CrossMark

ORIGINAL ARTICLE

Continuous clarification and emergent requirements flows in open-commercial software ecosystems

Eric Knauss[1] · Aminah Yussuf[2] · Kelly Blincoe[2] · Daniela Damian[2] · Alessia Knauss[2]

Abstract Software engineering practice has shifted from the development of products in closed environments toward more open and collaborative efforts. Software development has become significantly interdependent with other systems (e.g. services, apps) and typically takes place within large ecosystems of networked communities of stakeholder organizations. Such software ecosystems promise increased innovation power and support for consumer-oriented software services at scale and are characterized by a certain openness of their information flows. While such openness supports project and reputation management, it also brings requirements engineering-related challenges within the ecosystem, such as managing dynamic, emergent contributions from the ecosystem stakeholders, as well as collecting their input while protecting their IP. In this paper, we report from a study of requirements communication and management practices within IBM®'s Collaborative Lifecycle Management® product development ecosystem. Our research used multiple methods for data collection, including interviews within several ecosystem actors, on-site participatory observation, and analysis of online project repositories. We chart and describe the flow of product requirements information through the ecosystem, how the open communication paradigm in software ecosystems provides opportunities for "just-in-time" RE—and which relies on emergent contributions from the ecosystem stakeholders—, as well as some of the challenges faced when traditional requirements engineering approaches are applied within such an ecosystem. More importantly, we discuss two tradeoffs brought about by the openness in software ecosystems: (1) allowing open, transparent communication while keeping intellectual property confidential within the ecosystem and (2) having the ability to act globally on a long-term strategy while empowering product teams to act locally to answer end users' context-specific needs in a timely manner. A sufficient level of openness facilitates contributions of emergent stakeholders. The ability to include important emergent contributors early in requirements elicitation appears to be a crucial asset in software ecosystems.

Keywords Requirements engineering · Software ecosystem · Mixed method

✉ Eric Knauss
  eric.knauss@cse.gu.se

  Aminah Yussuf
  aminah.yussuf@gmail.com

  Kelly Blincoe
  kblincoe@acm.org

  Daniela Damian
  danielad@cs.uvic.ca

  Alessia Knauss
  alessiaknauss@gmail.com

[1] Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Gothenburg, Sweden

[2] Department of Computer Science, University of Victoria, Victoria, BC, Canada

1 Introduction

Recent research has regarded the development of large-scale software projects as ecosystems of interacting and interconnected organizations. Collaboration within an ecosystem increases the competitive advantage of the organization and the ecosystem itself [19]. By allowing third-party organizations to join a software ecosystem, a

⚙ Springer

development organization increases its innovation power and market reach [22].

One strategy to initiate and maintain a software ecosystem that is becoming popular is the open-commercial approach [21] (a.k.a Extended Software Enterprises [27]). In this approach, organizations open up internal information about the product, development process, and project communication, while maintaining a commercial licensing and copyright model to protect some of the organization's intellectual property. This approach lies between the more extreme approaches on the degree of openness—such as the widely proprietary, closed information flows around a defined set of partners (e.g. SAP) and the completely open information flows found in open-source ecosystems (e.g. IBM's Eclipse). The open-commercial model tends to abolish barriers, facilitate the building of communities and product adoption, and provide advantage over the non-open competitors [27, 28]. The increased transparency supports learning from observation and reputation management [12].

The openness in these types of software engineering environments, however, has important implications for the collection and management of the information related to software requirements from the multiple ecosystem stakeholders. One would expect decentralized ways of facilitating the flow of requirements among stakeholders, similar to those found in open-source projects [28]. However, unlike open-source projects, the scale and complexity of stakeholder relationships might be different in open-commercial enterprises; contractual considerations and commercial business models lead to problems in managing diverse sources of requirements and information flows, while the ecosystem's openness results in lowered ability to protect product strategic information. The lack of research in RE processes for software ecosystems has been highlighted in the systematic literature reviews [2, 42], and only recently we find case studies on this highly relevant topic [23, 61, 33]. In this paper, we extend this body of knowledge with findings from a mixed-method empirical study of requirements management practices within IBM®'s open-commercial software ecosystem CLM® (Collaborative Lifecycle Management).

The contribution of this work is twofold: First, we provide a detailed description of RE processes and how requirements flow through the open-commercial software ecosystem in IBM's CLM development environment. Specifically, we describe practices as well as challenges in stakeholder selection, communication and prioritization of requirements, managing context, and mapping requirements to the ecosystem's actors. We also identify two main underlying tradeoffs in open-commercial ecosystems: (1) Maintaining the openness and transparency in the ecosystem needs to be balanced with keeping the confidentiality of paying customers' business needs and (2) responding to market demands requires the ability to globally define requirements on the strategic level, while allowing self-organized teams to locally and timely address context-specific customer needs "just-in-time." These findings build upon our research results reported in our previously published conference paper [33].

This paper extends our previous investigation with a deeper analysis of the patterns of communication around requirements by the various ecosystem stakeholders, and the impact of their contributions in the requirements process. We find that more traditional requirements engineering activities are complemented by a considerable inflow of requirements-related information from *emergent contributors*, i.e. stakeholders who were not officially responsible but became directly involved in the clarification of a specific requirement. Our findings indicate these emergent stakeholders contribute to requirements discussions across organizational borders. We bring evidence that facilitating early emergent contributions on requirements is crucial in software ecosystems. These insights, while developed from investigating the IBM's CLM ecosystem, have implications for other software ecosystems in which information about proprietary products, their features, and development processes is being maintained in channels with some degree of openness.

2 Research setting and questions

In this study, we investigate requirements engineering and related knowledge flows in the IBM CLM ecosystem to understand how requirements flow through the ecosystem and how the ecosystem actors evolve their software products and services.

2.1 The IBM CLM software ecosystem

IBM's Collaborative Lifecycle Management (CLM) tool suite provides a set of integrated software development tools along with collaboration features. It is characterized by open communication channels through which all stakeholders can participate in discussions of work in progress, and it is built on top of the *Jazz platform®*, a service-oriented platform based on open standards. Begel et al. [3] refer to such software development tool suites with social media aspects as *software ecosystems*, a term defined by Jansen et al. [26] as

> "*A software ecosystem is a* <u>*set of actors*</u> *functioning as a unit and* <u>*interacting with a shared market*</u> *for software and services, together with the relationships among them. These relationships are frequently underpinned by a common* <u>*technological platform*</u> *or*

market and operate through the exchange of information, resources and artefacts."

Within the CLM ecosystem, a number of product teams (*set of actors in a shared market*) provide products especially designed to interoperate on the Jazz platform (*technological platform*), including[1,2] Rational Requirements Composer (RRC), Rational Quality Manager (RQM), Rational Software Architect (RSA), Rational Team Concert (RTC), and also some smaller products, such as Reporting which integrates Rational Insight capabilities in the ecosystem. In addition, products for crosscutting and platform-related concerns exist, such as Jazz Foundation[3] or Application Lifecycle Management (C/ALM).[4] The size of each product team ranges from tens to hundreds of engineers. In this paper, we refer to the product teams (or more generally: organizations) that provide or consume software products or services within a software ecosystem as *actors*.

Therefore, *actors* are constituents of the software value chain. *Stakeholders* are individuals or organizations, internal or external to the ecosystem, with an interest in the software products.

The CLM ecosystem is coordinated by IBM Rational® (an IBM brand), and integration adapters are available to connect the CLM ecosystem with other major players and ecosystems in the field. A special partner program allows third parties to offer products with certified compatibility to the Rational CLM products. Consequently, CLM environments are highly customizable and a solution for a specific customer is defined by the mixture of Rational and third-party products that work together in order to provide the required services.

For exploring RE practices in software ecosystems, the CLM ecosystem is particularly interesting, because of its transparent developer–customer communication through open communication channels. These channels include open chat rooms, wikis in which CLM developers author technical documentation, and a publicly available issue tracker with rich discussion forums around features, bug reports, and change requests. Developers learn specific end-user needs and important information on how the services in the software ecosystem are used. Customers and end users can articulate doubts about certain changes.

One could argue that our observations relate to any software development effort of significant scale and complexity. For example, Curtis et al. [11] discuss that in large-scale projects the distance between developers and customers is a major threat to project success. Software ecosystems have been proposed as one way to address complexity of large-scale software projects. Ecosystems divide a large-scale, complex software system into separate and independent products and encourage high innovation rates by allowing these products to be replaced and extended by third-party contributions [19]. We interpret results from this study as indication that while this helps with many activities of continuous software production, the underlying complexity still implies serious challenges for requirements engineering.

While we aimed at covering many product teams in our qualitative interviews, we focused our analysis of how external stakeholders contribute to requirements discussions of ecosystem actors (which is based on mining online repositories) on the RTC product in relation to contributions from members of Jazz Foundation, C/ALM, and external actors. Specifically, this allows us to understand how various stakeholders (e.g. managers, developers, users) contribute to the requirements clarification across organizational boundaries within the ecosystem.

2.2 An illustrative example

The following (fictitious but realistic) **running example** illustrates the complex RE landscape in the CLM ecosystem [32]:

The CLM ecosystem offers value for lifecycle management in all kinds of engineering projects. Yet, there is a strong focus on software development. Consider Alice Inc., a new actor in this ecosystem that wants to leverage the existing services for requirements management, quality management, version control, task management, and team collaboration to build a solution for the automotive domain. Alice Inc. aims at integrating existing tools and services in that domain like specialized requirements and quality management solutions, or domain-specific artifacts like Simulink models. Alice Inc. could offer CLM-based services in a cloud-based setup, but also based on local installation at a customer's site. Both variants offer similar challenges for the RE landscape as discussed in this paper.

2.3 Research questions for RE in software ecosystems

In order to examine requirements engineering practices and challenges in CLM's open-commercial paradigm, we

---

[1] https://jazz.net/products/clm/-visited2015-7-25.

[2] Some of the products have been renamed since the time of our study. RRC is now Rational DOORS Next Generation and RSA is now Design Manager.

[3] Jazz Foundation is a platform for building collaborative applications. https://jazz.net/jazz/oslc/projectareas/_Q2fMII8EEd2Q-OW8dr3S5w last accessed 2015-05-18 (needs Jazz account).

[4] C/ALM addresses integration and interoperability of CLM ecosystem actors. https://jazz.net/jazz/admin#action=com.ibm.team.process.editProjectArea&itemId=_0rSUcMixEd6A25wBGCmItw,last accessed 2015-05-18 (needs Jazz account).

employed Jansen et al.'s [27] proposal to analyze software ecosystems on three different scope levels. Scope level 1 examines the ecosystem's relationships to other ecosystems, Scope level 2 analyzes the relationships within the ecosystem, and Scope level 3 examines the ecosystem from the perspective of an individual organization within the ecosystem. The upper part of Fig. 1 depicts the CLM ecosystem at these three levels. Even with these scope levels, a full and complete description of any software ecosystem is very difficult to achieve. Therefore, in Fig. 1, we depict only the entities and relationships that serve our research investigation (note that we refer to actors with the name of the product or service they provide). The links between actors are reconstructed from our interviews and indicate that the actor on the right uses services of the actor on the left. The figure also shows the workflows, practices, and challenges we identified in this challenge and which we describe in detail in Sect. 4. In addition, we describe the role of emergent contributions crosscutting to these scope levels (see middle-left of Fig. 1).

Scope level 1 offers an external view on the ecosystem. The top-left part of Fig. 1 depicts the CLM ecosystem actors (RTC, RRC, RSA, RQM, Reporting teams) as well as actors of adjacent software ecosystems around the git, Jira, and SAP platforms. This scope level is concerned with differentiating features such as the technological platform, the target market, the participants, and its connectedness to other software ecosystems.

On Scope Level 1, RE can impact the ecosystem's ability to attract new ecosystem actors (e.g. customers), integrate with other service providers, or approach new markets (e.g. by adding new strategic features). Further, requirements need to be communicated within and beyond the software ecosystem.

Example Alice Inc. may require certain new features to be added to the existing products to make CLM services attractive to the automotive domain. If the CLM ecosystem fails to prioritize these features on its roadmap (e.g. because of other priorities), its economic survival might be endangered. If Alice Inc. is perceived as a strategic partner,
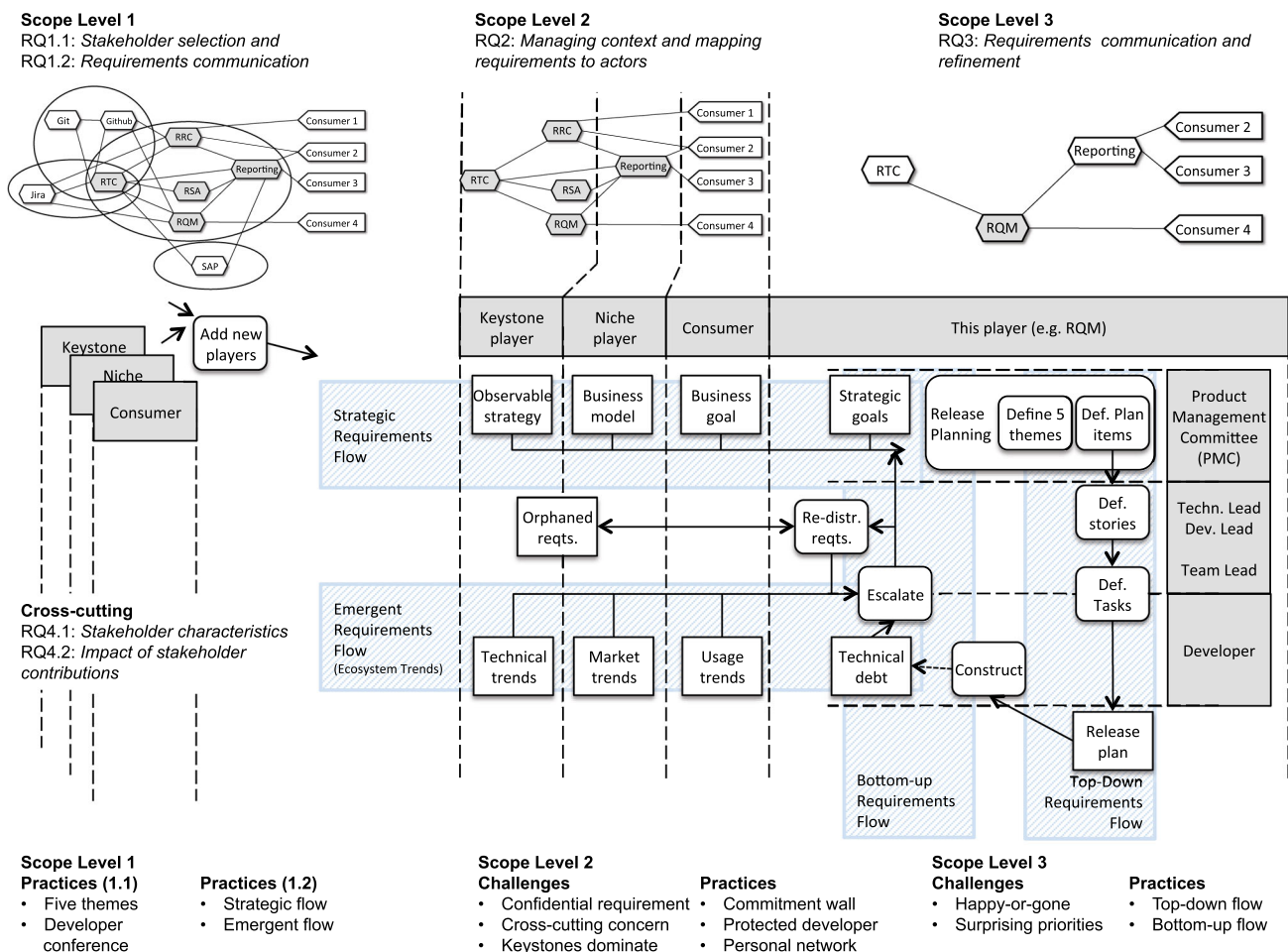
Fig. 1 Research questions, findings (flow of requirements, challenges, practices) in the context of software ecosystem scope levels

its requirements need to be communicated to all relevant actors within the ecosystem.

Stakeholder management discusses how different stakeholder perspectives can generally be synthesized into a roadmap based on collaborative decision making or game theory [18, 44], but it is unclear how this can be done on the scale of software ecosystems. Software ecosystems characterized by complex interrelationships among diverse and multiple stakeholders are susceptible to issues of stakeholder power [45] affecting activities of stakeholder selection or effective communication of their requirements [63]. Previous work proposed analyzing stakeholders, their relation, and their communication as requirements value chains [20] or based on social network analysis [15]. Yet, how stakeholders are selected and how requirements flow through software ecosystems in practice remains an open question:

RQ1.1: How are stakeholders selected across the software ecosystem?

RQ1.2: How are requirements communicated across the software ecosystem?

Scope level 2 shows the ecosystem from an internal perspective and allows the actor's roles and their relationships to be analyzed (top-middle in Fig. 1). Jansen et al. [27] discuss three important roles of ecosystem actors: *Dominators* control the capture of and creation of value in the ecosystem. Dominators are considered harmful to the ecosystem because they tend to take over functions of other actors who are then eliminated, reducing the diversity within the ecosystem [41]. *Keystone players* create highly connected hubs of actors by providing value to surrounding actors. They increase variability and robustness of the ecosystem and are considered beneficial to its health [41]. *Niche players* profit from the value that the keystone players provide and aim to differentiate themselves by developing special functions [41]. Based on our focus on RE, we add *consumers* as a fourth role, characterized by consuming services without offering their own services in the scope of the ecosystem. Table 1 gives an overview of the actors, their roles, and their typical stakeholders in the CLM ecosystem.

With respect to RE, it is important to understand how niche or keystone players position themselves in the ecosystem. According to related work, a significant amount of functionality in software ecosystems is based on the interplay of several subsystems [9, 54, 55]. Actors in the ecosystem offer services to consumers by re-using other services in the ecosystem. This value chain defines the context in which the consumer's requirements evolve and how difficult it will be to map new requirements to a specific service.

Example A consumer from the automotive domain would probably issue feature requests to Alice Inc., who is providing CLM ecosystem services to this new domain. Those feature requests can either be handled by Alice inc. directly or by one of the more fundamental service providers such as RTC or RQM product teams. It is not clear, however, how they are forwarded to the actor in the ecosystem that can handle them most effectively.

The open-commercial approach allows end users to articulate changed or new needs through a number of different channels [21]. This high level of openness, however, challenges end users and customers to send such requests to the correct recipient [9, 54]. To mitigate this challenge, suitable forums can be suggested to end users based on data mining techniques [9] or determined by exploiting sensors and monitoring capabilities in their mobile devices to automatically determine a suitable recipient in the ecosystem [54]. In a dynamic ecosystem, the ideal recipient for a given request changes over time, and it is necessary to periodically rethink the way requirements travel through the software organizations and which roles are responsible to seek alignment of goals [20]. Aligning goals depends on understanding requirements in their context, which calls for contextual techniques and observation. Observing how end users interact with their system allows requirements to be captured with their relevant context [39, 7]. End users could also articulate their needs themselves with the help of special feedback systems that facilitate adding screenshots and other context descriptions [38]. In addition, mobile devices allow users to present and discuss scenarios in the context of use [57]. Yet, it is not clear if and how practitioners can apply such approaches for managing the context of requirements and supporting software managers and developers in actively aligning development efforts and needs of relevant stakeholders at the scale of modern ecosystems.

RQ2: How are requirements mapped to ecosystem actors and how is the context of requirements managed?

Scope level 3 shows the ecosystem from the perspective of a single organization. For example, the top-right corner of Fig. 1 shows the perspective of the RQM actor in the CLM ecosystem. This allows investigating a specific actor's RE on a more tactical or operational level.

In a complex software ecosystem environment, it is unclear how actors can systematically understand the level of satisfaction of their external stakeholders (customers and end users) and share this information with internal stakeholders (software managers and developers).

Example Even for Alice Inc., it is difficult to understand the satisfaction of a potentially large and heterogeneous stakeholder landscape in the automotive domain. More

Table 1 Stakeholders in the scope of different CLM actors

Actor	Stakeholder	Role in RE activities
Consumer (*e.g. customer*)	Manager end user	Source of strategic reqts. and goals source of specific requirements from daily usage
Keystone (*e.g. RTC, RQM, RSA, RRC*)	End user	While using their own product, developers have specific requirements from daily usage
	Developer	Receiver of requirements for implementation
	Senior dev.	Source of requirements in relation to technical debt
	Techn. lead	Source of technical requirements, e.g. from architectural considerations
	Dev. lead	Assigning requirements for implementation and coordination of work
	Support	Facilitator of requirements flows from customers
	Product mgr.	Source of strategic requirements for the product
Niche (*e.g. Reporting*)	Same as keystone, but highly affected by technical decisions of keystones	

central actors like RTC or RQM might also suffer from the fact that important feedback is only given to other actors.

Agile and continuous development implies continuous clarification of requirements by software managers and developers throughout the development lifecycle [31]. Especially in distributed development, effective communication of requirements requires a high level of transparency [14, 13]. Strategies for such communication must address the challenge of achieving a good representation of a large and heterogeneous set of stakeholders. Traditional RE methods have been reported to be insufficient to support such wide audience requirements elicitation [60]. It remains an open question how a heterogeneous and potentially disagreeing audience that differs in their power to influence decisions can be integrated in the communication and refinement of requirements in open-commercial ecosystems.

RQ3: How are requirements communicated and refined in the scope of an actor in the software ecosystem?

Crosscutting, emergent communication is crucial in large-scale projects, e.g. to solve technical dependencies between products or components [56], to identify synergies [46], and to build communities of practice [58].

In an open-commercial ecosystem, stakeholders across the software ecosystem can participate in discussions that shape the way requirements are elicited, analyzed, and validated within the various ecosystem actors.

Example Developers and customers of Alice Inc. will be impacted by technical decisions and developments of the CLM products and thus could decide to participate as emergent contributors in online discussion of relevant issues.

This informal way of getting involved in requirements discussions shows similarities to open-source requirements engineering, where also informalisms dominate [52].

Recent studies found that requirement discussions are often cross-functional including developers, business analysts, and testers [16, 43]. However, these studies did not investigate how diverse the participants were in terms of their location within the software ecosystem.

RQ4.1: Who are the ecosystem stakeholders contributing to the requirements discussions?

RQ4.2: What is the impact of such contributions?

3 Research methodology

We examined the characteristics of the open-commercial development in the CLM ecosystem based on a mixed-method research methodology. Our data collection methods included participatory observation, semi-structured interviews, and analysis of software repositories.

3.1 Participatory observation

One of the researchers worked as an intern and developer in the Reporting team at IBM Ottawa for 2 months to obtain a broad view on the actors in the ecosystem as well as more in-depth knowledge with the development processes and practices within one specific actor of the ecosystem. During that time, he was directly involved in solving technical issues and participated in project meetings. In a daily journal, he logged information on how the development team analyzed requirements within this ecosystem actor and details of the team's interactions with a number of other internal and external actors in the ecosystem. This in-depth involvement with the development team within one actor in the ecosystem was invaluable in selecting interviewees and refining the research guidelines for the next phase in our data collection.

3.2 Semi-structured interviews

Based on the participatory observation, we conducted a series of 13 semi-structured interviews to explore the RE landscape in software ecosystems (RQs 1–3). The participatory observation allowed us to get in contact with developers and managers within a number of IBM internal actors of the CLM ecosystem that had a dependency with the Reporting team. We selected interviewees on different organizational levels, including six developers, one team leader, two development leaders, and four technical leaders. Experience of our interviewees in their role ranged from 6 month to 15 years, with an average of 2–3 years. During the interviews, we used a semi-structured interview guide[5] that covered our research questions, but allowed us to pursue related topics our interviewees brought up [6].

The interviews were conducted by the researcher who also did the participatory observation while the other researchers involved in this work iteratively analyzed the interview data based on the thematic analysis approach [6] as follows: We transcribed and read recordings from all interviews to get familiar with the data. One of the co-authors then coded the data from the perspective of research questions 1–3 and started searching for initial themes by grouping the initial codes. We then reviewed the initial themes in a series of workshops, regrouped and refined them by cross-checking the interview data with the generated codes and finally established a set of themes, consisting of both challenges and practices. For each theme, we defined a label and classified it based on scope level and process step. Based on this, we report on the identified challenges and practices from the perspective of a software ecosystem and in the context of the underlying process.

3.3 Analyzing repository data

One of the richer sources of information in this study was the ecosystem itself. Actors of the CLM ecosystem develop their products by using the CLM product suite itself, and developers can be considered as end users ("developer-end-users") in the scope of our study. As a consequence of its open-commercial development model, all of its requirements and tasks are documented as workitems in the integrated issue tracker and are available online. By querying and analyzing this repository, we were able to triangulate findings from participatory observation and semi-structured interviews. For example, we were able to query the issue tracker for all user stories with attachments

from external partners and confirm the statement from one of the interviews that while many companies consider such data to be confidential, others share them openly. We analyzed workitems and their meta-data (name and affiliation of the owner of workitems, comments, and attachments), as well as the discussion of all issues related to the workitem's clarification, coordination, and implementation. This was supported by the fact that each CLM project has a publicly available list of project members, which shows the current status of project membership.

We also relied on this data source for a closer analysis of stakeholders and their contributions (RQ 4), for which we focused on user stories from the RTC product. We chose RTC for our analysis because it has a rich issue-tracking repository in which communication about its requirements is documented online (in the form of comments) and is traceable to requirements. Requirements are defined in the form of user stories, and ongoing discussions around these user stories serve as the main mechanism to clarify the meaning of requirements and to coordinate their implementation [8].

In particular, we analyzed a snapshot of the issue-tracking system between December 2006 and June 2008. The snapshot contains 157 stories. Out of the 157 stories, 60 did not have any comments. We thus analyzed 97 stories and their requirements discussions with a total of 431 comments.

3.4 Manual content analysis and coding

To further analyze the types of contributions made by ecosystem stakeholders (RQ 4.2), we analyzed the contents of the contributions using grounded theory methods [10]. We identified categories based on the contributions and coded each contribution to a category. Two independent coders completed this coding. When each coder was satisfied with their codes, their code lists were combined to create a master code list. Each coder performed another iteration to apply a code from the master code list to each response. We measured inter-coder reliability with Krippendorff's alpha measure [34] and found the initial agreement between the two coders to be reliable (Krippendorff alpha score of 0.81). After the coding was completed separately by both coders, the two coders compared their findings and discussed any differences. This round of reconciliation resulted in a set of codes with 100 % agreement between the two coders.

4 Findings

Our findings (see summary in Fig. 1) describe how requirements flow in software ecosystems, themes of RE in the CLM ecosystem, and the role of emergent

---

[5] Examples of guiding questions: How are requirements elicited and communicated to you? How do you prioritize requirements? How do you deal with context in requirements engineering for software ecosystems?.

contributions. Our themes include practices for all three ecosystem scope levels and challenges for scope levels 2 and 3. We do not address challenges at scope level 1 since our interviewees, software managers and developers in various ecosystem actors, were not aware of specific challenges at this high level. Crosscutting through all three scope levels, we characterize the stakeholders that contribute to requirements discussions as well as the impact of their contributions.

RQ1.1: How are stakeholders selected across the ecosystem?

Software product release planning has been extensively discussed (e.g. [49]). In our interviews, we focused on the specifics of creating a roadmap for an actor within the software ecosystem. We found two practices that support the selection of stakeholders: The first practice, which our interviewees refer to as *five themes*, supports roadmapping and selecting features for the next release, while the second practice *developer conference* facilitates decisions about the technological platform. Since software ecosystems are inherently open to some degree, stakeholder selection is done mostly indirectly by announcing a roadmap that is attractive to certain actors, or by adjusting the technological platform to make it easier to join the ecosystem. The practices *five themes* and *developer conference*, which we discuss in more detail here, influence the willingness and ability of players to participate in the ecosystem.

Five themes The CLM actors that belong to IBM are usually managed by a Product Management Committee (PMC), consisting of technical leads, development leads, and product managers as well as representatives from support and sales. This PMC defines *five themes* to set the goal for the next release based on strategic considerations which are then used to define the roadmap for the next release. By including sales and support, knowledge about strategic needs of potential new actors is included in this discussion. These *five themes* can be considered as the primary steering instrument, which ultimately open the ecosystem for new actors by including or excluding crucial requirements:

"Having those high-level themes helps us to frame, looking at individual requirements and saying, yeah, actually that goes along with this sort of cluster of requirements."

Open communication of the *five themes* for the next release cycle allows the alignment of several actors and shapes the attractiveness of the ecosystem to new actors.

Developer conferences These conferences aim to bring together technical leaders of all actors in the ecosystem and

provide a forum for discussing future directions of the underlying platform and concepts of the ecosystem, including actor interdependencies. Thus, they play a major role in framing the software ecosystem on Scope Level 1.

"[At] innovate conference [we] get this senior technical team together regularly. We take advantage of that by spending three solid days planning for the next release."

Technical leaders participating in these yearly events come from all ecosystem actors (keystone players and consumers) and meet for 3 days. Many of them are part of the PMCs (or equivalent in non-IBM actors) and can shape the *five themes* and the roadmap based on input from these conferences.

Answer to RQ1.1: Stakeholders are indirectly selected across the ecosystem by publicly announcing a *roadmap* and high level themes, called *five themes*, that relate to the next software release's strategic goals. Stakeholders have the opportunity to discuss the technical scope of the ecosystem at *developer conferences*.

RQ1.2: How are requirements communicated across the software ecosystem?

Strategic requirements flow This flow is characterized by sales and support activities through which business goals from consumers as well as strategic information from other actors in the ecosystem are introduced into the ecosystem and need to be considered during release planning. The strategic flow resembles traditional RE (e.g. [47, 49]) by focusing on systematic analysis of business goals from stakeholders and their refinement to detailed requirements, but the ecosystem adds an increased need to take into account strategies of other keystone players and to anticipate their movement in the ecosystem. Business models of niche players can be considered, e.g. to understand how those can contribute to the own strategy. For example, the niche player Reporting offers additional business value for many CLM products.

Emergent requirements flow The complexity and ever changing nature of the ecosystem result in many new requirements that are based on *ecosystem trends* and introduced into the ecosystem as *emergent requirements flow*. Such emergent requirements originate from end users from consumers in the ecosystem, who find new ways of using existing services, and developers from other keystone or niche players, who come up with innovative solutions. Consequently, ecosystem trends become visible in discussions of low-level workitems in open communication

channels and need to be fitted into the different actors' plans, e.g. by "squeezing them into the scope of one plan item" as one team lead put it. According to our interviews, roughly $\frac{2}{3}$ of the end-user requirements originate from end users at the customer sites, while $\frac{1}{3}$ originate from CLM developer–end users. In addition to this, technical debt and bugs emerge as the ecosystem evolves and are communicated partly internally and partly on open communication channels. The emergent requirements flow resembles the requirements practice in open-source projects [52] where requirements emerge as informalisms, through continually emerging Webs of software discourse (e.g. email and discussion forums).

We will discuss the role of such emergent contributions in the scope of Research Question 4.

> **Answer to RQ1.2:** We found that requirements are largely introduced to the ecosystem via two general information flows. The *strategic requirements flow* takes into account business goals and global strategies, while the *emergent requirements flow* results from local just-in-time RE activities and the open communication paradigm (illustrated in Fig. 1).

RQ2: How are requirements mapped to actors in the software ecosystem and how is the context of requirements managed?

Our findings for RQ2 are illustrated in the center of Fig. 1. Managing the context of requirements and mapping requirements to actors are highly interconnected and challenging RE tasks associated with Scope Level 2. We identified three challenges that express how constraints and complexity of open-commercial software ecosystems affect the engineering of requirements at this level:

Confidential requirement In a commercial setting, customer specifics and requirements often contain confidential information about the customer and cannot be discussed on open-commercial channels. Development or team leads are responsible for this confidential information and forward it to developers as needed.

> "We can work on sanitizing the requirement [and] always translate these into public workitems on jazz.net [so that] either it does not have any customer identifiable information in it or [...] the customer is okay with this."

Consequently, important information (especially related to the context) is not shared through open communication channels, threatening the transparency of the software ecosystem.

> "Informally the context is captured with one requirement, we understand this requirement is coming from organizations with this kind of environment and this kind of expectations [...] shouldn?t we be capturing that context in a way that is more structured?"

Crosscutting concern Practitioners frequently struggle to understand whether requests of different customers could be addressed generally (closer to the platform) or only specifically (by a peripheral actor). This includes dealing with crosscutting concerns that affect several actors or requirements that should entirely be assigned to other actors in the software ecosystem. Particularly, when several actors need to collaborate to work on such a request, a systematic approach is desirable, yet missing.

Keystone dominates Niche players couple their value creation in a narrow niche closely to integrating services provided by keystone players. Thus, they depend on these other actor's technical decisions which can have more impact on the niche player's requirements than the needs of customers or end users.

> "Most of the stuff, we just have to do it to keep up with the ecosystem and platform."

This challenge is caused by the nature of software ecosystems and significantly impacts the way requirements are mapped to niche players. Yet, niche players have little power to deal with this challenge themselves. If it is encountered too frequently, this challenge can have a serious effect on the ecosystem's overall health [41]. Keystone players should be considerate in their action: while each major design decision can offer new opportunities to some of the actors in the ecosystem, it is potentially harmful to others. Keystone players need to provide some stability to niche players, because their limited resources might not allow them to follow new technical trends [41]. Yet we found no evidence for a systematic approach to monitor the requirements and needs of niche players to support decision making on this level.

Practitioners address these three challenges based on three practices:

Commitment wall As a consequence of the *confidential requirement* challenge, the release planning is partly intransparent, because confidential information like customer priorities cannot be disclosed. To mitigate potential conflicts (e.g. when a paying customer is over-bidden by another), developers are not allowed to commit to a feature or bug-fix for a specific milestone or time-frame.

> "We try to ensure that there is a bit of a wall in place between development and customers when it comes to commitment for enhancements [...] or for new features."

Protected developer   Open-commercial channels facilitate direct feedback from end users to developers, but developers are widely relying on their senior team members to give them all required information. The less seniority our interviewees had, the less awareness for the special challenges of ecosystems they showed. Senior team members intercept and resolve these ecosystem challenges and allow developers to focus on development tasks.

> *"There are customer calls [taken by team or technical lead], then priorities are passed down in weekly meetings."*

In this way, team leads and other managers effectively shield their developers from any challenges arising from context-specific requirements.

Personal network   In the ecosystem, it becomes difficult to map emergent requirements to specific actors, especially since these requirements are often crosscutting over several players in the ecosystem. It is important to understand the context of such requirements and to systematically forward this information to other relevant players in the ecosystem. Our interviews indicate that this task depends on individual excellence and ad hoc coordination between seniors of several ecosystem actors.

A common practice of internal stakeholders is to make use of their *personal network*. They meet senior staff of other actors in the ecosystem and try to follow relevant developments in the ecosystem, which allows them to discover and resolve crosscutting concerns, map (and forward) requirements to different subsystems, and to understand their general context.

> *"A lot of it is basically talking to the senior people on the [different] team[s]."*

Some internal stakeholders even actively track open communication channels of other actors to identify crosscutting problems without this task being formally assigned to them.

> *"You need to know all workitems of the last years and their history."*

The ability of these senior team members to function in this job depends, in large part, on their personal experience and network. We found no systematic approach for capturing and managing context. Our interviewees expressed their concern that without such an approach, the success of an actor in the ecosystem depends too much on individuals, endangering long-term health and reliability of large-scale software development.

> **Answer to RQ2:** Engineers in the CLM ecosystem rely on their *personal networks* to coordinate requirements between ecosystem actors, while team leads *protect developers* from the complexities with respect to context. *Cross-cutting concerns*, *domination by keystone players*, and the management of *confidential requirements* are particularly challenging. A *commitment wall* protects ecosystem actors from accidental commitments towards other actors.

RQ3:   How are requirements communicated and refined in the scope of an actor in the software ecosystem?

On Scope Level 3 (right-hand side in Fig. 1), developing teams struggle with the complexity of the software ecosystem and with the consequences of practices at Scope Levels 1 and 2. For example, *protecting developers* from the challenges of developing software for an ecosystem (practice on Scope Level 2) can cause those developers to be surprised by developments in the ecosystem (see *surprising priorities* below) or cause testers to struggle to set up a representative test frame or reproduce a bug. Another challenge on this scope level, *happy-or-gone*, refers to a lack of immediate contact with end users and customers.

Surprising priorities   We found especially junior developers to heavily rely on their managers and senior developers in the team for navigating the software ecosystem challenges (see also the *protected developer* practice). Consequently, it is hard for them to anticipate changes of requirements or priorities.

> *"You'd ask [name of technical lead] or one of the persons in charge, then you get pointed to the guys who know about an issue. [...I would like] a head start about something becoming a priority, before it becomes that urgent thing that needs to be fixed right away."*

Priorities become even more surprising due to the large number of information channels, which are only partly open, and due to the lack of systematic feedback about stakeholder satisfaction.

Happy-or-gone   Strong sales and support processes are in place to understand stakeholder needs, and user experience teams can provide further input (see *strategic requirements flow* practice), but there is a lack of transparency and systematic approaches for channeling positive stakeholder feedback. If there is a problem, development teams usually receive consumers' complaints, but if there is no complaint, they do not know if the consumers are happy with the service or if they have stopped using it.

"[Sometimes I wonder if it is] really going well for everybody or is nobody using it. And those two situations are sometimes hard to distinguish."

In the presence of these challenges, we found practitioners to rely on two general flows of requirements:

Top-down requirements flow Members of the PMC, team leads, and developers work with specific artifacts in the CLM issue tracker, as indicated by the *top-down requirements flow* on the right-hand side of Fig. 1. The PMC works with the so-called *plan items* in the issue tracker to create a release plan. Such *plan items* refer to major development efforts and comprise several *[user] stories* (another specific workitem type). User *stories* are derived from *plan items* and assigned to team leads. *Tasks* are defined based on these *stories* and assigned to developers. It is usually during this final refinement that most of the crosscutting concerns and context specifics are resolved, so that developers can focus on the task with minimal distraction.

"[...] PMC lead would assign me some of the plan items which I will own and which I can refine then and assign to [team lead name] or whoever else is on my team. Then refine those into stories, enhancements, and tasks that we can give to the developers to actually work on."

The top-down requirements flow addresses traditional requirements refinement in an effective way, but on its own is not sufficient to address the software ecosystem challenges.

Bottom-up requirements flow In order to deal with challenges discussed at both Scope levels 2 and 3, team leads and senior developers use their personal contacts to facilitate the flow of requirements between their team and support or other development teams. Open-commercial instruments allow customers and end users (including developer–end users) to introduce requirements themselves (by filing bugs or change requests) or to discuss their needs in the comment stream of an existing workitem.

"[Customers are] either submitting workitems directly, like defects, or there is some forum / blogs, where people ask questions."

Such information is often available on a very specific and low abstraction level, and senior developers who have a good overview of such issues are consulting the PMC directly or indirectly about trends as well as technical debt.

"When we come up with these high level themes, we often come, we always come up with this grab bag called technical debt."

In the absence of a systematic approach or formal process, we found the *personal network* and experience of team leads and senior developers to be the only way to understand the level of stakeholder satisfaction and to highlight technical debt that should be addressed.

"[You need to] Get a feeling about a large number of reported issues. It really depends on your experience". —"One thing I did was internally, just informally, poll the people who I know to be the technical leaders on the team."

We describe this ad hoc treatment of emerging requirements and technical debt as *bottom-up requirements flow* (on the right in Fig. 1).

With respect to knowledge management, this situation is dangerous, because experienced and well-connected team leads and senior developers are hard to replace. In addition, the complex environment results in a steep learning curve for new developers and managers, before they can assess stakeholder satisfaction or manage requirements flows. Managers among our interviewees agree that finding a more systematic solution for managing complexity of software ecosystems is one of the future challenges.

> **Answer to RQ3:** Communication of requirements depends on *top-down* and *bottom-up requirements flows*. Through the complexity and volatility of the ecosystem, *surprising priorities* are encountered as challenges, as well as disconnectedness or silence from customers, who are either *happy-or-gone*.

RQ4.1: Who are the ecosystem stakeholders contributing to the requirements discussions?

Through our analysis of the requirements discussions in the RTC issue tracker, we discovered developers openly discuss features they are currently developing. This behavior stems from the agile development process and actively promotes the ecosystem and its actors [21]. In addition, our analysis revealed that many contributions to these discussions were provided from stakeholders not associated with the RTC product team. This confirms our finding of the *emergent requirements flow* practice: We indeed find such emergent contributions from other actors of the CLM ecosystem. We refer to such contributors as *emergent stakeholder* and define them as follows:

Definition An *emergent stakeholder* (or emergent contributor) is a stakeholder who (1) is not officially required or responsible for the requirement, yet (2) contributes to the discussion of the requirement in some way.

We analyzed the RTC repository data to identify emergent stakeholders. We considered emergent stakeholders as those https://jazz.net users who (1) were not a

member of the RTC project but (2) contributed to an online discussion around an RTC requirement in the issue tracker. Emergent stakeholders identified through this analysis included developers from other IBM products and even https://jazz.net users external to the IBM organization.

Figure 2 shows a visualization of contributions made on RTC requirements discussions. The figure highlights the large number of contributions from emergent stakeholders. For visibility, the visualization is limited to emergent stakeholders from two other CLM products, C/ALM and Jazz Foundation, and emergent stakeholders external to the IBM organization. Table 2 provides more details on the number of emergent contributions. Of the 431 comments we analyzed, 231 or 54 % originate from emergent stakeholders, indicating that such stakeholders play an important role in the requirement clarification process.

> **Answer to RQ4.1:** More than half of the comments on user stories come from stakeholders who are not affiliated with an ecosystem actor.

*RQ4.2:*   What is the impact of such contributions?

Based on our analysis of RTC workitem discussions, we characterize the impact of emergent stakeholders'

contributions. Manual content analysis of the discussions revealed four main categories of contribution: requirement negotiation, coordination, information, and implementation status (Table 2). The remaining categories identified during this analysis each accounted for only a very small number of comments and are summarized as *other comments* in Table 2.

Emergent contributions account for 79 % of requirement negotiation comments indicating that emergent stakeholders play a large role in developing new requirements. On the other hand, non-emergent stakeholders account for 80 % of implementation status comments. This is not surprising since the non-emergent stakeholders are those responsible for the workitems.

Since emergent stakeholders bring a different perspective to the requirement process, we believe it is important for them to contribute early. We, thus, analyzed the timing of emergent contributions to identify whether emergent stakeholders are contributing early in the timeline of a requirement life cycle.

A Mann–Whitney test of difference in distribution shows that emergent stakeholder contributions occur later than non-emergent contributions when considering the number of days a contribution is made after the creation of the workitem (Table 3). The difference is even more
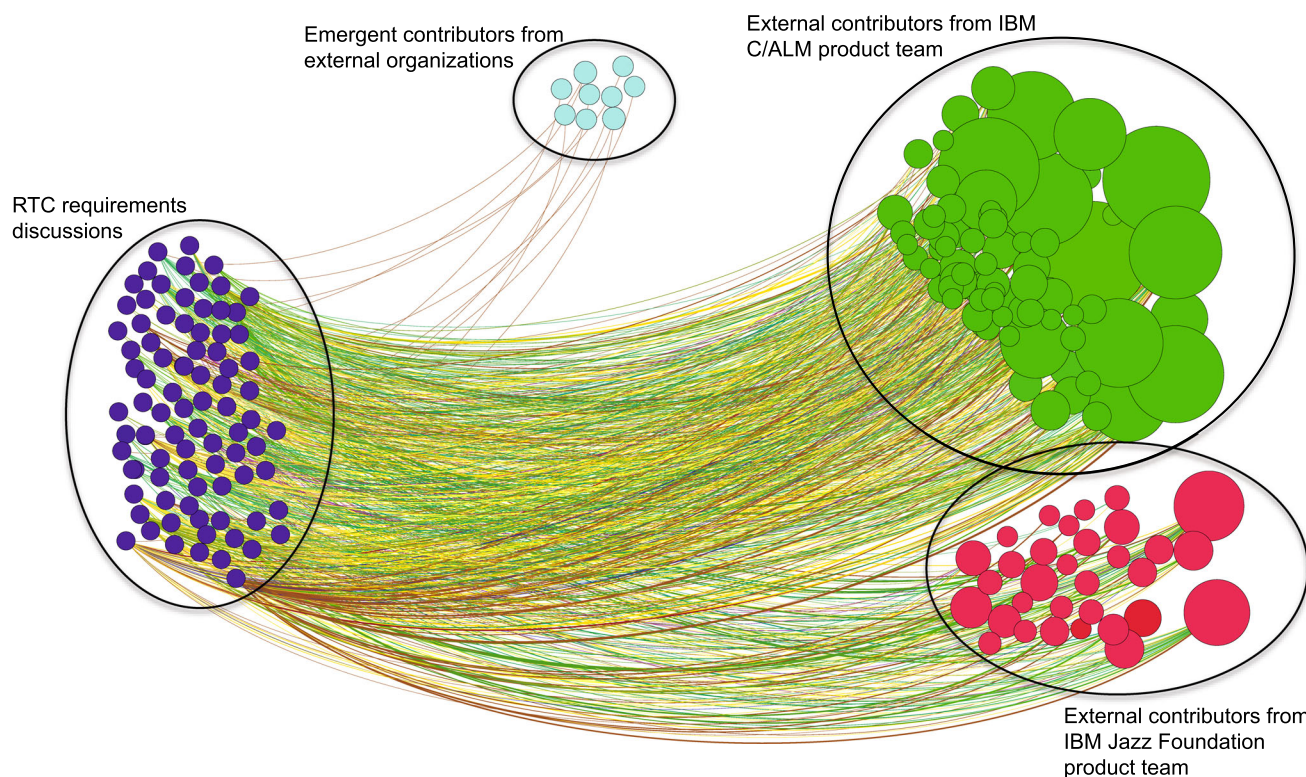
Fig. 2 Visualization of emergent communication in the CLM ecosystem (here: from the perspective of RTC). Discussions of requirements in the RTC issue tracker receive contributions from emergent contributors, either from other product teams within IBM or from other organizations. Size of nodes depicts number of emergent contributions by this stakeholder, *color* of links depicts comment type (*yellow* requirements negotiation, *green* coordination, *brown* information) (color figure online)

Table 2 Characteristics of comments from emergent/non-emergent stakeholders

Comment	Number of comments	Emergent	Non-emergent
Requirement negotiation	112	88 (**79** %)	24 (21 %)
Coordination	111	48 (43 %)	63 (57 %)
Information	65	31 (48 %)	34 (52 %)
Impl. status	51	10 (20 %)	41 (**80** %)
Sum (rel. classes)	339	177 (52 %)	162 (48 %)
Other comments	92	54 (59 %)	38 (41 %)
Sum (total)	431	231 (54 %)	200 (46 %)

Table 3 Timing of contributions (median number of days after creation of workitem)

	All contributions	First contribution
Emergent	29 days	28 days
Non-emergent	21 days	14 days
Mann–Whitney test	W = 24, 864.5*	W = 8316**

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

significant when considering the timing of the first contribution of each stakeholder on a workitem. Emergent stakeholders begin contributing later than non-emergent stakeholders.

Since emergent stakeholders play a large role in requirement negotiation yet contribute later than other stakeholders, we analyzed the comments on the workitems to see how these emergent contributions impact the requirement process.

Early contributions by emergent stakeholders drive the requirement: We observed some cases where the emergent stakeholders commented early in the workitem timeline describing their need for the particular feature implementation or defect fix. The emergent stakeholders, therefore, acted as catalysts for these changes. From the discussions, it appears that the early contributions from the emergent stakeholders speed up the requirements clarification and decision-making process. Those responsible for implementing the task (non-emergent stakeholders) respond quickly to the needs of the emergent stakeholders.

The discussions around Workitems 1 and 2 shown in Tables 4 and 5, respectively, illustrate this pattern. On Workitem 1, we see two emergent stakeholders clarifying the requirements. The developer assigned to the workitem agrees on the requirements and implements the functionality. On Workitem 2, we see two emergent stakeholders commenting to voice their need for the new feature that is later implemented.

Late contributions by emergent stakeholders can cause disruption and rework in the requirement process. However, we also observed cases where emergent stakeholder contributions start or accumulate much later in the workitem time line. In many cases, the late involvement of an external stakeholder caused problems. For example, Workitem 3, in Table 6, shows a case where two emergent stakeholders provide some input early, but then are not involved for a long time. Only late in the lifetime (after day 120), more emergent contributors start to clarify additional requirements, after much of the work has already been completed. Such late clarification can cause rework and problematic changes in scope. Based on these observations, we argue that without emergent stakeholders driving the need for a user story early, decision making is slower and requirements are not always well understood. This can lead to a unsuccessful outcome.

> **Answer to RQ4.2:** Emergent contributors (from other ecosystem actors) provide mainly clarification of requirements. It appears that early engagement of emergent contributions can drive the clarification of the requirement, while late emergent contributions can lead to disruption and rework.

5 Discussion: continuous and emergent requirements clarification

In this paper, we studied Requirements Engineering practices and challenges in the CLM ecosystem. We employed a mixed-method methodology that included participatory observation, semi-structured interviews and qualitative coding, analysis of repository data as well as manual content analysis. Our findings suggest that communication about requirements in the ecosystem involves many groups and layers. We found requirements flow both bottom up and top down. In particular, we found that a traditional inflow of strategic requirements is complemented by an emergent requirements flow. To more closely analyze the latter phenomenon, we investigated characteristics of ecosystem stakeholders and their contributions in the emergent requirements flow. We found that more than half of all contributions originate from emergent stakeholders and that those contributions can be critical success factors when they can be elicited early in a new features lifecycle.

The observations and findings in our study are of course specific to the CLM ecosystem. However, they reveal underlying implications for research and practice (including tools) as well as tradeoffs that openness brings to how requirements are managed within software ecosystems. In this section, we discuss these implications and tradeoffs, as

Table 4 Sample Workitem 1. Emergent stakeholders' needs driving requirement process

User	Type	Day	Comment
1	Non-emergent	0	This story talks about how we will enable [feature x] for team concert 1.0
2	**Emergent**	43	Don't we need the ability to create [feature y], with [a and b] functionalities as well?
3	**Emergent**	43	I agree and we really need to define the scope here with a scenario. I've added a [P] item to the [M] plan
2	**Emergent**	102	Without [b] functionality, you can't complete the [a] task, I think we need to include that. If I think about how our IT folks work, I would really like to see the Web UI cover the [function a] scenario
4	Non-emergent	108	We will introduce the notion of [z] feature. The web ui should allow to carry out [function a and b]
5	Non-emergent	132	Primary functionality added in M5...will continue to polish in M6

Table 5 Sample Workitem 2. Emergent stakeholders' needs driving requirement process

User	Type	Day	Comment
1	Non-emergent	0	I'm currently working on [function a] for Jazz
2	**Emergent**	122	I'd be very interested in this functionality. I am investigating using RTC for my Team (and my Area). Is this available in RTC 1.0?
3	**Emergent**	148	Do you need testers for this? We have an immediate need [for function a]
4	**Emergent**	189	So is this still happening or was it taken off of the board?
3	**Emergent**	190	The [team X] folks have started an open source project inside IBM. It's currently in early design phase
5	**Emergent**	204	If this feature is ready, will RQM benefit from this too? Then people using RQM can [utilize function a]. Really looking forward to it
6	Non-emergent	214	Dear all, iteration 1 for [function a] has been released
6	Non-emergent	278	Dear all, iteration 2 for [function a] has been released

well as the ways in which IBM teams approached associated challenges.

5.1 Implications for RE research, practice, and tools in software ecosystems

Our findings highlight the need for development of theories, techniques, and tools to help developers, and their organizations effectively manage the growing amount of information in their product development ecosystem. For organizations, staying abreast of market trends while maintaining strategic partnership with other organizations in their product development ecosystem creates real competitive advantage in today's interconnected software industry.

Toward this goal, our research contributes a method for identification of conceptual dependencies between project stakeholders in the ecosystem; those who contribute to discussions have some relationship to the requirements being discussed. These relationships could be known or emerging in the products' architectures. Either way, knowledge about such relationships helps toward understanding the stakeholder structure and dependencies of particular features and thus should be leveraged for more effective support for coordination of requirements development.

Our research suggests that project managers and technical leaders, in particular, would immensely benefit from development of new methods and supporting tools (e.g. recommender systems) for release planning of current iterations. Such methods and tools should help to involve relevant ecosystem stakeholders early in the requirements discussion and validation. Involving the right ecosystem stakeholders early ensures appropriate product direction, improving overall productivity and quality. Supporting tools could automatically analyze existing information from online stakeholder collaboration and inform effective planning of future product releases. Those stakeholders that have been found to have a positive impact in earlier iterations on requirements that are similar or dependent on those in the current release should be included in the current release.

5.2 Tradeoffs in open-commercial software ecosystems

5.2.1 Tradeoff 1: act openly versus act proprietarily

The open-commercial approach in the CLM ecosystem means that customers and end users, in fact every stakeholder interested in the development, have access to the issue tracker and can see the current progress of the project.

Table 6 Sample Workitem 3. Late emergent contribution causing rework

User	Type	Day	Comment
1	Non-emergent	0	We need a common approach and guidelines for how to improve error messages. The result of this effort is a wiki page with agreed on guidelines for the component teams
1	Non-emergent	10	I've added some screenshots with examples of non-supportive error messages
2	Non-emergent	19	A first draft of the wiki page is available
3	**Emergent**	20	Looks good:) couple points I did not see in the wiki [list of recommendations]
2	Non-emergent	23	The draft has been updated
4	**Emergent**	31	We currently don't have a story for expressing process problems to REST clients, such as our web UIs. Do we address this gap here or somewhere else? Also, was it our intent for this work item to also improve how web UIs display error messages or is this work item focused on only the Eclipse-based UIs?
2	Non-emergent	31	So far we only looked at the eclipse UI and I agree that we need to extend the scope to include the WebUi
5	**Emergent**	35	I've read the proposal and tried to implement it, I have some feedback based on this experience. Let's start with an example. [long list of issues omitted, including:] This message has to be internationalized based on the clients locale
	Emergent 3	35	> *Re : This message has to be internationalized based on the clients locale.* Ouch right, the server may be in China and the client in Germany...[...]
2	Non-emergent	39	In respect to comment 12: having the client library in charge as in the suggestions for issues (1) and (2) makes sense if there are appropriate client library calls[...]
3	Non-emergent	39	[C.] Made the following recommendations/observations: *[...] the recommendation to "hide functionality" when the connection is lost might not work well in situations where connectivity is frequently interrupted for a short time (e.g. when you're on a train that goes through many tunnels [...] This supports the user's intent to "create a work item" without having to figure out what the preconditions are*
	Emergent	39	I'm wondering what is implied by "Always handle StaleDataException for all of your save methods." What exactly does "handle" mean? Should auto-merging always be attempted, or can the user be prompted whether to overwrite or not? [...] Or is simply displaying a nicer error than the raw StaleDataException an acceptable way of handling it?
5	**Emergent**	39	The low bar is what you proposed as the last option, provide a nicer explanation to the user about what happened [...]
7	**Emergent**	122	[...] We need a rule that code must never throw a new exception, checked or unchecked, without a message. [...] Several times we've been presented with exceptions where there was no associated message which made it difficult for consumers to solve their own problem, and difficult for developers to match up the error with the problem. The line numbers often don't line up between the stack trace and the current version of the code, so any message is helpful to the developer
3	**Emergent**	126	Ahhh, the joy of 'exception with no message':) [...] do we need more than the stack trace? maybe the build? Some screen capture? [...should we] provide internal info as first failure data capture to the development team [...]?
7	**Emergent**	126	[...] Seeing an actual message, whether it's in a log file, a bug report, a JUnit test result in a build, or an instant message window, makes diagnosing the problem much easier[...]
3	**Emergent**	126	[...] How much value would we have getting an 'anArg is null' and a stack trace versus just the stack trace that points us to the line in the code where I can see I was doing an assert?[...]
7	**Emergent**	126	The stack trace with a line number works great when the line numbers match up. Oftentimes however the developer has a newer version of the file loaded in his workspace, and so a line number only will not always get you to the right place[...]
3	**Emergent**	126	[...] I see 2 things that could be done in parallel (1) adding a message that is unique so I can find the exact line in the code (and can provide value) (2) [...] when one opens a work item and attaches a stack trace, RTC can [...] parse it, find the exact build and open the 'source code' at the exact line (in the old code stream) and shows the new one...crazy idea?:)
8	**Emergent**	126	You're not the only one with crazy ideas;-). See item NNN

End users can comment and submit bugs, and they can see the status of their requests. Yet, business needs make it necessary to treat certain information confidentially. The following *forces* need to be balanced with respect to this tradeoff.

Information flow. The open-commercial approach with its open communication channels facilitates information flows between end users, customers, developers, and software managers. This high level of transparency is, one could argue, an unprecedented opportunity in how software

projects are engineered and an asset in dealing with the complexity of such ecosystems.

Confidentiality. If, for example, a new report has to be created for a given customer, this will appear as a workitem in the issue tracker on https://jazz.net. For the development, it is important to know the context of this report: How is it embedded in the customer's processes and what exactly are the customers information needs. An example of an old report this customer is using would be helpful. However, most customers are reluctant to share such intimate information about their central business processes in an openly accessible issue tracker. For this reason, it is not possible to have all information at one place, and the openness is broken.

Priorities. A special case of confidential information is priorities and their rationales, which cannot be openly shared in many commercial settings, leading to incomplete information and intransparent decisions on open channels.

General solution strategy: Introduce layers between customers and developers. In our case study, actors acquire sensitive, confidential context-specific information through sales and support groups and share them directly with the developers in charge. They ensure that discussions in open communication channels are on a high level of abstraction, e.g. by introducing acronyms such as LUGA (Large Unknown Government Agencies) to refer to anonymous entities. We found, however, that this strategy hinders the information flow, significantly adds to management effort, and increases the challenge of creating a holistic product strategy that is in line with context-specific requirements of specific customers, which we will discuss next.

5.2.2 Tradeoff 2: act globally versus act locally

The CLM ecosystem is in direct or indirect competition with other lifecycle management solutions, e.g. *Visual Studio Application Lifecycle Management* [48] or *SAP Solution Manager* [53]. In order to position CLM against these competitors, strategic decisions need to be made that affect the whole ecosystem. To define a global strategy in a software ecosystem of this complexity, a strict plan-driven top down approach is suitable. At the other end of the spectrum, very local decisions need to be made to adjust services to meet the ever changing consumer needs or to address technical debt. In a constantly evolving software ecosystem, these fast and agile decisions ask for local empowered development teams. The following *forces* need to be balanced with respect to this tradeoff:

Understanding customers in context. User stories and tasks can be difficult to understand when the context is not clear [57]. Context in the CLM ecosystem depends on various factors, and descriptions of workitems frequently show gaps due to confidential information. Consider, for example. a given customer with a unique setup, consisting of server and client platforms as well as a specific mixture of CLM and third-party products. If such a customer has a new requirement, it is often not obvious whether this requirement is only valid for this single customer or if this is a request for a feature with general value. Developers in the CLM ecosystem had clear difficulties making decisions about the customer's specific context especially when the customer's development process and culture were different from those that CLM developers experience everyday, thus not being able to rely on their own domain knowledge. This situation is similar to the one found in Fricker's requirements value-chain vision paper [20], where locally empowered development teams actively pulled in requirements to offer value in the ecosystem based on their domain expertise and context-related knowledge. In fact, it is unclear, if any other approach can scale for tomorrows ultra-large-scale software systems [19].

Learning curve and dependence on experience. Product and team leadership remove requirement conflicts and inconsistencies that result from the large set of stakeholders and their different contexts. The complexity of this task imposes high requirements on experience and a strong personal network across the CLM ecosystem, to allow basing decisions on the views of senior people in related projects. Hiring new software managers or promoting developers is made difficult by this steep learning curve.

Crosscutting concerns. Empowering local teams typically increases the need to deal with crosscutting concerns and hidden technical dependencies [56]. In the CLM ecosystem, even if discovered, such crosscutting concerns are difficult to tackle. Descriptions of workitems are often unclear, and it is difficult to identify the person that might help with clarifying. Because differences in time zones do not allow regular synchronous communication, emails or workitem comments are used to identify a developer who might provide clarification, often leading to long email threads over several days without any valuable information.

Emergent Requirements Flow. Emergent stakeholders from across the ecosystem, especially keystone players and consumers, play an important role in the requirements negotiation. If involved, emergent users are catalysts and drive the requirements discussion. However, late emergent contributions can cause disruption and rework in the requirements process.

General solution strategy: 1. Leverage personal excellence. We found that the challenges caused by the tradeoff between acting globally to define a strategy for the ecosystem top down and acting locally to understand and react to context-specific requirements are mitigated by excellent lower and middle management. Successful managers have several years of experience of working in the CLM ecosystem and in addition a strong network throughout the keystones of the ecosystem. By this, they are able to resolve crosscutting concerns and reassign misclassified requirements. Nevertheless, this situation is far from satisfactory. It is increasingly difficult to hire qualified managers that can be effective in a reasonable amount of time. Dealing effectively with context-specific requirements depends partly on luck and requires that the right person becomes aware of an issue at the right time. A more systematic approach to manage and distribute the contextual knowledge is needed to ensure continuous excellence. Such an approach could systematically facilitate bottom-up information flows to support global decisions in the ecosystem, and horizontal information flows to handle crosscutting concerns between actors or even teams [56].

General solution strategy: 2. Get emergent users involved early. Our study findings suggest that the majority of requirements clarification should be done early in the user story's life cycle (i.e. in the first half). As emergent users mostly contribute to requirements clarification and often trigger more discussion, early involvement could significantly reduce the development risk by reducing uncertainty early. Thus, facilitating networking across ecosystem actors, e.g. through developer conference, becomes more and more important.

5.3 Threats to validity

Conclusion validity In our mixed-method approach, different research methods have interacted to provide an in-depth exploration of the CLM ecosystem. Participatory observation helped to inform semi-structured interviews, and both offered input to the qualitative analysis. The findings of the qualitative analysis shaped our quantitative analysis of repository data as well as the manual content analysis. In each step, we reported and motivated our decisions. While this research is still highly specific to its particular circumstances, we hope that other researchers find this information useful when retracing our inquiry in similar environments.

Construct validity To mitigate threats to *construct validity* [29], we examined the CLM ecosystem using terms defined in the extensive treatment of ecosystems edited by Jansen et al. [29], and we triangulated our findings with insights from multiple methods: participatory observation, interviews, and repository analysis.

External validity With respect to *external validity*, our study was on a particular domain of action, and our findings should be regarded as tendencies rather than predictions [62]. We are confident that our results will prove useful for other similar organizations and contexts, i.e. software ecosystems that are neither fully open nor fully closed with respect to requirements-related communication. For example, our conceptualization and quantitative analysis of emergent contributors should be applicable to other scenarios where an issue tracker is used for requirements management and clarification.

In addition, we should mention that we only quantitatively analyzed emergent contributions to RTC and can only speculate that similar behavior would also be found in the data of other IBM CLM product teams.

Internal validity To increase the *internal validity*, our author team closely collaborated during the participatory observation phase to create the interview guide for the semi-structured interviews. All interviews were performed by the same author, who did not participate in transcription and coding of interviews but was available for clarification questions. The other authors searched, reviewed, and defined the themes iteratively and discussed intermediate results regularly with practitioners.

In the same way, the qualitative analysis of the type of contributions from emergent stakeholders was driven by two of the authors in several iterations, and cross-examined by the other authors.

During the analysis of repository data, it was challenging to automatically identify emergent contributors, because while the issue tracker provided a history of changes to the workitem, it did not reveal a history of the contributors' affiliation. Instead, we could only retrieve a snapshot of all contributors' affiliation that was valid at the very end of the timeline of our dataset. Project membership, however, is dynamic, and it is not uncommon for developers to work for a while in one project before getting assigned to a different one. Thus, if a contributor was part of a product team at the time of their contribution, but later changed to a different product team, they could have been falsely classified as emergent. In the same way, we could have missed emergent contributors that were not part of the product team when they made a comment, but later switched into this project team. In the course of this research, we prioritized precision over recall. Hence, we accepted that we would not find all emergent contributors, and focused on mitigating the risk of wrongly classifying a contributor as emergent. For this, we introduced a second criteria: A contributor from product team A that commented on user stories in product team B was not classified

as an emergent contributor if she also did major changes on issues of product team B, such as marking an issue as solved, assigning stakeholders to the workitem. To mitigate this problem, we checked for each user that we classified as emergent, whether this person made a major change to any workitem of the project (i.e. creating and modifying the workitem text or metadata like resolution status) and only identified them as emergent when they did not. This way, we ensure that we only classify persons as emergent who are emergent with respect to our definition and, as a tradeoff, accept that some of the persons that we classify as non-emergent were indeed emergent at the time of their discussion contribution.

During this work, we also checked whether the comment a person made that we classified as emergent was plausible, i.e. we controlled for wrong classification of emergent contributors.

6 Related work

To adequately describe the related work around software ecosystems, we cover three software engineering research areas: open-source software [52], modeling and architecture (e.g. software evolution, architecture, and product lines [5]), and managerial perspectives (e.g. business aspects and co-innovation [27]). Some degree of openness is a precondition for software ecosystems. Different degrees exist, from widely proprietary ecosystems to pure open-source ecosystems [1, 28]. Jansen et al. [25], Bosch et al. [5], and Manikas et al. [40] discuss how to analyze software ecosystems and relationships among the actors of ecosystems. While considering these related works, our study focuses on understanding the implications of the actor relationships on RE practice and vice versa.

The literature discusses both proprietary ecosystems and free-and-open-source ecosystems [42]. RE practice in traditional proprietary software projects (as, e.g., described in [47, 49]) differs significantly from the way requirements are handled in open-source projects, where requirements are post-hoc assertions of functional capabilities and included into the system feature set after their implementation [52]. Our study indicates that although the open-commercial approach of the CLM ecosystem does include a more open way of communicating requirements than in traditional approaches to RE, the requirements processes and flows are different than in open-source projects. The emergent requirements flows generated by the technical implementation at the operational level are complemented by strategic requirements flows that allow the ecosystem to consider the refinement of requirements from high-level, business goals into strategic release planning.

Emergent stakeholders play a major role in driving requirements discussion and decision making. To our knowledge, however, research has so far mainly investigated emergent developers [46, 24, 51] and emergent knowledge [59], while only few works exist that start to investigate the effect of emergent contributors on requirements [35]. This is however an important aspect, as software ecosystems facilitate communication between the various levels of involved organizations and the role of emergent collaboration spans beyond developers and software teams [33]. Users with in-depth domain knowledge, the implicit knowledge about client needs, their business domain and the system's environment [17], must participate even when they span team or geographic boundaries [4]. Our research here adds to this body of knowledge by offering insights into how often contributions from emergent stakeholders occur, what the characteristics of these contributions are, and how they affect requirements clarification.

Transparency has been proposed as a non-functional requirement in order to address the increasing demand of society to understand digital infrastructure in the information age [37]. Our research speaks to the value of transparent requirements information in complex, evolving, commercial systems but also highlights limitations and challenges for such openness in proprietary environments. Despite these challenges, open communication channels have shown their value for building communities around healthy ecosystems [30]. For commercial software ecosystems, this offers an exciting opportunity to improve scalability by facilitating decentralized "just-in-time" RE and to support agile development [36].

Tamburri et al. [58] give a taxonomy of social communities in software development. This work offers a complementary perspective of the requirements landscape in our case study by focussing on the community of stakeholders. Accordingly, our study here exhibits a strong situatedness of developers and managers (development and technical leads), and while in many cases this community is already defined over the organization (a feature team), it can often also be characterized as a (emergent) community of practice. A good example is the technical lead describing the situation of his new formed feature team where he was not officially responsible for the majority of developers recruited (and still belonging to) other teams.

In addition, Tamburri et al.'s [58] work allows us to characterize the community of stakeholders in CLM as *informal network*, based on that it is highly dispersed and has a high degree of informality, e.g. with end users and customers discussing issues with developers through workitem comments.

There is, however, also a formal way of tying customers and key users to the developing organization through the

sales team as well as through the support process, which has to be seen as a formal network, according to [58]. The coexistence of formal and informal network is one source of irritation we encountered during our interviews, since requirements from both sources of information need to be aligned.

7 Conclusion

In this paper, we described RE challenges and practices within the CLM software ecosystem and identified two tradeoffs that openness brings about in software ecosystems. First, open information channels support both global strategic and local just-in-time action. Both global and local actions are needed to make the software ecosystem a competitive business partner, but to allow for both, bottom-up and horizontal information flows need to be dealt with systematically. Second, organizations must find ways to adhere to non-disclosure agreements, protect intellectual property, and maintain confidentiality in such open environments. We also found that openness encourages contributions from emergent stakeholders whose early involvement can be crucial for the success of requirements. Our work addresses a particular lack of RE research in the rapidly growing field of software ecosystems. A good starting point for future work is the development of methods and tool support for commercial organizations to optimize their RE to address the following challenges:

- Manage stakeholder interaction across multiple organizational boundaries and between teams.
- Manage domain and technical knowledge during continuous deployment across all organizational levels and actors.
- Systematically transform requirements flows into technological and strategic decisions to position actors in the software ecosystem.
- Connect the right people early when clarifying requirements for new features in software ecosystems.

References

1. van Angeren J, Kabbedijk J, Popp KM, Jansen S (2012) Managing software ecosystems through partnering. In: Software ecosystems: analyzing and managing business networks in the Software Industry, chap 5. pp 85–102
2. Barbosa O, Pereira R, Alves C, Werner C, Jansen S (2012) A systematic mapping study on software ecosystems through a three-dimensional perspective. In: Software Ecosystems: analyzing and managing business networks in the Software Industry, chap 4. pp 87–129
3. Begel A, Bosch J, Storey M (2013) Bridging software communities through social networking. IEEE Softw 30(1):26–28. doi:10.1109/MS.2013.3
4. Boden A, Avram G (2009) Bridging knowledge distribution-the role of knowledge brokers in distributed software development teams. In: Cooperative and Human Aspects on Software Engineering, 2009. CHASE'09. ICSE Workshop on IEEE, Vancouver, Canada
5. Bosch J (2009) From software product lines to software ecosystems. In: Proceedings of the international conference on software product lines. San Francisco, CA, USA, pp 111–119
6. Braun V, Clarke V (2006) Using thematic analysis in psychology. Qual Res Psychol 3:86–94
7. Brill O, Knauss E (2011) Structured and unobtrusive observation of anonymous users and their context for requirements elicitation. In: Proceedings of the international requirements engineering conference (RE'11). Trento, Italy, pp 175–184
8. Cao L, Ramesh B (2008) Agile requirements engineering practices: an empirical study. IEEE Softw 25(1):60–67
9. Castro-Herrera C, Duan C, Cleland-Huang J, Mobasher B (2008) Using data mining and recommender systems to facilitate large-scale, open, and inclusive requirements elicitation processes. In: Proceedings of the international requirements engineering conference (RE'08). Barcelona, Spain, pp 165–168
10. Corbin J, Strauss C (2008) Basics of qualitative research, 3rd edn. Sage, Beverly Hills
11. Curtis B, Krasner H, Iscoe N (1988) A field study of the software design process for large systems. Commun ACM 31:1268–1287
12. Dabbish L, Stuart C, Tsay J, Herbsleb J (2013) Leveraging transparency. IEEE Softw 30(1):37–43. doi:10.1109/MS.2012.172
13. Damian D (2007) Stakeholders in global requirements engineering: lessons learned from practice. IEEE Softw 24:21–27
14. Damian D, Didar Z (2003) RE challenges in multi-site software development organisations. Requir Eng 8(3):149–160
15. Damian D, Marczak S, Kwan I (2007) Collaboration patterns and the impact of distance on awareness in requirements-centred social networks. In: Proceedings of the international requirements engineering conference (RE'07). New Delhi, India, pp 59–68
16. Damian D, Kwan I, Marczak S (2010) Requirements-driven collaboration: leveraging the invisible relationships between requirements and people. In: Collaborative software engineering. Springer, pp 57–76
17. Damian D, Helms R, Kwan I, Marczak S, Koelewijn B (2013) The role of domain knowledge and hierarchical control structures in socio-technical coordination. In: Proceedings of the international conference on software engineering (ICSE'13). San Francisco, CA, USA, pp 442–451
18. Easterbrook S (1991) Elicitation of requirements from multiple perspectives. PhD thesis, University of London

19. Feiler P, Sullivan K, Wallnau K, Gabriel R, Goodenough J, Linger R, Longstaff T, Kazman R, Klein M, Northrop L, Schmidt D (2006) Ultra-large-scale systems: the software challenge of the future. Software Engineering Institute, Carnegie Mellon University. ISBN:0-9786956-0-7. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30519

20. Fricker S (2010) Requirements value chains: stakeholder management and requirements engineering in software ecosystems. In: Proceedings of the international working conference on requirements engineering: foundation for software quality (REFSQ'10). Essen, Germany, pp 60–66

21. Frost R (2007) Jazz and the eclipse way of collaboration. IEEE Soft 24(6):114–117

22. Gawer A (2009) Platforms, markets, and innovation. Edward Elgar, Cheltenham

23. Guzman E, Maalej W (2014) How do users like this feature? a fine grained sentiment analysis of app reviews. In: Proceedings of 22nd international requirements engineering conference (RE '14). Karlskrona, Sweden, pp 153–162

24. Haenni N, Lungu M, Schwarz N, Nierstrasz O (2014) A quantitative analysis of developer information needs in software ecosystems. In: Proceedings of the 2014 European conference on software architecture workshops (ECSAW '14), vol 12. ACM, Vienna, Austria, pp 12:1–12:6. doi:10.1145/2642803.2642815

25. Jansen S, Cusumano MA (2012) Defining software ecosystems: a survey of software platforms and business network governance. In: Software ecosystems: analyzing and managing business networks in the software industry, chap 1. pp 13–28

26. Jansen S, Finkelstein A, Brinkkemper S (2009) A sense of community: a research agenda for software ecosystems. In: Proceedings of the international conference on software engineering. NIER Track, pp 187–190

27. Jansen S, Brinkkemper S, Finkelstein A (2012a) Business Network Management as a Survival Strategy. In: Software ecosystems: analyzing and managing business networks in the software industry, chap 2. pp 29–42

28. Jansen S, Brinkkemper S, Souer J, Luinenburg L (2012b) Shades of gray: opening up a software producing organization with the open software enterprise model. J Syst Softw 85:1495–1510

29. Jansen S, Cusumano MA, Brinkkemper S (eds) (2012c) Software ecosystems: analyzing and managing business networks in the software industry. Edward Elgar, Cheltenham

30. Kilamo T, Hammouda I, Mikkonen T, Aaltonen T (2012) Open source ecosystems: a tale of two cases, chap 13. In: Jansen S, Brinkkemper S, Cusumano M (eds) Software ecosystems. Analyzing and managing business networks in the software industry. pp 276–306

31. Knauss E, Damian D, Poo-Caamaño G, Cleland-Huang J (2012) Detecting and classifying patterns of requirements clarifications. In: Proceedings of the international requirements engineering conference (RE'12). Chicago, pp 251–260

32. Knauss E, Damian D, Cleland-Huang J, Helms R (2014a) Patterns of continuous requirements clarification. Requir Eng J. doi:10.1007/s00766-014-0205-z

33. Knauss E, Damian D, Knauss A, Borici A (2014b) Openness and requirements: opportunities and tradeoffs in software ecosystems. In: Proceedings of 22nd international requirements engineering conference (RE '14). Karlskrona, Sweden, pp 213–222. doi:10.1109/RE.2014.6912263

34. Krippendorff K (2003) Content analysis: an introduction to its methodology. Sage, Cheltenham

35. Kwan I, Damian D (2011) The hidden experts in software-engineering communication: NIER track. In: Proceedings of the international conference on software engineering (ICSE'11). Waikiki, Honolulu, Hawaii, USA, pp 800–803. doi:10.1145/1985793.1985906

36. Lee M (2002) Just-in-time requirements analysis—the engine that drives the planning game. In: Proceedings of the 3rd international conference on extreme programming and agile processes in software engineering. Alghero, Italy, pp 138–141

37. Leite JCSdP, Cappelli C (2010) Software transparency. Bus Inf Syst Eng 2(3):127–139

38. Liskin O, Herrmann C, Knauss E, Kurpick T, Rumpe B, Schneider K (2012) Supporting acceptance testing in distributed software projects with integrated feedback systems: experiences and requirements. In: Proceedings of the international conference on global software engineering. Porto Alegre, Rio Grande do Sul, Brazil, pp 84–93

39. Maalej W, Thurimella AK (2009) Towards a research agenda for recommendation systems in requirements engineering. In: Proceedings of the international workshop on managing requirements knowledge (MaRK'09). Atlanta, USA, pp 32–39

40. Manikas K, Hansen KM (2013a) Characterizing the Danish telemedicine ecosystem: making sense of actor relationships. In: Proceedings of the international conference on management of emergent digital ecosystems (MEDES'13). Neumünster Abbey, Luxembourg, pp 211–218

41. Manikas K, Hansen KM (2013b) Reviewing the health of software ecosystems–a conceptual framework proposal. In: Proceedings of the international workshop on software ecosystems. Potsdam, Germany, pp 33–44

42. Manikas K, Hansen KM (2013c) Software ecosystems: a systematic literature review. Syst Softw 86:1294–1306

43. Marczak S, Kwan I, Damian D (2009) Investigating collaboration driven by requirements in cross-functional software teams. In: Requirements: communication, understanding and softskills, 2009 collaboration and intercultural issues on, IEEE. pp 15–22

44. Menzies T, Easterbrook S, Nuseibeh B, Waugh S (1999) An empirical investigation of multiple viewpoint reasoning in requirements engineering. In: Proceedings of the international symposium on requirements engineering. IEEE Computer Society Press, pp 7–11

45. Milne A, Maiden N (2012) Power and politics in requirements engineering: Embracing the dark side? Requir Eng J 17(2):83–98

46. Minto S, Murphy GC (2007) Recommending emergent teams. In: Proceedings of the international workshop on mining software repositories (MSR'07). Minneapolis, USA. doi:10.1109/MSR.2007.27

47. Robertson S, Robertson J (1999) Mastering the Requirements Process. Addison-Wesley, Reading

48. Rossberg J, Olausson M (2012) Pro application lifecycle management with visual studio 2012, 2nd edn. Apress, New York

49. Ruhe G (2010) Product release planning: methods, tools and applications. CRC Press, Boca Raton

50. Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14:131–154

51. Sadi MH, Dai J, Yu E (2015) Designing software ecosystems: How to develop sustainable collaborations? scenarios from apple ios and google android. In: Proceedings of CAiSE 2015 workshops, vol 215. Stockholm, Sweden, LNBIP, pp 161–173. doi:10.1007/978-3-319-19243-7_17

52. Scacchi W (2009) Understanding requirements for open source software. In: Proceedings of design requirements workshop, vol 14. Springer LNBIP, pp 467–494

53. Schäfer MO, Melich M (2011) SAP solution manager. SAP Press, Quincy

54. Schneider K (2011) Focusing Spontaneous Feedback to Support System Evolution. In: Proceedings of the international requirements engineering conference (RE'11). Trento, Italy, pp 165–174

55. Schneider K, Meyer S, Peters M, Schliephacke F, Mörschbach J, Aguirre L (2010) Feedback in context: supporting the evolution

of IT-ecosystems. In: International conference on product focused software process improvement, pp 191–205

56. Sekitoleko N, Evbota F, Knauss E, Sandberg A, Chaudron M, Olsson HH (2014) Technical dependency challenges in large-scale agile software development. In: proceedings of the international conference on agile software development. Rome, Italy

57. Seyff N, Maiden N, Karlsen K, Lockerbie J, Grünbacher P, Graf F, Ncube C (2009) Exploring how to use scenarios to discover requirements. Requir Eng 14(2):91–111

58. Tamburri D, Lago P, van Vliet H (2013) Uncovering latent social communities in software development. IEEE Softw 30(1):29–36. doi:10.1109/MS.2012.170

59. Treude C (2012) The role of social media artifacts in collaborative software development. PhD thesis. University of Victoria, Victoria

60. Tuunanen T, Rossi M (2004) Engineering a method for wide audience requirements elicitation and integrating it to software development. In: Proceedings of the hawaii international conference on system sciences, vol 7. doi:10.1109/HICSS.2004.1265420

61. Valenca G, Alves C, Heimann V, Jansen S, Brinkkemper S (2014) Competition and collaboration in requirements engineering: a case study of an emerging software ecosystem. In: Proceedings of 22nd international requirements engineering conference (RE '14). Karlskrona, Sweden, pp 384–393

62. Walsham G (1995) Interpretive case studies in is research: nature and method. Eur J Inf Syst 4:74–81

63. Williams C, Wagstrom P, Ehrlich K, Gabriel D, Klinger T, Martino J, Tarr P (2010) Supporting enterprise stakeholders in software projects. In: Proceedings of workshop on cooperative and Human Aspects of Software Engineering. Cape Town, South Africa, pp 109–112