (46 %). However, some specifics of the opening procedure differed so that specialized steps were also mentioned, such as getting an overview of existing functionality (15 %), acquiring funding (8 %) or reducing expectations for features to be realized (8 %). In this same line, the case-study subjects had individually distinct opening procedures as follows.

SE Platform. Initially, the platform only was opened to few lead customers to receive early feedback on platform concepts and interfaces. Afterwards, the platform was opened to a wider group of external software vendors, including "off-the-shelf" customers. **A:** *Since then, the platform organization constantly built up better domain knowledge by close contact to customers.* This was considered as critical for constant improvement and success. Based on additional customer requirements and constant feedback, platform interfaces were redesigned following two basic design principles: First, make APIs more easy to find, understand, and use (especially by externals). Second, make APIs evolution-ready and backward compatible. **A:** *A new platform version must be always binary-compatible to the previous version for plug-and-play upgrades on customer sites.*

IoT Platform. First, the vendor re-engineered the individual monolithic tools into micro-services relying on the Spring framework: **B:** *So now we split them up into smaller pieces and these are much easier to use than before.* While the application logic could largely be reused, code needed to be written for handling the lifecycle of the services. In addition, interfaces for the services were implemented as language-independent REST APIs. The APIs were defined with potential external developers in mind, for instance, by repeatedly developing plausible mock applications for the services. The functionality of the services was determined from the acquired tools, and the APIs were created mostly anew instead of re-engineering existing ones. Second, the services were lifted into a cloud infrastructure relying on Cloud Foundry. The latter also offers a marketplace (similar to Google Play), allowing external customers to distribute their developed third-party services: **B:** *In this market place, you have your service, then someone can buy this service and then the service is copied into the organizational space of the person who has bought the service.* Focus teams were formed for functionalities that were identified to be useful in the new cloud-based platform. The overall process was iterative so that both steps were performed multiple times.

Table 2: Overview of challenges and solutions

	Challenge	Solutions
opening the platform	1: Assure Backward Compatibility	Version Option Structures, Functionality Deprecation, Immutable Interfaces
	2: Incorporate New Technology	Generative Technologies, Micro-Services, Scripting Engine
	3: Alter Organizational Structure	Level Playing Field, DevOps Teams, Support for External Developers
	4: Restructure Software Architecture	Interface Redesign, Refactoring to Cloud Micro-Services, JavaScript-Based Extension Engine
	5: Interface with Extensions	Long-term Stability and Legacy Support, Design of New Interfaces, Standardization of Interfaces
sustaining the platform	6: Assure Quality	Capture-and-Replay Mechanism, Proactive Error Avoidance, Automated Test Cases
	7: Manage Releases	Maintenance Plan for Multiple Releases, Decoupled Releases of Micro-Services, API Compatibility for Service Releases
	8: Protect Intellectual Property (IP)	Vendor-Private Extensions, Customer-Private Extensions, Only Public Extensions
	9: Determine New Extensions	Extensions Through Community Request, Extensions Through Prototype Feedback, Extensions Through Customer Request
	10: Sustain Strategic Advantage	Focus on Stability, Focus on Variety, Focus on Openness

VM Platform. First, as a preliminary activity, the platform vendor participated in a research project to determine requirements for an exchange format to interface with external tools. The result was a file format, commands, and a specific protocol applicable by the platform and the interfacing tools. Platform developers then created exporters and importers to integrate this standardized format into the platform. Second, platform developers devised Java APIs for use by the prospective plug-ins. The following implementation of the plug-in mechanism benefited greatly from the platform's underlying Eclipse framework, which already provides extension mechanisms relying on OSGi. Third, to create support for the scripting mechanism, the vendor incorporated a JavaScript engine to parse and execute scripts. To also make functionality for interfacing with other tools available to the scripting mechanism, platform developers created a dedicated data binding of the interfaces devised for the Java APIs to make them available in JavaScript.

4.3 Opening Challenges and Solutions

We identified the following challenges and solutions related to *opening* the platforms, as summarized in the upper half of Table 2.

In the survey, our participants most frequently (67 %) pointed out that backward compatibility was difficult to maintain, directly followed by challenges with introducing new technologies (56 %) and restructuring the architecture (50 %). Restructuring the teams was also seen as a challenge, but only by less than a third of the participants (28 %). Interestingly, modeling of the ecosystem and user acceptance were infrequent challenges (17 % each). The former could be attributed to insufficient support of existing approaches for interpreting and leveraging such models. We can also assume that the structure of the ecosystems is less complex for our small and medium-scale platforms, since their users are technically skilled.

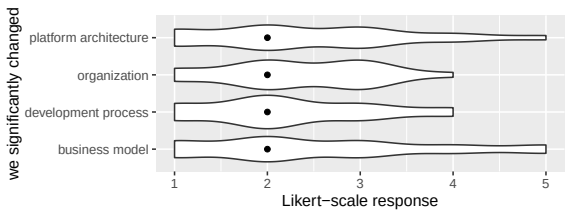
These challenges are supported by the results of a survey Likert-scale question where we asked whether any of the following four aspects—business model, platform architecture, development process, and organization of the development—had to be changed during opening. As can be seen in Fig. 1, for all aspects, the tendency was clearly on the change side, whereas we see some small differences. For the platform development process, we see the strongest agreement (67 % of our participants either agreed or strongly agreed), whereas for the other aspects, the agreement is a bit less dominant (56 % agreed or strongly agreed). The weakest agreement can be seen for the business model, where 22 % of our participants either disagreed or strongly disagreed that it changed. Yet, the differences are small among the four aspects.

4.3.1 Challenge 1: Assure Backward Compatibility. The API is the interface for extension developers to communicate with the platform. As this creates dependencies from the extension to the platform, software evolution that modifies the API becomes a major challenge when backward compatibility is an essential concern to permit extensions to remain functional with new platform versions.

Version Option Structures (VOS): For the **SE Platform**, which operates in a market aimed at longevity, backward compatibility is imperative. In consequence, the vendor chose an upgrade strategy that guarantees binary compatibility to the last revision: Every function of the C interface has as last parameter a struct representing the currently handled "object", called *version option structure (VOS)*. When the API evolves, instead of changing the function's signature, platform developers create a new version of the VOS. It carries members that represent the intended parameters of the API at that version and can only be created via a specific macro. Furthermore, the implementation of the functions can handle VOS of arbitrary versions. In case they are outdated, the version is automatically upgraded on first use in a function. This ensures binary compatibility of the application with any future version of the platform.

Functionality Deprecation: The **IoT Platform** operates in an innovation-driven domain, which allows a more liberal approach to API evolution. During the inception of new platform functionality, interfaces are generally considered unstable and may change without notice. During production use of an API, outdated functionality in the interfaces may be marked as deprecated, and then removed in later releases. This practice gives extension developers time to adapt to API changes, but delegates the burden of maintaining compatibility between platform and extensions to them.

Immutable Interfaces: The **VM Platform** operates in a domain that equally values innovation and stability so that the platform vendor strives for a balanced approach to API backward compatibility. The general guideline is to not remove but only add functionality to APIs during evolution. If a certain functionality of the API is truly outdated, it may be marked as deprecated. In this case, it is not



1: strongly agree, 2: agree, 3: neutral, 4: disagree, 5: strongly disagree

Figure 1: Aspects that needed to be changed during opening

developed further, but remains a part of the API, as the platform vendor yet has to remove the first deprecated API method.

4.3.2 Challenge 2: Incorporate New Technology. With the opening, all survey and case-study subjects needed to incorporate new technology. This new technology primarily centered around the realization of APIs. In fact, the majority of our survey participants (77 %) stated that an API was incorporated. Yet, the actual API technologies were diverse, ranging from C-based APIs via object-oriented APIs (in Java, C#, and C++) to REST and JavaScript-based APIs. Half of the platforms (47 %) also introduced a plug-in system, e.g., relying on Eclipse (two platforms) or on a home-grown system. The others did not specify the plug-in technology further but reported the use cases of their plug-in system: drivers, user-interface elements, and human-machine interaction components.

Other common technologies, each mentioned by almost a third of our participants (29 %), are isolated runtime containers (i.e., sandboxes; examples mentioned are Docker, PikeOS, and Cloud Foundry), web services, and Domain-Specific Languages (examples mentioned are XML-based configuration files, scripts, and macros). Explicitly formulated conventions also play a role for four (24 %) of the platforms. Finally, we asked for the use of conditional compilation (e.g., #IFDEF) but it was not mentioned by any participant and does not seem to be practical for their platforms.

Interestingly, for almost all (88 %) of the 18 surveyed platforms, the platform opening had no significant impact on the code size, which is a promising insight. Only two participants (12 %) reported an increase; one estimated a code-size increase by 30 %.

Our case-study subjects incorporated diverse new technologies during the platform opening process. Despite the introduction effort, all three subjects benefited from the new technology and used this opportunity to reduce the amount of legacy accumulated over time, which also reduced maintenance burdens.

Generative Technologies: For the **SE Platform**, the introduction of new technology was problematic, given its domain, which values stability and backward compatibility. The platform APIs were originally written exclusively in C/C++, but new customers urgently demanded C# interfaces. To avoid duplicate development and the burden of maintaining both C/C++ and C# interfaces manually, the platform vendor decided to employ generative technologies to automatically create C# interfaces from C/C++ ones. Furthermore, to test the C# interfaces with the existing C/C++ test suite, the vendor created a generator for inverse interfaces that are specified in C/C++ but forward calls to the C# interfaces. Due to the complexity of the generative procedures, incorporating this new technology into the platform entailed substantial effort. However, according to our interviewee, this cost was soon amortized by avoiding ongoing cost for manually maintaining the API in multiple languages.

Micro-Services: During opening, the **IoT Platform** and its extensions were structured into micro-services that can be composed to applications. To realize the services, the vendor incorporated the Spring component framework (which manages the service lifecycles), integrated the Cloud Foundry technology for cloud deployment, and adopted various NoSQL databases (MongoDB, among others). While proving to be an aid in handling services in the long term, the infrastructure required writing significant amounts of boilerplate code for each service to utilize lifecycle management, which yielded substantial cost in the short term. **B:** *You have the old code base, then you take that and rewrite it a bit and then you remove parts of it, but then you have to write new parts like this Cloud Foundry related stuff; and then all this Spring lifecycle-management thing [...] I think in the end it is not less code than before.* Our interviewee also stated that they had expected the infrastructure to provide further functionality that would better justify adoption efforts.

Scripting Engine: For the **VM Platform**, one of the two created extension mechanisms, the scripting, required incorporating a JavaScript engine, used for parsing and interpreting JavaScript code. To this end, the created Java APIs also needed to be made available to the JavaScript extensions via data bindings.

4.3.3 Challenge 3: Alter Organizational Structure. Opening the platforms did not only affect technology, but also the organizational structures, especially for development teams.

Level Playing Field: The **SE Platform** vendor operates a so-called *level playing field* business model offering exactly the same platform capabilities to external and internal customers. Nevertheless, internal customers developed functionality containing reusable business knowledge that should not be shared. Consequently, this functionality was not made part of the platform but of the internal applications. The organizational structure had to be changed in a way that development teams aligned with these architectural boundaries dividing them into developers of applications for internal use and developers of extensions for external use. Furthermore, our interviewee suggested to **A:** *Make sure to get a clear interaction model in place*, which captures and makes apparent the value chains in the ecosystem. Its prior lack was reflected in some problematic platform adoptions by internal customers.

DevOps Teams: When the **IoT Platform** vendor re-engineered the tools of the closed platform into micro-services, development teams were restructured from teams focusing on the tools to teams around the prospective services to better align with the overall restructuring. This yielded smaller teams with new combinations of team members. Each team, in addition to the service developers, was joined by an operator designated to administer the service in the future. These so-called DevOps Teams created short feedback loops between development and usage of a service, which fostered rapid improvement of each service's capabilities and interfaces.

Support for External Developers: The **VM Platform** vendor aimed at reducing own development efforts for adapters that interface with external engineering tools. This shift toward the external developers in fact freed resources at the vendor side, but created a need to cope with an increase in support requests. As a consequence, the platform vendor reassigned roles of their employees to have fewer in-house developers and more employees supporting extension developers through seminars, workshops or live support.

Finally, an interesting challenge reported in the survey is the organization and maturity of remote development teams who realize extensions. For a monitoring platform, the opening consisted of allowing regional development teams to realize local monitoring solutions (using concepts of the platform, including alarms, points, drivers). While it could technically be supported, our participant reported process immaturity of the regional teams, which appears to be difficult to realize. Although the platform uses various extension-mechanism technologies (e.g., API into processes and

drivers, ability to run custom scripts), we can speculate that it does not use strict encapsulation mechanisms (e.g., interaction binding, strong interface definition facilities) [3] of large open platforms such as Android, which are hard to implement when many and diverse extension mechanisms are offered.

4.3.4 Challenge 4: Restructure Software Architecture. Recall that the median of our 18 surveyed platforms agreed that the platform needed to be changed (cf. Fig. 1). Our case study revealed that substantial architectural changes (i.e., changing how the platform is decomposed) were only required for the IoT Platform, where tools had to be restructured completely to realize micro-services. For the SE Platform and the VM Platform, which were already based on extensible frameworks, some re-engineering was required, but no further restructuring.

Interface Redesign: For the **SE Platform**, even the initial platform, designed for only one customer, had a clearly separated external interface. There was no need for fundamental architecture restructure; each customer is running a dedicated platform instance. However, based on customer feedback, platform interfaces were redesigned to increase interface quality, to simplify interface usage and reduce integration costs of customers. To obtain requirements for the architecture, our interviewee explained that the vendor extensively tries to negotiate requirements but that this depends on the clients. **A:** *We have some customers with very good relationships. Here we can perform a full architectural analysis what could be the best overall solution for all participants. However, there are also customers that just try to put requirements into the platform, whether or not it makes sense. We [...] try to develop "champions" [...] who act as hubs and contact persons at the client.*

Refactoring to Cloud-Based Micro-Services: For the **IoT Platform**, recall that the closed platform's architecture consisted of a set of tools loosely connected via technology bridges. Due to the insufficient extensibility and interoperability, the platform vendor fundamentally changed the architecture to a cloud-based micro-service architecture. In this process, the individual tools were re-engineered into services comprising the essential business logic of the tools. To this end, a pragmatic scoping was applied: **B:** *I mean the old tools are capable of much more than what the micro-services [...] can do. But we decided to open up this functionality exactly [because we think those are] the things people need to buy to create IoT applications.* While performing these architectural changes resulted in an increase in code size (recall the boilerplate code from Sec. 4.3.2), the re-engineering of the monolithic tools reduced redundancy stemming from overlapping functionality.

JavaScript-Based Extension Engine: The **VM Platform**'s architecture is heavily influenced by the underlying Eclipse/OSGi framework, which already provides expressive extensibility. Thus, the opening (i.e., adding a plugin mechanism for tool adapters) did not jeopardize the original platform architecture. However, in addition to dedicated plugins, the platform vendors intended to allow lightweight extensions in an ad hoc manner (e.g., for customers to add individual UI elements). For this extension mechanism, a JavaScript engine was incorporated into the platform to process JavaScript scripts of a specific format to incorporate UI elements with respective actions assigned to them. This, however, did not require any significant architecture changes according to the interviewee.

4.3.5 Challenge 5: Interface with Extensions. The survey showed that adding or modifying APIs is the most essential activity during platform opening. Our three subjects struggled to varying extents.

Long-term Stability and Legacy Support: For the **SE Platform**, existing APIs needed to be redesigned. The goal was to reduce integration cost for extension developers by increasing API usability. When initially opening the platform, the platform vendor had devised an interface for extensions. Over the course of time, this interface had to be re-engineered to uphold quality standards and to adopt new technology. In consequence, the platform vendor designed a C interface that utilized object-based principles where instructions of a procedural language are used to largely emulate structures known from object-oriented languages. To this day, this C API (or its generated C# counterpart) is used for extensions to interface with the platform. However, the platform vendor still offers the previous API, which is almost 20 years old, to satisfy needs of long-term customers.

Design of New Interfaces: For the **IoT Platform**, APIs were designed completely anew, as the platform consisted of multiple tools before the opening. While this procedure provided more liberty in API design, it was complicated, as no prospective users existed who could provide concrete requirements. The engineers addressed this challenge by taking on the role of platform users themselves to develop mock-up applications upon the platform. During this process, benefits and shortcomings of the current iteration of tentative APIs were collected to form the basis for further development. After the opening, a similar process was employed, where external users of the platform developed products and provided feedback regarding benefits and shortcomings of the interfaces, which could then be used for further development of the APIs.

Standardization of Interfaces: For the **VM Platform**, the vendor's research project elicited requirements for a data exchange format. This format largely guided the design of APIs for assembling, modifying, and processing information of engineering artifacts (e.g., requirements, design models, code) in the external tools. In addition, the platform vendor also applied an iterative design procedure where, after internal revisions, earlier versions of APIs were provided in a beta phase to customers, who could suggest changes until the interfaces reached a stable state.

5 SUSTAINING THE OPEN PLATFORMS

We identified the following challenges and solutions for *sustaining* the opened platforms, as summarized in the lower half of Table 2.

5.1 Essential Aspects for Sustaining

In the survey, we asked about aspects our participants consider essential for sustaining the open platform. As shown in Fig. 2, the strongest agreement with the seven aspects we suggested in the question can be seen for stable extension mechanisms and the platform quality, where 83 % and 67 % of our participants strongly agreed. These two aspects are followed by the extension quality and the extension quality assurance, where 67 % and 56 % agreed. For community management (e.g., forum, blog, social channels), we still see more agreement than disagreement. In contrast, while the medium of participants expressed a neutral agreement, the distribution is skewed toward disagreement with the two aspects

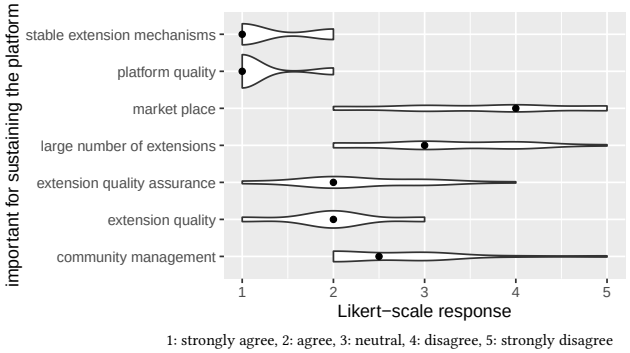


Figure 2: Aspects for sustaining the open platform

large number of extensions and availability of a market place. This attitude toward "quality over quantity" for their extensions confirms the overall characteristics and goals of our subject platforms, which are different from the large platforms, such as Android [3], which strongly encourage large numbers of extensions to further their platforms' successes, specifically, they foster community innovation and let the community define the scope of extensions.

5.2 Sustaining Challenges and Solutions

Our case-study participants expressed challenges related to quality assurance, release management, protection of intellectual property (IP), and how to scope future extensions and exceeding competitors.

5.2.1 Challenge 6: Assure Quality. Recall that quality assurance of the platform and of the extensions (yet, with a stronger focus on the former) were among the most essential aspects for sustaining the open platform. Our survey shows a high diversity of quality-assurance techniques. Interestingly, the most frequent technique (28 %) is the manual review of extensions (mentioned both for the smaller and larger platforms), followed by certification of extensions (17 %), of the contributors (11 %), and of the development process (11 %). Some participants (17 %) also pointed out that contracts are made that oblige contributors to use certain quality-assurance mechanisms. In only one platform (6 %), which also uses extension certification, the extensions are reviewed automatically. Interestingly, four participants explained that no specific quality-assurance mechanisms are used and three participants that extension developers assure quality themselves. Our three case-study platforms employ different quality-assurance mechanisms as follows.

Capture-and-Replay Mechanism: For the **SE Platform**, in addition to a platform test suite, an elaborate capture-and-replay mechanism was created. If components expose faulty behavior, a recording mechanism logs the conditions under which the behavior appeared and retraces the calls that led to the behavior. The log is sent to the platform developers for inspection. If the cause lies within the platform, it is addressed directly; otherwise, extension developers are informed. In either case, the captured error provides the basis for new test cases to assure an improved error handling of functionality within platform or extensions.

Proactive Error Avoidance: In the **IoT Platform**, multiple mechanisms to proactively avoid extension errors were realized (as opposed to detecting and fixing errors retroactively) as part of the contribution process for the market place: For one, checklists assure compliance of extensions with technical requirements. Furthermore, extensions are checked for resource consumption especially when using other extensions. Finally, a procedure to certify extension developers was established to give them an increased level of autonomy in contributing to the market place when they apply adequate development procedures.

Automated Test Cases: For the **VM Platform**, an existing platform test suite was extended to cover extensions. This is possible due to the standard interface used by all extensions. Test cases are integrated in an automated build process yielding nightly or release builds after all tests passed.

5.2.2 Challenge 7: Manage Releases. With existing applications and extensions built upon the platform, our subjects face potentially negative impacts (e.g., incompatibilities) on existing applications during platform evolution. Our vendors carefully devised a release strategy for new platform versions as follows.

Maintenance Plan for Multiple Releases: The **SE Platform** uses one single release strategy for internal and external customers, given its level playing field (cf. Sec. 4.3.3) business model. A full release with major changes is rolled out every six months. In addition, the vendor actively maintains the last three major platform versions, for each of which a maintenance patch is provided every three weeks. The patch supply date is staggered, so that one patch is released every week.

Decoupled Releases of Micro-Services: The **IoT Platform** has a special situation regarding platform upgrades. The platform is comprised solely of a collection of services that may be combined freely, so only the applications connect the services, not the platform itself. Consequently, there are no distinct releases of the platform. Instead, individual services are constantly updated to new versions to upgrade the platform incrementally.

API Compatibility for Service Releases: The **VM Platform** has a major release cycle of 1.5–2 years. During major releases, existing functionality may be deprecated and previously deprecated functionality may be removed from the platform. As this can cause incompatibilities, four service releases are provided per year, which are guaranteed to maintain API compatibility and only fix existing or add new functionality.

5.2.3 Challenge 8: Protect Intellectual Property (IP). For our platforms, many customers are direct competitors in the market. This leads to challenges when innovative new functionality needs to be protected against competitors use.

Vendor-Private Extensions: The **SE Platform** has a significant number of company-internal customers. When new, innovative functionality is devised within the company, it is tempting to provide it in the platform in a reusable way. However, the functionality may be of substantial business value that should not be shared with competitors. As a result, the vendor tends to place such innovative code in extensions even when it complicates the overall architecture of applications built upon the platform.

Customer-Private Extensions: The **IoT Platform** offers extensions in the form of services potentially available to all customers to build applications. The vendor's extensions are generally made public, unless they are developed on customer request, where customers can choose whether to make them public. If customers decide otherwise, the extension is run for their exclusive use. Apart

from prototypes, only third parties – not the vendor itself – develop applications based on the platform.

Only Public Extensions: The **VM Platform** is not challenged by IP issues. Extensions are made available for public use. Recall that the platform is extensible with adapters for external engineering tools. While the extension developers may be competitors, the actual intellectual property is in the engineering tools, not the adapters. So there are no benefits in protecting the extensions against a competitor.

5.2.4 Challenge 9: Determine New Extensions. Especially with the smaller scales of our subjects, driving innovation is still the responsibility of platform vendors. Our subjects devise and develop new extensions as follows.

Extensions Through Community Request: The **SE Platform** collects extension requests from customers. If a request suits the platform evolution plans, the vendor creates and offers the respective extensions to all customers. However, the platform vendor explicitly refuses to develop extensions that are specific to individual customers and delegates these efforts back to the customers. Our interviewee also emphasized that a clear expectation management is needed: **A:** *clear communication what [the customer] need[s], what the platform can deliver, and how much effort it would be to match the needs with what the platform can provide [is required].*

Extensions Through Prototype Feedback: The **IoT Platform** also collects user requests, but has a strong focus on proactively developing extensions: The vendor performs a prototypical development of extensions and examines their adoption potential. Upon sufficient interest, the prototype is developed into a full extension; otherwise development is discontinued.

Extensions Through Customer Request: The **VM Platform** creates new extensions for interfacing with other tools exclusively on request. However, platform customers are proactively inquired about their development practices in order to determine external engineering tools that are prospective candidates for interfacing. Furthermore, the need for creating full-fledged extensions is reduced by providing the scripting mechanism for smaller (mainly UI-related) extensions.

5.2.5 Challenge 10: Sustain Strategic Advantage. In the case study, we identified various, technology-related strategies how our platforms strive to exceed competitors.

Focus on Stability: The **SE Platform** vendor accumulated significant expert knowledge by engineering and operating the platform. This knowledge helps to constantly improve platform capabilities but also poses an entry barrier to competitors that consider launching competing platforms. In addition, the platform's strong focus on stability and reliability through automated test suites, failure capture-and-replay mechanisms (with resulting regression tests), and backward binary compatibility are strong advantages over competing platforms in the stability-focused domain. The decisions to maintain the existing test suite even with the new C# interfaces through generative procedures and to handle API evolution through *version-option structures* respect this focus on stability (cf. Sec. 4.3.1).

Focus on Variety: For the **IoT Platform**, the wide variety of IoT-related functionality is perceived as a significant advantage over competitors. Furthermore, the structuring into services allows freedom in composing the offered functionality, which is essential for the IoT domain where any prescribed general control flow would limit the application development. The re-engineering of the tools into micro-services when opening the platform further supported these advantages.

Focus on Openness: The **VM Platform** has a general policy of being open with respect to the information they provide. This is perceived as an essential success criterion for their stable market position: **C:** *I think a large part of acceptance [of the platform] is due to the possibility for individual extensions.* The platform opening further strengthened this advantage over competitors, easing the interoperability of external engineering tools with the platform. As further advantage for the platform vendor, it empowers external tool developers to proactively connect their tools without depending on the platform vendor.

6 THREATS TO VALIDITY

External Validity. Our survey participants cover a wide range of domains and companies; the interviews have been conducted on successfully opened platforms of different domains, technologies, and maturity levels. Each challenge is confirmed by data of multiple companies. The reported solutions are more individual and vary depending on the opened-up systems and their technologies.

Internal Validity. Participants might have given answers that do not fully reflect reality as they were recorded. To address this, we guaranteed anonymity and assured the interviewees that we will seek for feedback on conclusions to avoid misunderstandings. The survey also covers a broad range of roles, which avoids that drawn conclusions are limited to the participants' particular perspectives. The interviews were performed with participants having a strong technical background and substantial industrial experience. Thus, our work provides both a general perspective on platform opening as well as in-depth insights with respect to technical issues.

Construct Validity. To limit the effect of potentially subjective statements by participants, the reported challenges are based exclusively on statements affirmed by more than one interviewee and by several survey responses. Furthermore, to mitigate different interpretation of questions by researchers and participants, survey and interviews started with introductory explanations about the notions of platform and platform opening. Regarding completeness, survey and interview participants were provided the opportunity to add challenges and solutions not covered by the devised questions. Finally, we used violin plots to analyze the Likert-scale questions by interpreting the Likert scale as a continuous scale, which is possible due to equal value distances.

Conclusion Validity. Our qualitative data analysis depends on our interpretation. The main classification was performed by one author, but results were cross-checked by at least two others. As described in Sec. 2, we used recommended methods to improve validity, such as triangulation and carefully formulating conclusions.

7 RELATED WORK

Many studies on open software platforms and their surrounding software ecosystems exist. However, to our best knowledge, none focuses on opening of small and medium-scale industrial platforms.

Anvaari et al. [2] investigate to what extent mobile-app platforms allow extensions to interact with the platform. Their work is

orthogonal to ours: they focus on extension mechanisms in large mobile-app platforms; we focus on the opening process (how mechanisms are introduced) in smaller platforms from various domains.

Bosch [4] describes the general procedure of migrating a closed product line to an open ecosystem. He emphasizes that the opening process has to be managed. We complement his work by providing insight into detailed challenges and solutions for technical aspects of the opening. Bosch also provides a classification of ecosystems according to which all our platforms are *application-centric*.

Jansen [13] describes the centrally steered process of a single company for opening eleven of their product lines for extensions. We extend upon this work, as we investigate platforms of different vendors, from different domains and with different characteristics regarding challenges to overcome and liberty for solutions to apply.

Schultis et al. [15] study modes of collaboration among partners and resulting architecture challenges for internal software ecosystems. They focus challenges that arise in ecosystems within large-scale organizations, while we complement their work by focusing challenges that arise in ecosystems involving external partners.

Bosch [5] discusses architectural challenges for opening platforms (which form an ecosystem) and provides basic hints how to deal with them. We provide concrete practical solutions for these challenges and extend his list of challenges with many more.

Bosch et al. [6–8] analyze collaboration practices within software ecosystems forming around an open platform. Yet, they are not concerned with the process of platform opening.

Jansen et al. [14] conduct a case study to evaluate their open software enterprise model to assess the degree of openness of software platforms. They strive for encouraging platform vendors to open their platforms, which is complementary to our work where we elicit challenges and solutions in the process of platform opening.

Wnuk et al. [17] conducted a case study identifying bridges and barriers to engage in an ecosystem around a video surveillance systems. While it analyzes the success and blocking factors for further engagement of ecosystem partners, it does not explicitly address the process of opening up the platform.

Dal Bianco et al. [9] conduct an industrial case study on third-party developer experience, particularly on the role of "platform boundary resources" in exposing the platform architecture. Their work emphasizes the need for a platform to interface with its extensions through specific APIs as presented in our list of challenges.

Hanssen [11] studies a product-line organization and its transition towards an ecosystem. He focuses on change in collaboration practices and partner relationships as a consequence of an emerging ecosystem, while we focus on technical challenges and solutions.

Espinha et al. [10] investigate challenges for migrating applications to new versions of web APIs, commonalities in the evolution policies for web APIs, and the impact on the applications' source code when APIs start to evolve, which complements our insights for the backward-compatibility challenge and its solutions.

8 CONCLUSION

We presented a survey and a case study on the platform-opening process in small and medium-scale industrial platforms. We identified a range of successfully opened platforms that exhibit characteristics and goals that are different from those of the large and well-known platforms typically studied in the literature, such as Android, Eclipse, or the Linux kernel. By triangulating results from a questionnaire and interviews, we elicited ten challenges together with individual solutions related to the opening process and the longer-term endeavor of sustaining an open platform.

We learned that the inspected platforms put primary emphasis on the platform and extension quality. Large numbers of contributions are considered lower in priority. Specifically, an attitude towards "quality over quantity" for their extensions confirms the overall characteristics and goals of the subject platforms, which are different from the large platforms, which strongly encourage large numbers of extensions to foster their platforms' success [3].

The opening processes seemed diverse, but we found similar activities reflected in the five challenges related to opening, including API (re-)engineering or providing techniques for backward compatibility (or not allowing breaking changes at all). The solutions were mostly pragmatic, such as the version-option structures of the SE Platform to cope with backward compatibility. Interestingly, challenges such as modeling the ecosystem were less relevant.

In the future, we plan to assemble process templates from the identified common procedures to guide the opening processes according to clear engineering principles. In fact, we believe that the majority of the differences in the opening strategies results from the lack of such templates.

REFERENCES

[1] Online Appendix. http://gsd.uwaterloo.ca/openPlatformStudy.
[2] M. Anvaari and S. Jansen. Evaluating Architectural Openness in Mobile Software Platforms. ECSA, 2010.
[3] T. Berger, R.-H. Pfeiffer, R. Tartler, S. Dienst, K. Czarnecki, A. Wąsowski, and S. She. Variability Mechanisms in Software Ecosystems. *Information and Software Technology*, 56(11):1520–1535, 2014.
[4] J. Bosch. From Software Product Lines to Software Ecosystems. In *SPLC*, 2009.
[5] J. Bosch. Architecture Challenges for Software Ecosystems. In *ECSA*, 2010.
[6] J. Bosch and P. Bosch-Sijtsema. Coordination Between Global Agile Teams: From Process to Architecture. In D. Łmite, N. Moe, and P. Ågerfalk, editors, *Agility Across Time and Space*. Springer, 2010.
[7] J. Bosch and P. Bosch-Sijtsema. From Integration to Composition: On the Impact of Software Product Lines, Global Development and Ecosystems. *Journal of Systems and Software*, 83(1):67–76, Jan. 2010.
[8] 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*. Springer, 2010.
[9] V. Dal Bianco, V. Myllarniemi, M. Komssi, and M. Raatikainen. The Role of Platform Boundary Resources in Software Ecosystems: A Case Study. In *WICSA*, 2014.
[10] T. Espinha, A. Zaidman, and H. G. Gross. Web API Growing Pains: Stories from Client Developers and their Code. In *CSMR-WCRE*, 2014.
[11] 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.
[12] G. K. Hanssen and T. Dybå. Theoretical Foundations of Software Ecosystems. In *IWSECO*, 2012.
[13] S. Jansen. Opening the Ecosystem Flood Gates: Architecture Challenges of Opening Interfaces Within a Product Portfolio. In *ECSA*, 2015.
[14] S. Jansen, S. Brinkkemper, J. Souer, and L. Luinenburg. Shades of Gray: Opening up a Software Producing Organization with the Open Software Enterprise Model. *Journal of Systems and Software*, 85(7):1495–1510, July 2012.
[15] K.-B. Schultis, C. Elsner, and D. Lohmann. Architecture Challenges for Internal Software Ecosystems: A Large-Scale Industry Case Study. In *FSE*, 2014.
[16] C. Seidl and U. Aßmann. Towards Modeling and Analyzing Variability in Evolving Software Ecosystems. In *Proceedings of the 7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, VaMoS'13, 2013.
[17] K. Wnuk, P. Runeson, M. Lantz, and O. Weijden. Bridges and Barriers to Hardware-dependent Software Ecosystem Participation - A Case Study. *Information and Software Technology*, 56(11):1493–1507, Nov. 2014.