Patterns of Store-oriented Software Ecosystems: Detection, Classification, and Analysis of Design Options

Bahar Jazayeri, Paderborn University, Paderborn, Germany
Olaf Zimmermann, University of Applied Sciences of Eastern Switzerland, Rapperswil, Switzerland
Jochen Küster, Bielefeld University of Applied Sciences, Bielefeld, Germany
Gregor Engels, Paderborn University, Paderborn, Germany
Daniel Szopinski, Paderborn University, Paderborn, Germany
Dennis Kundisch, Paderborn University, Paderborn, Germany

Software companies nowadays create ecosystems of users and third-party providers around their platforms. They often provide online stores so that the third-party developments can be exposed to users directly. The resulting ecosystems differ significantly from each other in their architectural designs because their providers differ in terms of business goals and contexts. Until now, this architectural diversity and rationale behind it are not well-understood. Therefore, it is not clear which software features contribute to ecosystem's success with respect to certain business goals and context. This hinders systematic creation of ecosystems in the future. Thus, decision-making becomes too risky; for future ecosystem providers, which may lead to creation of inefficient ecosystems that lack critical features, and for third-party providers to rely on ad-hoc choices while deciding on suitability of an ecosystem for their future career. In this paper, we introduce three design patterns for store-oriented software ecosystems by classifying the design decisions, business goals, and context of 111 store-oriented software ecosystems. Each design pattern provides an architectural solution to achieve a different business goal while supporting a different context. We discuss how the design patterns are applied together in order to achieve more business goals. Our work supports ecosystem and third-party providers by sharing practice-proven architectural solutions, helping them to take informed architectural decisions and reduce technical risks.

Categories and Subject Descriptors: [**Software and its engineering**]

ACM Reference Format:

Bahar Jazayeri, Olaf Zimmermann, Jochen Küster and Gregor Engels. 2018. Patterns of Store-oriented Software Ecosystems: Detection, Classification, and Analysis of Design Options HILLSIDE Proc. of Latin American Conf. on Pattern Lang. of Prog. 25 (November 2018), 14 pages.

1. INTRODUCTION

Software ecosystems have become an emerging architectural approach for many companies to grow. The term *ecosystem* is inspired from ecological ecosystems that are the result of an interplay between organisms as well as interactions with a physical environment. Comparably, a *software ecosystem* consists of third-party providers, who are external to an enterprise, in service to a community of users, while they interact on the basis of a software platform [Bosch 2009]. For instance, Apple Inc. is the provider of an ecosystem in which independent developers provide mobile Apps to users of the iOS platform. Recently, marketing third-party solutions using online stores like Apple App Store has become a significant competitive advantage whereas many ecosystems comprise such stores [West and Mace 2010]. We call these ecosystems *store-oriented software ecosystems*. While prominent examples are mobile App ecosystems with millions of users, further ecosystems have been created in many other

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing"(CRC 901).
 A preliminary version of this paper was presented in a writers' workshop at the 12th Latin American Conference on Pattern Languages of Programs (SLPLoP). SLPLoP'18, November 20-23, Valparaíso, Chile. HILLSIDE 978-1-941652-11-4

application domains like cloud computing (e.g., Citrix and Salesforce) and Internet of Things (IoT) (e.g., IFTTT[1] and Stringify[2]) [Jazayeri and Schwichtenberg 2017].

To fulfill third-party providers' requirements and generally to succeed, existing ecosystems include different sets of software features. For instance, in massive ecosystems like Apple with millions of Apps, rating and ranking features are vital to the ecosystem's success [Martin et al. 2017]. However, in ecosystems of trusted partners like Citrix, third-party solutions are promoted by ecosystem providers instead of being rated. Until now, this architectural diversity and rationale behind it are not well-understood. Therefore, it is not clear which software features contribute to ecosystems' success. Thus, future ecosystem providers can not learn from the existing ones in order to decide on suitability of an architectural design with respect to their goals. A source of the architectural diversity is diversity in business decisions of ecosystem providers [Manikas and Hansen 2013]. Moreover, a great part of business decisions is derived from the orginizational context of providers, which adds another key source of diversity [Jansen et al. 2009]. Among others, company size, domain criticality, and commerciality are often identified as the most influential contextual factors. So far, existing literature such as studies on variability mechanisms (e.g., [Berger et al. 2014]) and empirical studies of existing ecosystems (e.g., [Van Angeren et al. 2011]) have treated business decisions, context, and software architecture separately. Therefore, their relation is not understood well yet. Furthermore, best practices and patterns of enterprise applications (e.g., [Fowler 2002]) have already been studied extensively. However, research on well-developed designs of software ecosystems is still in its infancy.

A holistic overview of existing ecosystems including business, context, and software aspects gives insight to the architectural diversity and rationale behind them. In our previous work [Jazayeri et al. 2018a], we detect three main design options of store-oriented software ecosystems by classifying 111 ecosystems from a business viewpoint using a variability model in [Jazayeri et al. 2017]. The variability model comprises key design decisions such as *openness*, *fee*, *choice of third-party providers*, and *knowledge sharing*. As a result, each design option represents a group of ecosystems that have common design decisions while contributing to a main business goal. Accordingly, these questions still remain unanswered: *What are the relations between these design options and the contextual factors? And, how to make this knowledge reusable for future use?*

In this paper, we provide a comprehensive view by developing three design patterns, each representing a design option of store-oriented software ecosystems. Design patterns are known to be practice-proven solution templates for recurring problems [Meszaros and Doble 1997]. Using patterns facilitates communication of knowledge among researchers and practitioners. We develop the patterns of this paper by a) systematically identifying the context of the design options, b) presenting the design options using a pattern structure in [Wellhausen and Fießer 2012], and c) providing further exemplary real-world ecosystems. In this context, our work makes two main contributions:

A) We introduce three architectural design patterns for store-oriented software ecosystems. The patterns provide insights into ecosystem architecture and its relation to ecosystem provider's business decisions and context, as well as forces and consequences in terms of business goals and quality attributes.

B) We analyze the patterns with respect to how they are related and their known uses. Results show that existing ecosystems mostly combine *Resale Software Ecosystem* and *Open Source Software-based Ecosystem* patterns to achieve *business scalability* while *enhancing innovation*.

Our work supports prospective ecosystem providers by sharing practice-proven architectural solutions so that they can right away decide on an ecosystem architecture, which fits to their business needs. Existing ecosystem providers can use the patterns to knowledgeably transform the architecture of their ecosystems. Moreover, using the patterns, third-party providers will be able to compare ecosystems and to decide on the most suitable one before entering an ecosystem. This in turn helps the ecosystem and third-party providers to take more informed architectural decisions and reduce technical risks in the future. Supplementary material including a complete list of ecosystems can be found in our technical report [Jazayeri et al. 2018b].

[1] `https://ifttt.com/` [2] `https://stringify.com/`

2. DESIGNS PATTERNS OF STORE-ORIENTED SOFTWARE ECOSYSTEMS

In our previous work [Jazayeri et al. 2018a], we detected three major design options of store-oriented software ecosystems. In this section, we describe these design options as patterns. The three patterns, i.e., *Resale Software Ecosystem*, *Partner-based Ecosystem*, and *Open Source Software (OSS)-based Ecosystem*, provide alternative ways to create store-oriented software ecosystems; Each pattern helps to overcome different forces and to achieve different business goals and quality attributes.

Accordingly, *Resale Software Ecosystem* supports ecosystem providers to get control over the third-party developers by facilitating ecosystem membership, developer's independence, and discoverability of high quality extensions. While third-party developments are known under different terminologies like plug-ins, add-ons, and Apps, in this paper, we refer to them as *extensions*. Ecosystem provider is a large company and involves a mass number of independent developers in software development. The developers are in completely separate teams. After the extensions are developed, they are sold several times to a mass number of users on a store. An example of this pattern is the ecosystem that belongs to the Apple Inc. and is built around the iOS and MacOS platforms and the Apple App Store.

Partner-based Ecosystem can be used to grow an industrial and complex software system to a new sector while enhancing commerciality. Ecosystem provider involves third-party providers only by establishing partnerships. Different customization of openness policies helps the ecosystem provider to protect the intellectual property. Extensions on the store are often labeled as *tested* or *validated*. The ecosystem provided by Citrix Inc. around Citrix platforms and the Citrix Ready Marketplace is an example of this pattern in real-world.

OSS-based Ecosystem aids to attract developers of open source software in order to cost-effectively create an ecosystem around an open source platform. Ecosystem provider is a foundation comprising several software companies or volunteers. The developers are non-commercially motivated, e.g., to gain reputation within a community or to extend the platform for their own purposes [Hanssen 2012]. While revenue generation is not generally high, granting the developers access rights to the code opens the ecosystem for innovative extensions. The ecosystem around Mozilla Firefox and its store, i.e., Firefox Add-ons, is an example of this pattern.

Figure 1 sketches the relations between the three patterns. Both *Partner-based Ecosystem* and *OSS-based Ecosystem* can be evolved to *Resale Software Ecosystem*. Additionally, these two patterns can be a building block of each other, which leads to the creation of ecosystems of ecosystems. In this paper, we describe the patterns by using the knowledge on methodical pattern development in [Wellhausen and Fießer 2012] [Buschmann et al. 2007, chap. 1]. Accordingly, patterns consist of these standard sections:

Context defines a situation in which one can apply a pattern. We identify context of the patterns by investigating ecosystems of our list with respect to the following contextual factors: **Company size**, **market size**, **domain criticality**, and **commerciality**. For company size, we refer to the number of employees in an enterprise. For

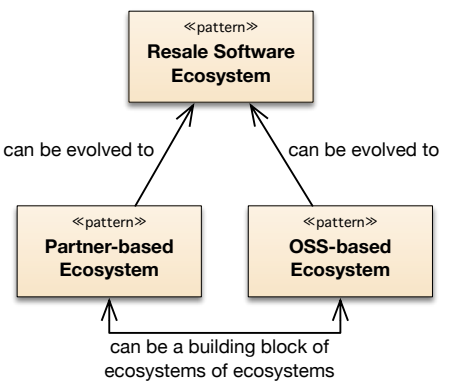


Fig. 1: Relationships between the architectural design patterns

Quality Attribute	Description
Productivity	The ecosystem shortens time-to-market from the development to the time that the extensions are made available for sale.
Sustainability	The ecosystem should be safe from external threats. Successfully confronting such threats has a long-term impact on ecosystem's success during its evolution.
Robustness	On the managerial level, it is the survival degree of ecosystem's participants either in relation to other ecosystems or over time.
Interoperability	From the perspective of enterprise architecture, the ecosystem should help an enterprise to minimize preparatory efforts to make a new relationship with other companies.
Modifiability	Degree to which the source code of software platform can be modified without introducing defects or degrading quality.
Creativity	The capability of an ecosystem to accommodate extensions with diverse characteristics like use case, programming language, execution environment.

Table I. : Performance drivers in software ecosystems *[Ben Hadj Salem Mhamdia 2013]*

market size, we count the number of extensions on the stores. Domain criticality determines whether software failure is dangerous to human lives. To decide on criticality of a domain, we refer to its application domain. For instance, we consider enterprise software to be a non-critical domain whereas domains like safety and security are the critical ones. Additionally, commerciality defines the degree of protecting intellectual property. We use thought bubbles to concisely refer to the contexts in Figures 2(a), 3(a), and 4(a). Appendix A.2 provides detailed information on the choice of contextual factors.

Problem is a difficulty to overcome by a pattern. In case of the patterns in this paper, the problem is a managerial difficulty that an ecosystem provider would solve while creating an ecosystem.

Forces identify why the problem is difficult to solve. This includes risks when the pattern is not applied. Generally, **market growth** and **cost** are the two main top-level forces that make software ecosystem an architectural choice for companies to open up their platforms to third-party providers [Bosch 2009]. Furthermore, high performance of software ecosystems are associated with several quality attributes as described in Table I [Ben Hadj Salem Mhamdia 2013].

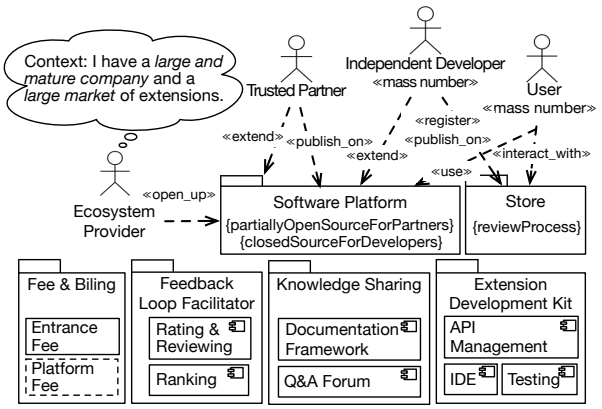
Solution resolves the problem and its associated forces. In this paper, each solution section proposes an arrangement of actors and choices of software features that help to solve an architectural problem at the managerial and governance level. We call such a solution an *architectural landscape*. ROZANSKI and WOODS [Rozanski and Woods 2012, p. 254] propose the term *architectural landscape* as an abstraction of the system, which denotes key building blocks of the architecture and their relations while concealing the details. This abstraction is particularly crucial for stakeholders with different viewpoints during problem-solving. We derive the architectural landscape of the patterns from the common design decisions for each design options in [Jazayeri et al. 2018a] and outline them using the UML notation in Figures 2(a), 3(a), and 4(a). Appendix A.1 provides definitions and classification of the design decisions.

Consequences discuss how the solution resolves the forces. This will be the quality attributes (cf. table I) that are achieved or degraded as a result of applying the patterns.

Examples are concrete instances that conform to a pattern. An ecosystem conforms to the patterns of this paper if its architectural design decisions comply with the patterns, e.g., in terms of types of third-party providers and software features, and their relations.

Known uses provide information on popularity of the patterns and their typical application domains.

Related patterns describe how a pattern is related to other patterns. Specifically, we elaborate on the pattern map in Figure 1. In addition, we discuss the relation between the patterns in this paper and existing patterns.



(a) Architectural Landscape adopted from *[Jazayeri et al. 2018a]*

	Pattern Characteristics		Adobe
Context	**Company Size**		Large (15,500+ employees)
	Market Size		Large (2,500+ extensions)
	Domain Criticality		Graphic & multimedia software
Design Decisions	**Extender**	**Trusted Partner**	Esri, Microsoft, Magento, etc. Partner programs
		Independent Developer	Mass number of producers
	Feedback Loop Facilitator	**Rating & Reviewing**	Star rating system, Reviewing
		Ranking	Lists of popular extensions
	Knowledge Sharing	**Documentation Framework**	Producer / partner portals
		Q&A Forums	Coding Corner, Adobe Forums
	Extension Development Kit	**API Management**	Photoshop and InDesign SDKs, etc.
		IDE	Eclipse Extension, IntelliJ IDE, etc.
		Testing	Extension Manager, A/B testing, PlayerDebugMode, etc.

(b) Exemplary Ecosystem: Adobe

Fig. 2: Resale Software Ecosystem: An architectural solution to achieve business scalability

2.1 Resale Software Ecosystem

Context. Ecosystem provider owns a large enterprise and a large market of extensions. The platform and extensions are not safety-critical.

Problem. The ecosystem provider wants to have control over quality of the extensions while opening the platform to a high number of independent developers. Thus, these questions arise: Firstly, **how to manage the ecosystem membership for the mass number of developers while giving the developers a high degree of independence?** Secondly, **how to ensure discoverability of high quality extensions among the high number of offers?**

Forces. In a lack of any solution, the users may receive low quality or malfunctioning extensions. This adversely affects user experience and consequently the ecosystem provider's reputation. Another risk is that the developers are not supported with suitable software features that ensure their independence. Thus, they may fail to extend the platform and abandon the ecosystem. So, they might walk away and start coding extensions for a competitor's ecosystem (or contribute to open source projects). With this respect, the forces can be related to the most important quality attributes of software ecosystems mentioned earlier:

- **Sustainability**
- **Robustness**
- **Productivity**

Solution. **To manage ecosystem membership, the independent developers need to register in the ecosystem and pay entrance fee.** Platform fees vary depending on an ecosystem provider's strategies. Figure 2 shows the architectural landscape. The platform is closed to the developers and partially open to the partners. By registering in the ecosystem, the developers are obliged to follow a wide range of policies that are regulated by the ecosystem provider. Examples are legal policies like licensing and technical policies like choice of programming language and execution environment.

To support developers' independence, the developers are provided with a toolchain for the whole software life cycle. This includes an API management to access platform's functionality, Integrated Development Environment (IDE), and testing features including emulators and simulators to imitate an execution environment. An example of existing toolchains is Xcode in the Apple ecosystem. Furthermore, using Q&A forums, the users and developers trigger discussions on different topics and spread the knowledge throughout the ecosystem.

To ensure discoverability of high quality extensions among the available offers, rating & reviewing and ranking features are used. A ranking feature generates different lists of high quality extensions such as lists of popular / featured / newly published extensions. In addition, the ecosystem provider might proactively avoid low

quality extensions entering the ecosystem by conducting a review process before the extensions are published on the store. Static code analysis is an example of such a review process to ensure maleware-free extensions.

Consequences. By successfully accommodating the high number of developers and ensuring discoverability of high quality extensions, the ecosystem achieves **productivity**, **sustainability**, and **robustness**. In this case, **business scalability** is the high-level business goal addressed by this pattern [Jazayeri et al. 2018a]. However, establishing a working store and maintaining it in terms of extension visibility impose extra **costs** on the ecosystem provider. Therefore, we suggest to apply the pattern only if the ecosystem provider owns a large market of high quality extensions or there are strategic partners, who can promote the ecosystem by their contributions.

Examples. Adobe Systems (shortly Adobe) provides a product line of graphic software platforms such as Dreamweaver, Photoshop, and InDesign. Figure 2(b) presents the context and design decisions of the Adobe ecosystem. Adobe is a large company with a large market of more than 2,500 extensions. The extensions are not safety-critical. In addition to the trusted partners like Esri, a mass number of independent developers, namely producers, extends the platforms. Entering the ecosystem is subject to an entrance fee. Platform SDKs, e.g., Adobe Photoshop CC and InDesign SDKs, and different IDEs for different programming languages enable the partners and producer to develop independent of Adobe. Furthermore, Extension Manager, A/B testing, and PlayerDebugMode are the examples of testing frameworks in the ecosystem. Adobe Exchange[3] is the online store, where the extensions are published. Star rating and textual reviewing features are used to expose quality of extensions. Subsequently, a ranking feature generates a list of popular extensions on the store. Furthermore, Adobe Partner Portal[4] as well as Producer Portal[5] provide the extenders with necessary documentation. In addition, Coding Corner[6] and Adobe Forums[7] are the Q&A forums of the ecosystem.

Known uses. 37% of ecosystems in our list conform to *Resale Software Ecosystem*. Operating system, mobile application, and web browser are typical examples of application domains. Further ecosystems applying *Resale Software Ecosystem* are Apple, Salesforce, and Esri. More examples can be found in our technical report [Jazayeri et al. 2018b].

Related patterns. Quality of extensions can systematically be managed by establishing a governance model or providing sandboxes as a controlled medium as described by *Manage Complements*. In addition, defining coding conventions assists developers to adhere to quality standards [Weiss and Noori 2013].

2.2 Partner-based Ecosystem

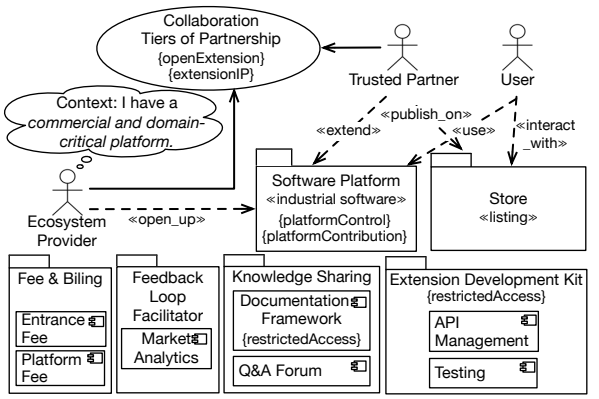
Context. High commerciality and criticality are the major characteristics of software platform. The platform is complex (and often safety-critical) software for industrial sectors such as supply chains, aerospace, and healthcare.

Problem. Ecosystem provider wants to grow the platform to a new industrial sector. Thus, reasonable software and hardware resources as well as human expertise are needed to develop solutions for that sector. Given the platform criticality, the question is: **How to ensure high quality of future extensions in the new sector?** Furthermore, considering the platform commerciality, **How to control platform openness while protecting the intellectual property?**

Forces. It is too costly for the ecosystem provider to supply all resources or too risky in terms of failing to deliver a certain degree of quality. In addition to the high-level forces, i.e., market growth and cost, the important forces are:

- **Productivity**
- **Robustness**

[3] `https://www.adobeexchange.com` [4] `https://www.adobe.com/partners.html` [5] `https://technologypartners.adobe.com/home.html`
[6] `https://forums.adobe.com/community/coding-corner` [7] `https://forums.adobe.com`



(a) Architectural Landscape adopted from *[Jazayeri et al. 2018a]*

	Pattern Characteristics		Symantec
Context	**Domain Criticality**		Cyber security
	Commerciality		Commercial platforms & extensions, Costly licensing
Design Decisions	**Extender**	**Trusted Partner**	Fujitsu, Deloitte, Amazon, etc. Symantec partner programs
	Openness	**Closed**	Different policies for acquisitions, strategic partners, and partner programs
	Fee	**Entrance Fee**	Partner Licensing
		Platform Fee	On-premise payment
	Feedback Loop Facilitator	**Market Analytics**	Planning, consulting, training Partner Marketing Planner tool
	Knowledge Sharing	**Documentation Framework**	PartnerNet including several partner portals
		Q&A Forums	Symantec Connect
	Extension Development Kit	**API Management**	Platform SDKs, e.g., Symantec PGP SDK
		Testing	System testing, Acceptance testing on user behalf

(b) Exemplary Ecosystem: Symantec

Fig. 3: Partner-based Ecosystem: An architectural solution to enhance profitability

- **Interoperability**
- **Modifiability**

Solution. **To ensure high quality of extensions in the new sector, the ecosystem provider establishes partnerships only with the providers, i.e., domain experts, who already possess the required resources and expertise in that sector**. The architectural landscape is presented in Figure 3(a). The ecosystem provider and partners collaborate in developing and marketing joint solutions as well as conducting road map sessions. They pursue the same user segment and together compete for market success. In case of highly commercial extensions, acceptance tests are performed on user behalf. Such extensions are distinguished on the store by the labels like *tested* or *validated*. In contrary to *Resale Software Ecosystem*, in *Partner-based Ecosystem*, rating & reviewing features do not play a critical role. The extensions are rather marketed using market analytics. This mainly transforms the store to a listing. The market analytics features, e.g., customer relationship management (CRM) and repository mining, enable the partners to track user satisfaction in the ecosystem.

To control the platform openness while protecting the intellectual property, the ecosystem provider specifies a range of openness policies and realizes them by defining partner programs and monetizing resources. Deciding on different customization of openness policies specifies the degree of openness. BOUDREAU [Boudreau 2010] introduces two major types of openness policies, i.e., *platform openness* and *complement openness* (in this paper, we use the term *extension* for *complement*). Both types of openness are defined in a more fine-granular way: *Platform openness* comprises *platform control* and *platform contribution*. *Platform control* specifies the situation, when third-party providers have equity ownership of the platform, and *platform contribution* is when the third-party providers only contribute to the development. Furthermore, *extension openness* is defined as *open extension* and *extension intellectual property (IP)*. *Open extension* is the policy to grant licenses to the third-party providers, whereas *extension IP* is a policy to share the intellectual property with them.

Different openness policies can be realized by forming various partner programs. The programs with tighter partnership contain less entrance barriers and more openness. For instance, while platinum partners may have *platform contribution* but not *platform control*, they need to regularly prove a minimum amount of revenue generation in the sector. However, gold partners that have *platform control* are not subject to such a requirement.

Furthermore, the openness policies are realized by monetizing resources and demanding fees. An API management specifies a partner's access permission and corresponding fees. Furthermore, the partners need to pay entrance and platform usage fees on a periodic basis. Another facet of the monetization is documentation frameworks. Such frameworks are a part of the partner programs and only accessible to the partners. However, Q&A forums are publicly accessible.

Consequences. By successfully growing to the new sector, the ecosystem provider enhances **productivity** while **saving costs** of supplying new resources. Establishing collaborations with the partners enhances **robustness** and **interoperability**. However, it demands high strategic movements and the efforts to make the ecosystem profitable

for the partners. The high-level business goal addresed by *Partner-based Ecosystem* is **profitability** [Jazayeri et al. 2018a]. The ecosystem provider succeeds in consolidating commerciality by monetizing the platform. However, the entrance barriers degrade **modifiability** and **creativity**. Therefore, the pattern is not recommended for the situation, where ecosystem's growth depends on developers' creativity, similar to what is known from open source projects.

Examples. Symantec Corporation is a provider of cyber security services like email, endpoint, and cloud security. Extensions are listed online[8]. Figure 3(b) presents the design decisions and context of the Symantec ecosystem. Due to domain criticality, the ecosystem needs to provide highly reliable services. An example of acquisitions by Symantec is Norton that shares its intellectual property with Symantec (*Platform control*). Additionally, Symantec shares *extensions IP* with highly strategic partners like Fujitsu and Deloitte. Moreover, it provides partner programs to include other providers by granting them licenses (*open extension*). The platforms are closed source. Platform fees are associated with purchasing Symantec solutions. In addition to portfolio requirements, entering the partner programs requires an entrance fee. Global Systems Integrator[9] and Technology Integration[10] are examples of the partner programs. Knowledge sharing is enabled by PartnerNet[11], which is accessible to the partners. As a part of market analytics, a tool namely Partner Marketing Planner[12] gives partners a personalized marketing plan to enhance revenue generation. Furthermore, Symantec Connect[13] is a portal for users' and partners' forums. Functionality of the platforms is extendable using their SDKs. For instance, Symantec PGP SDK gives access to cryptographic functionality of the PGP platform.

Known uses. Partner-based Ecosystem is applied by 35% of ecosystems in our list. Examples of application domains are cloud computing, industrial design and simulation software, and security. Exemplary ecosystems are Citrix, SAP, and IFTTT [Jazayeri et al. 2018b].

Related patterns. Increasing growth of the partner community and store alongside each other is considered as a natural evolution of successful ecosystems [Plakidas et al. 2016]. Thereby, *Partner-based Ecosystem* can be evolved to *Resale Software Ecosystem* when the number of partners increases and the ecosystem provider includes rating, reviewing, and raking features to improve extension discoverability. In addition, by demanding registration, instead of a direct partnership, the ecosystem provider manages membership of the growing number of extensions. 5% of the ecosystems, e.g., Intuit QuickBooks[14], relate these two patterns.

2.3 Open Source Software-based Ecosystem

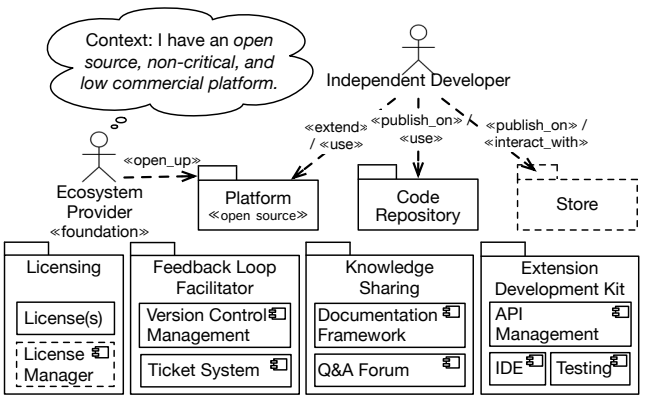
Context. A midsize to large market of extensions resides on a code repository. In addition, the platform is open source and commerciality is low. The platform and extensions are not critical to human lives.

Problem. Ecosystem provider is interested in winning contributions from skilled developers of open source software (FOSS) community. In addition to cost-saving reasons, an ecosystem provider might be willing to inform him- / herself about market directions and innovative trends through such developers. However, due to low commerciality, revenue is not generally returned to the ecosystem. Thereby, the question are: **How to save costs of developers by attracting developers of open source software?** And, **how to save cost of software?**

Forces. Since revenue generation is barely a motivation for the developers to contribute, hence, if the ecosystem provider fails to include the developers from the open source community, the platform will not grow. Essentially, the most important quality attributes are:

- **Cost of developers and software**
- **Modifiability**

[8] https://www.symantec.com/integration [9] https://www.symantec.com/partners/programs/global-systems-integrator
[10] https://www.symantec.com/partners/programs/technology-integration-partners [11] https://www.symantec.com/partners
[12] https://resource.elq.symantec.com/partner_resource_centre [13] https://www.symantec.com/connect/ [14] quickbooks.intuit.com/



(a) Architectural Landscape adopted from *[Jazayeri et al. 2018a]*

	Pattern Characteristics		Cloud Foundry
Context	**Market Size**		Large (4,000+ forks)
	Domain Criticality		Cloud computing
	Commerciality		Free entrance, Open platform
Design Decisions	**Extender**	**Trusted Partner**	CloudFoundry Foundation
		Independent Developer	Freelancer, Companies
	Openness	**Open**	CloudFoundry platforms, Extensions on GitHub
		License	Apache License. Cloud Foundry contributors' licenses
	Feedback Loop Facilitator	**Ticket System**	Issues as a GitHub feature
		Version Control Management	GitHub
	Knowledge Sharing	**Documentation Framework**	CloudFoundry Docs
		Q&A Forums	Public Q&A Forums
	Extension Development Kit	**API Management**	SDK on GitHub
		IDE	Language-specific IDEs
		Testing	Unit / integration / scalability / performance testing

(b) Exemplary Ecosystem: Cloud Foundry

Fig. 4: OSS-Based Ecosystem: An architectural solution to enhance innovation while saving costs

- **Creativity**

Solution. **Attractiveness of open source platforms is highly associated with modifiability of software components in two levels, i.e., governance and technical.**

At the governance level, this needs to be facilitated by giving third-party providers suitable ownership and decision rights to access the code. *License management* is one of the mostly used governance mechanisms in software ecosystems [Alves et al. 2017]. In general, a software license specifies use permissions and conditions for such permissions. Thus, license management becomes a way to control the protection of intellectual property. A well-balanced license, i.e., not too closed and not too open, is crucial for an ecosystem that is created around an open source software, because, the license specifies the interplay between the capability of open source development and the business model [Mizushima and Ikawa 2011]. Therefore, an additional complexity is to find a suitable licensing that gives the developers rights to use the code while supporting business model of another group of developers with commercial milestones.

While deciding on a suitable licensing, the ecosystem provider should consider conflict management and future access rights during ecosystem evolution. The choice of licensing needs to clearly answer the following questions without introducing any license conflict [Scacchi and Alspaugh 2012]: What are third-party providers' rights and obligations? Are the extensions already pertained to other licenses? Which part of the system can be evolved or replaced? What are dependencies between those parts and the rest of the system?

Management of license conflicts can be done manually by staff, similar to *Eclipse Legal Process*, which is an examination procedure performed by Eclipse foundation to prevent publication of code with any license other than Eclipse Public License (EPL). Moreover, the ecosystem provider might provide an automated environment to create and manage licenses including checking conflicts [Mizushima and Ikawa 2011].

At the technical level, the ecosystem should provide third-party developers with software features that are necessary to access, reuse, and develop software components collaboratively. Figure 4(a) illustrates the architectural landscape. Version control management and ticket system facilitate a feedback loop between the developers. A version control management like Apache Subversion supports forking, branching, and merging the code. Additionally, a ticket system like Jira[15] helps to track issues and bugs.

[15] `www.atlassian.com/software/jira`

To save costs of software, the ecosystem provider uses a wide range of free and open source resources and tools on the web. An example of such tools is the testing frameworks like Selenium[16].

Consequences. By attracting developers of open source software and by relying on resources of the FOSS communities, the ecosystem provider succeeds to **save cost**. High degree of openness and eliminating the entrance barriers like fees give the developers freedom in developing extensions. This ultimately enhances **innovation and creativity**. However, warranties and liability indemnity are not supported on user's behalf. This makes the ecosystem vulnerable to low quality extensions and threatens ecosystem's **sustainability**. In addition, achvieing **robustness** is a challenge, because the survival of open source platforms is generally subject to retaining contributors whereas the contributors are free to leave the ecosystem at any time [Yamashita et al. 2014].

Examples. Cloud Foundry Foundation is the provider of an OSS-based ecosystem. Figure 4(b) presents the design decisions and context of the Cloud Foundry ecosystem. Cloud Foundry[17] is the free and open source platform. Its services are non-critical, i.e., enterprise application. A large market of extensions with more than 4,000 forks exists on GitHub. IBM, SAP, and Google are the exemplary members of the foundation. In addition, other companies as well as freelancers extend the platform by directly accessing the source code. The platform is under an an Apache License. So, third-party providers have the right to use the software under the terms of this license. Whereas extensions are licensed based on the *Cloud Foundry contributors' licenses*. With this respect, the foundation has the right to use and re-license the extensions, but does not own copyright of those extensions. Furthermore, Cloud Foundry Docs[18] contains the technical documentation whereas Pivotal Knowledge Base[19] is the Q&A forum. Moreover, Slack[20] and StakOverflow are the examples of public knowledge sharing portals. The SDK resides on GitHub[21]. Moreover, a wide range of testing features for Cloud Foundry exists on GitHub, e.g.,unit / integration / service quality testing.

Known uses. 28% of ecosystems in our list are OSS-based ecosystems. Operating systems and software development are examples of the application domains. Further exemplary ecosystems are Apache Cordova, Ubuntu, and Zotero [Jazayeri et al. 2018b].

Related patterns. A pattern language by WEISS [Weiss 2018] provides an overview of engagement levels in open source businesses and identifies strategic decisions related to each level. The engagement levels start with *using* and *contributing*, and advance to *championing* and *collaborating*. Furthermore, two cross-cutting concerns, i.e., architecture and licensing, affect all levels. The pattern language captures fine-granular decisions that can be used to grow an open source business in a stepwise manner. For example, with regard to licensing, it suggests that a software owner, who possesses the full ownership of the code, can provide the same product under two different licenses, i.e., both open source and commercial (*Dual License*) or offer an enhanced version of the open source software as commercial (*Dual Product*).

Furthermore, *OSS-based Ecosystem* is related to the other patterns in this paper: Similar to *Partner-based Ecosystem*, *OSS-based Ecosystem* can also be evolved to *Resale Software Ecosystem*, e.g., by making high quality extension discoverable using rating, reviewing, and ranking features. This happens when an ecosystem provider offers a store for the extensions, in addition to providing a code repository. In some cases, the extensions on the store are free and open source such as Mozilla.org and LibreOffice.org. However, the ecosystem might generate profit by offering commercial extensions, despite the platform itself being open source. An example of this case is Eclipse Marketplace[22] that offers both free and commercial plug-ins.

Moreover, *OSS-based Ecosystem* and *Partner-based Ecosystem* can be a building block of each other once an ecosystem provider creates an *ecosystem of ecosystems*. It means different platforms from the same family of software are the basis for a different ecosystem. For instance, Pivotal is the provider of a *Partner-based Ecosystem*

[16] https://www.seleniumhq.org/ [17] www.cloudfoundry.org [18] https://docs.cloudfoundry.org/
[19] https://community.pivotal.io/s/communities [20] https://slack.cloudfoundry.org/ [21] https://github.com/cloudfoundry
[22] https://marketplace.eclipse.org

(created around Pivotal Web Services (PWS)) and an *OSS-based Ecosystem* (built around Cloud Foundry). PWS and Cloud Foundry are both products of the same software family, i.e., Pivotal software.

3. CONCLUSION

Many modern software companies create store-oriented ecosystems of third-party providers and users on top of their platforms; online stores serve as distribution channels for third-party developments. This architectural approach has been applied widely; however, the diversity of the existing ecosystem designs hinders prospective ecosystem providers to gain a sound overview of the existing designs and to understand what the best architectural design with respect to their business goals and organizational context is. In this paper, we present three architectural design patterns for store-oriented software ecosystems as: 1) *Resale Software Ecosystem*, 2) *Partner-based Ecosystem*, and 3) *Open Source Software-based Ecosystem*. Each pattern suggests an arrangement of human actors and choice of software features in order to solve an architectural problem related to ecosystem governance while fulfilling certain business goals and quality attributes.

This knowledge introduces ecosystem providers the practice-proven reusable designs and helps them to decide on when to apply any of these designs, or to transform their existing ecosystems. In addition, this should help third-party providers to decide on suitability of an ecosystem before entering it, by benchmarking existing ecosystems against the patterns, identifying key features of ecosystems, understanding enhanced and degraded quality attributes, and deciding on suitability of the ecosystems with respect to their goals. Therefore, they will be able to save efforts of participating in inefficient ecosystems that miss critical features. In the future, practical effectiveness of the patterns can be further evaluated by architects on real projects. Furthermore, several aspects inside the patterns of this paper can be elaborated separately as further patterns in the future, for example licensing or quality management. Developing a pattern language that fill the gaps and includes pattern sequences is a part of future research.

A. PROCESS OF PATTERN MINING

This appendix describes the process of mining the patterns from data that exists in real-world and transferring them to the patterns. To do so, we follow the guidelines provided by *KJ Method* [Scupin 1997], which is well-known in the pattern community to identify patterns by organizing qualitative data collected from various sources. The KJ method includes three main steps as follows: *Element mining* is a divergent approach to collect as much data as possible. *Clustering* is a convergent approach to discover relationship between the data. *Labeling* includes specifying potential patterns based on the clusters that are identified in the previous step. To identify our patterns, we use existing models as a basis to collect and later to cluster the data during the element mining and clustering steps. In the following, Sections A.1 and A.2 elaborate on these models and the results of clustering. Supplementary material can be found in our technical report [Jazayeri et al. 2018b].

A.1 Common Design Decisions

In our previous work [Jazayeri et al. 2018a], we collect a list of 111 ecosystems and classify them based on similarities in their design decisions by using a variability model that comprises both business and application viewpoints. The variability model [Jazayeri et al. 2017] is developed by using a research method based on the design science paradigm and from the material of existing ecosystems, e.g., websites and annual reports, and by conducting a rigorous literature review. The variabilities are expressed in form of *variation points* and *variants*. A variation point is the subject of a variability whereas a variant is the object of the variability, i.e., a concrete design decision. Accordingly, variation points are as follows: **Extender** defines who the third-party provider is. **Openness** specifies whether a platform is open source. **Fee & Billing** are the main costs of participating in an ecosystem. **Feedback Loop Facilitator** determines the software features that enable a positive feedback loop between users and extenders. In the context of markets, this happens when more users adopt a platform. Thus, the number of extenders increases [Holzer and Ondrus 2011]. **Knowledge Sharing** is to communicate ecosystem-specific

	Variation Point	Variant	Group 1: Resale Software Ecosystems	Group 2: Partner-Based Software Ecosystems	Group 3: OSS-Based Software Ecosystems
Business Viewpoint [Jazayeri et al. 2018]	**Extender** Who is the third-party provider?	**Trusted Partner**	Hardware / software suppliers, strategic partners	Hardware /software suppliers, strategic partners, system integrator, etc.	Foundation members, Strategic partners
		Independent Developer	Mass number of independent developers	–	High number of independent developers
	Openness Is the source code open?	**Open**	Fully or partially open libraries	–	Source code on a public repository, e.g., GitHub or SourceForge
		Closed	Fully or partially closed libraries	Closed source code. Entrance barriers: Static code analysis, Review process, Financial requirements	–
	Fee & Billing What are costs of participating in the ecosystem?	**Entrance Fee**	One time / periodic payment	Membership in partner programs / different payments for different partners	–
		Platform Fee	–	Different payments for platform editions. Monetized APIs	–
	Feedback Loop Facilitator Which software features enable a positive feedback loop between the users and extenders?	**Rating & Reviewing**	Binary rating, Scale rating (Stars, Sliders), Reviewing	–	–
		Ranking	Featured / popular / new extensions	–	–
		Market Analytics	–	User statistics, CRM, Marketing planer	–
		Version Control	–	–	Tools from FOSS community, e.g., SVN, Git, Mercurial, Perforce, etc.
		Ticket System	–	–	Test planing, bug tracking, notification interfaces, e.g., Jira
	Knowledge Sharing Using which software features enable knowledge?	**Documentation Framework**	Developer manuals. Wikis	Partner portal: Documentation of software frameworks	Manuals, wiki, public resources from open source software community
		Q&A Forums	Developer / user / idea forums	Partner portal	Public developer forum, e.g., StackOverflow
Application Viewpoint	**Extension Development Kit** Which software features enable extenders to extend the platform?	**API Management**	SDK, API Reference, Source code	SDK, API Reference	SDK, API Reference, Source code
		Integrated Development Environment (IDE)	Ecosystem-specific IDEs, e.g., Xcode, Android Studio, AWS tool kit	–	IDEs of free and open-source software (FOSS), e.g., Eclipse
		Testing	Crowd / Unit testing, Code review	Acceptance / System / Smoke Integration Testing	Unit / Integration / System Testing / Quality Check

Table II. : Common architectural design decisions and instances of three groups of ecosystems ("–" means that a variant is not realized in the architecture) *[Jazayeri et al. 2018a]*

knowledge among users or extenders [Grover and Kohli 2012]. Finally, **Extension Development Kit** is a set of software features that enable extenders to develop software on top of a platform. Examples of such features are software development kit (SDK) and integrated development environment (IDE). The result of the investigation and classification is three groups of ecosystems that have common design decisions as shown in Table II. Some cells are marked as "–", i.e., a decision is not realized by a group of ecosystems.

A.2 Common Contextual Factors

After identifying three groups of ecosystems, we extract the context of patterns by investigating the context of ecosystem providers for each group of ecosystems. To do so, we use a model by KRUCHTEN [Kruchten 2013] that comprises eight key contextual factors as size, stable architecture, business model (commerciality), team distribution, rate of change, age of system, criticality, and governance. The author uses the term *context* and argues that software projects become unique in terms of architectural practices mainly due to their different contexts. The model is not specific to software ecosystems and is generally defined for software projects, which are going to be agile and at the same time scale up. However, in a lack of a contextual model for software ecosystems, this model provides us with a solid basis to investigate the context of ecosystem providers. Among others, significance of three factors, i.e., **company size**, **domain criticality**, and **commerciality**, are supported by the literature on software ecosystems [Berger et al. 2014; Jansen and Cusumano 2013; Axelsson et al. 2014]. Although, here, the

use of the term *context* is slightly different than how the term is used in the pattern community (where context is a precondition for the application of pattern), investigating contextual factors of existing providers helps us to understand the linkage between these factors and different ecosystem architectures.

Furthermore, since software ecosystems include third-party providers, company size as an intra-organizational factor does not suffice to decide on the size of an ecosystem [Bosch 2009]. Therefore, we add **market size** as another contextual factor and measure it by counting the extensions on the stores. As suggested by DUC ET AL. [Duc et al. 2014], in case of ecosystems that are built around open source software, the number of forks is an indicator to decide on its size. For the interpretation of size, we use the scales provided by Gartner [SMB 2018], which is a well-reputed market observer firm. We identify the context of the patterns by identifying similar contextual factors inside each group of ecosystems. Table III shows the resulting common contextual factors.

Contextual Factor	Resale Software Ecosystem	Partner-based Ecosystem	OSS-based Ecosystem
Company size Small (1-99 employees) Midsize (100-999 employees) Large (1000+ employees)	Large companies	–	–
Market size Small (1-99 extensions) Midsize (100-999 extensions) Large (1000+ extensions)	Midsize to large markets of extensions	–	Large markets of extensions
Domain criticality Failure in extensions dangerous to human lives	No criticality	Industrial & safety-critical applications	No criticality
Commerciality The degree of protecting intellectual property	–	Commercial extensions, Monetized APIs, Entrance Fee	Free and open access to the platforms

Table III. : Common contextual factors and instances of three groups of ecosystems ("–" means that a factor can have any value)

ACKNOWLEDGMENTS

We thank our shepherd, Michael Weiss, for his valuable comments that have significantly helped to improve our work.

REFERENCES

Last access: January 2018. What Is SMB? - Gartner Defines Small and Midsize Businesses. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. (Last access: January 2018).

Carina Alves, Joyce Oliveira, and Slinger Jansen. 2017. Software Ecosystems Governance-A Systematic Literature Review and Research Agenda. In *19th Int. Conf. on Enterprise Info. Sys.*, Vol. 3. 26–29.

Jakob Axelsson, Efi Papatheocharous, and Jesper Andersson. 2014. Characteristics of Software Ecosystems for Federated Embedded Systems: A Case Study. *Information and Software Technology* 56, 11 (2014), 1457–1475.

Amel Ben Hadj Salem Mhamdia. 2013. Performance Measurement Practices in Software Ecosystem. *International Journal of Productivity and Performance Management* 62, 5 (2013), 514–533.

Thorsten Berger, Rolf-Helge Pfeiffer, Reinhard Tartler, Steffen Dienst, Krzysztof Czarnecki, Andrzej Wasowski, and Steven She. 2014. Variability Mechanisms in Software Ecosystems. *Info. and Soft. Tech.* 56, 11 (2014), 1520–1535.

Jan Bosch. 2009. From Software Product Lines to Software Ecosystems. In *Int. Conf. on Soft. Product Line*. CMU, 111–119.

Kevin Boudreau. 2010. Open Platform Strategies and Innovation: Granting Access vs. Devolving Control. *Management science* 56, 10 (2010), 1849–1872.

Frank Buschmann, Kelvin Henney, and Douglas Schimdt. 2007. *Pattern-Oriented Software Architecture: On Patterns and Pattern Language*. Vol. 5. John wiley & sons.

Anh Nguyen Duc, Audris Mockus, Randy Hackbarth, and John Palframan. 2014. Forking and Coordination in Multi-Platform Development: A Case Study. In *ACM/IEEE Int. Symp. on Empirical Soft. Eng. and Measurement*. ACM, 59.

Martin Fowler. 2002. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Varun Grover and Rajiv Kohli. 2012. Cocreating IT Value: New Capabilities and Metrics for Multifirm Environments. *Mis Quarterly* (2012), 225–232.

Geir K. Hanssen. 2012. A Longitudinal Case Study of an Emerging Software Ecosystem: Implications for Practice and Theory. *Journal of Systems and Software* 85, 7 (2012), 1455–1466.

Adrian Holzer and Jan Ondrus. 2011. Mobile Application Market: A Developer's Perspective. *Telematics and informatics* 28, 1 (2011), 22–31.

Slinger Jansen and Michael A. Cusumano. 2013. Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance. *Software ecosystems: analyzing and managing business networks in the software industry* 13 (2013).

Slinger Jansen, Anthony Finkelstein, and Sjaak Brinkkemper. 2009. A Sense of Community: A Research Agenda for Software Ecosystems. In *International Conference on Software Engineering (ICSE)-Companion Volume*. IEEE, 187–190.

Bahar Jazayeri and Simon Schwichtenberg. 2017. On-The-Fly Computing Meets IoT Markets – Towards a Reference Architecture. In *Int. Conf. on Soft. Arch. Companion Volume*. IEEE, 120–127.

Bahar Jazayeri, Olaf Zimmermann, Gregor Engels, and Dennis Kundisch. 2017. A Variability Model for Store-Oriented Software Ecosystems: An Enterprise Perspective. In *Int. Conf. on Service-Oriented Computing*. Springer, 573–588.

Bahar Jazayeri, Olaf Zimmermann, Gregor Engels, Jochen Küster, Dennis Kundisch, and Daniel Szopinski. 2018a. Design Options of Store-Oriented Software Ecosystems: An Investigation of Business Decisions. In *Int. Symp. on Business Modeling and Soft. Design*. Springer, 573–588.

Bahar Jazayeri, Olaf Zimmermann, Jochen Küster, and Gregor Engels. 2018b. *Patterns of Store-Oriented Software Ecosystems: Detection, Classification, and Analysis of Design Options. Dataset . https://www.overleaf.com/read/njqzqhsmvctk*. Technical Report.

Philippe Kruchten. 2013. Contextualizing Agile Software Development. *Journal of Software: Evolution and Process* 25, 4 (2013), 351–361.

Konstantinos Manikas and Klaus Marius Hansen. 2013. Software Ecosystems–a Systematic Literature Review. *Journal of Systems and Software* 86, 5 (2013), 1294–1306.

William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2017. A Survey of App Store Analysis for Software Engineering. *IEEE transactions on software engineering* 43, 9 (2017), 817–847.

Gerard Meszaros and Jim Doble. 1997. A Pattern Language for Pattern Writing. In *Int. Conf. on Pattern Lang. of Program Design*, Vol. 131. 164.

Kazunori Mizushima and Yasuo Ikawa. 2011. A Structure of Co-Creation in an Open Source Software Ecosystem: A Case Study of the Eclipse Community. In *Technology Management in the Energy Smart World (PICMET)*. IEEE, 1–8.

Konstantinos Plakidas, Srdjan Stevanetic, Daniel Schall, Tudor B. Ionescu, and Uwe Zdun. 2016. How Do Software Ecosystems Evolve? A Quantitative Assessment of the r Ecosystem.. In *International Systems and Software Product Line Conference*. ACM, 89–98.

Nick Rozanski and Eóin Woods. 2012. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley.

Walt Scacchi and Thomas A. Alspaugh. 2012. Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems. *Journal of Systems and Software* 85, 7 (2012), 1479–1494.

Raymond Scupin. 1997. The KJ Method: A Technique for Analyzing Data Derived from Japanese Ethnology. *Human organization* (1997), 233–237.

Joey Van Angeren, Jaap Kabbedijk, Slinger Jansen, and Karl Michael Popp. 2011. A Survey of Associate Models Used within Large Software Ecosystems.. In *Int. Work. on Soft. Ecosystems*. CEUR-WS, 27–39.

Michael Weiss. 2018. The Business of Open Source. In *EuroPLoP2018*.

Michael Weiss and Nadia Noori. 2013. Architecture as Enabler of Open Source Project Contributions. In *EuroPLoP2013*.

Tim Wellhausen and Andreas Fießer. 2012. How to Write a Pattern?: A Rough Guide for First-Time Pattern Authors. In *EuroPLoP2012*. ACM, 5.

Joel West and Michael Mace. 2010. Browsing as the Killer App: Explaining the Rapid Success of Apple's iPhone. *Telecomm. Policy* 34, 5 (2010), 270–286.

Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, and Naoyasu Ubayashi. 2014. Magnet or Sticky? An Oss Project-by-Project Typology. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 344–347.