Openness and Requirements: Opportunities and Tradeoffs in Software Ecosystems

Eric Knauss
Department of Computer Science and Engineering
Chalmers | University of Gothenburg, Sweden
eric.knauss@cse.gu.se

Daniela Damian, Alessia Knauss, Arber Borici
Department of Computer Science, University of Victoria
PO Box 1700, STN CSC, Victoria, BC V8W 2Y2, Canada
{danielad, alessiak, borici}@uvic.ca

***Abstract*—A growing number of software systems is characterized by continuous evolution as well as by significant interdependence with other systems (e.g. services, apps). 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 some challenges to Requirements Engineering (RE) within the ecosystem. We report from a mixed-method study of IBM®'s CLM® ecosystem that uses an open commercial development model. We analyzed data from from interviews within several ecosystem actors, participatory observation, and software repositories, to 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, 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 the openness in software ecosystems: i) allowing open, transparent communication while keeping intellectual property confidential within the ecosystem, and ii) 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.**

***Index Terms*—requirements engineering; software ecosystem; mixed method**

I. Introduction

Software ecosystems promise to address the inherent complexity of large-scale software projects by removing the need for centralized management [1]. By opening up and attracting 3^{rd}-party actors into joining a software ecosystem, an organization increases its innovation power and market reach [2]. The strategies to initiate and maintain a software ecosystem vary in their degree of *openness* and range from following widely proprietary, closed information flows around a defined set of partners (e.g. SAP) to the more open information flows found in open source ecosystems (e.g. IBM's Eclipse) [3].

More recently a prevalent strategy along this spectrum of openness in ecosystems is the open commercial approach [5] (a.k.a Extended Software Enterprises [6]), where 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. Abolishing selected barriers the ecosystem facilitates building of communities and product adoption, and provides advantage over the non-open competitors [6], [7]. The increased transparency supports learning from observation and reputation management [8].

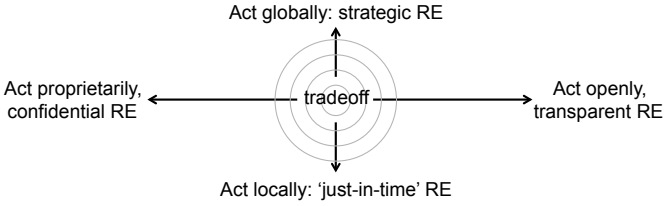

Fig. 1. Tradeoffs in open commercial software ecosystems.

The openness in these types of software engineering environments would seem to support decentralized ways of facilitating the flow of requirements among stakeholders, similar to those found in open source projects [7]. However, open-commercial enterprises are different from open source projects. The scale and complexity of stakeholder relationships – governed by contractual considerations and commercial business models – creates problems in managing diverse sources of requirements and information flows, while the ecosystem's openness results in lowered ability to protect product strategic information. To the best of our knowledge no systematic study examined RE in software ecosystems, although systematic literature reviews have reported on many other areas concerning software ecosystems [9], [10], leaving the description of RE processes within software ecosystems an outstanding research problem [11]. In this paper, we report from a mixed-method empirical study of IBM®'s open-commercial software ecosystem CLM® (Collaborative Lifecycle Management) *to identify requirements engineering challenges and practices in an open-commercial software ecosystem.*

Our contribution is two-fold: First, we provide a detailed description of RE processes and how requirements flow through the IBM CLM software ecosystem; specifically, we identify challenges around stakeholder selection, communication and prioritization of requirements, managing context, and mapping requirements to the ecosystem's actors. Secondly, we identify two main underlying tradeoffs in open commercial ecosystems (Figure I): (1) Maintaining the openness and transparency in the ecosystem needs to be balanced with keeping the confiden-

978-1-4799-3032-6/14/$31.00 © 2014 IEEE

RE 2014, Karlskrona, Sweden

tiality of paying customers' business needs and (2) responding to market demands requires the ability to globally define requirements on the strategic level, while scale and complexity of software ecosystems require self-organized teams' ability to locally and timely address context-specific customer needs 'just-in-time'. 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.

II. Background and Research Questions

A. The IBM CLM Software Ecosystem

The IBM Collaborative Lifecycle Management (CLM) tool suite is characterized by a) the *Jazz platform*®, a service oriented platform based on open standards, b) its open communication channels that allow all stakeholders to participate in discussions of work in progress, and c) its goal to be a strategic platform for its customers and to support seamless interaction between tools and phases in the software development lifecycle. Jansen et al. refer to such systems as software ecosystems and define them as follows [12]:

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

The CLM ecosystem consists of a number of products (*set of actors in a shared market*) especially designed to interoperate on the Jazz platform (*technological platform*). These products include[1] Rational Requirements Composer (RRC), Rational Quality Manager (RQM), Rational Software Architect (RSA–now: Design Manager), Rational Team Concert (RTC), and smaller products like Reporting which integrates Rational Insight capabilities in the ecosystem. IBM Rational® (an IBM brand) coordinates this ecosystem and offers integration adapters to connect the CLM ecosystem to other major players and ecosystems in the field. IBM Rational also runs a partner program to allow third parties to offer products with certified compatibility to the Rational CLM products. For this reason, customers can have highly customized environments defined by the mixture of Rational and third party products that work together in order to provide the required services.

The CLM ecosystem is particularly interesting for investigating RE in software ecosystems, because of its open communication channels that allow transparent developer-customer communication: (i) a publicly available issue tracker with rich discussion forums around features, (ii) wikis with technical documentation from CLM developers, and (iii) open chat rooms. They enable customers to articulate doubts about a certain change as well as developers to learn from end-user comments about their specific needs and how they use different services in the software ecosystem. Developers also openly discuss features they are currently developing based on a large-scale agile development process and by this they actively promote the ecosystem and its actors [5].

[1] https://jazz.net/products/clm/ – visited 2014-3-4

B. An Illustrative Example

We will illustrate the complex RE environment in the CLM ecosystem with the following (fictive but realistic) **running example:**

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.

C. Requirements Engineering in Software Ecosystems

We examine practices and challenges of requirements management in the CLM's open commercial paradigm based on Jansen et al.'s proposal to analyze software ecosystems on three different scope levels [6]. A full and complete description of any software ecosystem is very difficult to achieve. In Figure 2, we describe the entities and relationships in the CLM ecosystem that serve our research investigation. Section IV describes the workflows depicted in the main part of Figure 2 as well as the practices and challenges we identified on each scope level. At the top of Figure 2, the CLM ecosystem is shown on Scope Level 1 in the context of neighbouring ecosystems such as Github and SAP (circles group actors to ecosystems), on Scope Level 2 with a focus on its internal structure, and on Scope Level 3 with a focus on a single actor (here: RQM) and its direct neighbours. Links between actors indicate that the actor on the right uses services of the actor on the left.

Scope Level 1 is the highest level of abstraction and offers an external view on the ecosystem (Figure 2, top-left). The scope at this level is defined based on differentiating features such as the technological platform, the target market, the participants, and its connectedness to other software ecosystems. The figure shows CLM ecosystem actors (RTC, RRC, RSA, RQM, Reporting) as well as actors of adjacent software ecosystems around Git, Jira, or SAP platforms.

RE can have an impact on Scope Level 1 by facilitating integration or attraction of new actors into the ecosystem as well as approaching new markets, e.g. by adding strategic features to the roadmap or adjusting the technological platform. Further, these requirements need to be communicated throughout the software ecosystem.

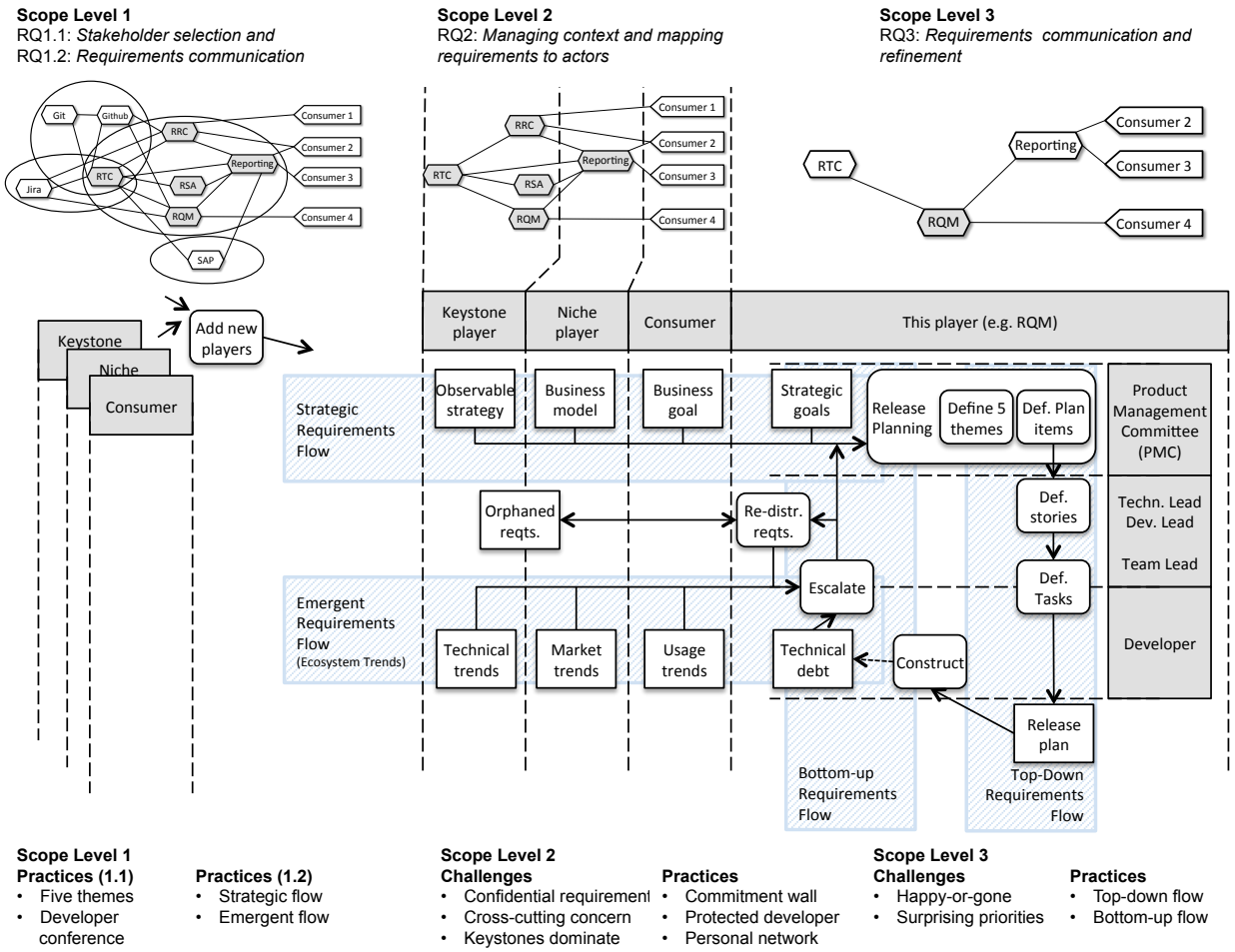

Fig. 2. Research questions, findings (flow of requirements, challenges, practices) in the context of software ecosystem scope levels.

Example: *Alice Inc. might require certain features in the existing products to make CLM services attractive for the automotive domain. If the CLM ecosystem (by purpose or because of other priorities) fails to prioritize these features on its roadmap, or if the technological platform hinders Alice Inc. in offering stable services to its customers, its economic survival might be endangered. If Alice Inc. is perceived as a strategic partner, its requirements need to be communicated to all relevant actors, such as RTC or even the Jazz platform.*

Previous work proposed that stakeholders, their relation, and their communication can be analyzed as requirements value chains [11] or based on social network analysis [13]. Yet, how stakeholders are selected and how requirements flow through software ecosystems in practice remains an open question:

RQ 1.1: How are stakeholders selected across the software ecosystem?
RQ 1.2: How are requirements communicated across the software ecosystem?

Scope Level 2 offers an internal view on the ecosystem (top-middle in Figure 2) that allows analyzing roles of the actors and their relations. Jansen et al. [6] distinguish three roles of actors in software ecosystems: *Dominators* control value capture and creation of the ecosystem and tend to expand by taking over functions of other ecosystem actors. By eliminating those other actors they reduce diversity in the ecosystem and are therefore considered harmful [14]. *Keystone players* create or occupy highly connected hubs of actors. They are considered beneficial to the ecosystem health by providing value to surrounding actors as well as increasing variability and robustness [14]. *Niche players* draw value from the keystones and aim to separate from other niche players by developing special functions [14]. To support our perspective on RE, we include *consumers* as a fourth role, characterized by consuming services without offering own services in the scope of the ecosystem. Table I gives an overview of actors, their roles, and their typical stakeholders in the CLM ecosystem.

From a RE perspective, it is important to understand how actors position themselves as niche or keystone players. Related work suggests that a significant amount of functionality in software ecosystems is based on the interplay of several subsystems [15]–[17]. The way different actors offer services to consumers by re-using other services in the ecosystem de-

TABLE I
STAKEHOLDERS IN THE SCOPE OF DIFFERENT CLM ACTORS.

Actor	Stakeholder	Role in RE activities
Consumer *(e.g. a customer)*	Manager	Source of strategic reqts. and goals
	End-user	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.
	Productmgr.	Source of strategic requirements for the product.
Niche *(e.g. Reporting)*	Same as keystone, but highly affected by technical decisions of keystones.	

fines the context in which the consumer's requirements evolve and how difficult it will be to map those 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 the new actor directly or by one of the more fundamental service providers such as RTC or RQM. It is not clear however, how they are forwarded to the actor in the ecosystem that can handle them most effectively. Also, the actor who is finally working on the feature request does not necessarily have the required domain knowledge.*

The open-commercial paradigm provides end-users with a set of different channels to articulate changed or new needs [5], but challenges end-users and customers to send such requests to the correct recipient [15], [16]. The fact that software ecosystems are constantly evolving causes the ideal recipient for a given request to change over time, thus making it necessary to rethink the way requirements travel through the software organizations and which roles are responsible to seek alignment of goals [11]. This alignment of goals requires understanding requirements in their context, for which related work focuses on contextual techniques and observation. Requirements can be captured together with their relevant context by observing end-users interacting with their system [18], [19]. Feedback systems allow end-users to articulate their needs themselves, enriched with screenshots and other context descriptions [20]. In addition, mobile devices allow to present and discuss scenarios in situ, i.e. in the context of use [21]. It remains an open question how such approaches can be applied by practitioners to manage the context of requirements and to support software managers and developers to actively align their development efforts with needs of relevant stakeholders (e.g. by mapping feature requests to particular actors in the ecosystem) at the scale of modern software ecosystems.

RQ 2: How are requirements mapped to ecosystem actors and how is the context of requirements managed?

Scope level 3 offers an organization centric perspective. Figure 2, top right corner shows the immediate surrounding of the RQM actor. This scope level allows to investigate RE of single actors on a more tactical or operational level.

For RE in software ecosystems, it is unclear how actors can systematically understand the level of satisfaction of their external stakeholders (customers and end-users) and forward this information to internal stakeholders (software managers and developers) in a complex software ecosystem environment.

Example: *Even for Alice Inc., it is difficult to understand the satisfaction of a potentially large and heterogenous stakeholder landscape in the automotive domain. More central actors like RTC or RQM might in addition suffer from the fact that important feedback is only given to other actors.*

Agile and continuous development, especially on a large scale, implies continuous clarification of requirements by software managers and developers throughout the development lifecycle [22]. Especially in distributed development, effective communication of requirements requires a high level of transparency [23], [24] to 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 [25]. It is not clear 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.

RQ 3: How are requirements communicated and refined in the scope of an actor in the software ecosystem?

III. RESEARCH METHODOLOGY

We employed a mixed-method research methodology to examine the characteristics of the open-commercial develpoment in the CLM ecosystem. Our data collection methods included participatory observation, semi-structured interviews and analysis of software repositories.

1) Participatory observation: To obtain a broad view on the actors in the ecosystem and also more in-depth knowledge with the development processes and practices within one specific actor of the ecosystem, one of the co-authors worked as an intern and developer in the Reporting team at IBM Ottawa for two months. During that time, he observed work practices through his direct involvement in solving technical issues but also through participation in project meetings. He kept a daily journal and logged information on how requirements were analyzed by the development team within this ecosystem actor but also details of 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 for the next phase in our data collection.

2) Semi-structured interviews: In order to explore the RE landscape in software ecosystems, we conducted a series of 13 semi-structured interviews. 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 levels, including six developers, two development leaders, four technical leaders, and one team leader. Experience of our interviewees in their role ranged from 6 month to 15 years, with an average of 2-3 years. Our interviews were based on a semi-structured interview guide[2] that covered our research questions, but allowed us to pursue related topics our interviewees brought up [26].

The interviews were conducted by the co-author who also did the participatory observation. The other co-authors iteratively analyzed the interview data based on the thematic analysis approach [27] as follows: We transcribed and read recordings from all interviews to get familiarized with the data. One of the co-authors then coded the data from the perspective of the research questions and started searching for initial themes by grouping the initial codes. In a series of workshops, we then reviewed the initial themes, 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. We defined a label for each theme 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) Analyzing online repository of workitems: One of the richer source 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 the other two methods. 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. In our analysis we analyzed workitems and their meta-data (name and if available the 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.

IV. Findings

Our findings (Figure 2) include the flow of requirements and the themes we identified in our analysis of RE in the CLM ecosystem. For the ecosystem's Scope Levels 2 and 3, our themes include both challenges and practices. For Level 1, however, our report focuses on practices for our interviewees, software managers and developers in various ecosystem actors, although able to describe the larger context of their work, were not aware of specific challenges at this high level.

[2]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?

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

Software product release planning has been extensively discussed (e.g. [28]). In our interviews we focused on the specifics of creating a roadmap for a software ecosystem actor and 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 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. Thus, *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: Each IBM internal actor is 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 sets the goal for the next release by defining *five themes* based on strategic considerations to guide the definition of the roadmap for the next release. By including sales and support, knowledge about strategic needs of potential new actors are included in this discussion. These *five themes* can be considered as the primary steering instrument, which ultimately allows to 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 to shape the attractiveness of the ecosystem to new actors.

Developer conferences: Conferences aim at bringing together technical leaders of all actors in the ecosystem to allow discussion of 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 who participate in these yearly events come from all ecosystem actors (keystone players and consumers) and meet for three days. Many of them are part of the PMCs (or equivalent in non-IBM actors) and can shape the *five themes* and the roadmap accordingly.

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

We found two general information flows that introduce requirements to the ecosystem. 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 (Fig. 2).

Strategic requirements flow: Business goals from consumers as well as strategic information from other actors in the ecosystem are introduced into the ecosystem via sales or support activities and need to be considered during release planning. As in traditional RE (e.g. [28], [29]), this flow focuses on systematic analysis of business goals from stakeholders and their refinement to detailed requirements. 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 of the ecosystem causes many new requirements to be based on *ecosystem trends* and introduced into the ecosystem as *emergent requirements flow*. Sources are end-users at consumers, who find new ways of using existing services and developers at other keystone or niche players, who come up with innovative solutions. Such 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. This requirements flow resembles the requirements practice in open source projects [30] where requirements emerge as informalisms, through continually emerging webs of software discourse (e.g. email and discussion forums).

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

Our findings for this research question are shown in the center of Figure 2 (Scope Level 2). We found managing context and mapping requirements to actors to be two highly interconnected and challenging RE tasks on Scope Level 2 and identified three challenges that express how constraints and complexity of open-commercial software ecosystems affect the engineering of requirements:

Confidential requirement: In a commercial setting, customer specifics and requirements often cannot be discussed on open-commercial channels because they contain confidential information about the customer. Development or team leads manage 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) has to be removed from 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?"*

Cross-cutting concern: Practitioners frequently struggle to understand if requests of different customers could be addressed generally (closer to the platform) or only specifically (by a peripheral actor). This includes dealing with cross-cutting concerns or requirements that should entirely be assigned to other actors in the software ecosystem. Especially, 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 integration of other actors' services. Thus, they become dependent on these other actor's technical decisions which can have more impact on the niche player's requirements than 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 and if it is encountered too frequently, this challenge can have a serious effect on the ecosystem's overall health [14]. Keystone players should be considerate in their action. While each major design decision can offer some of the actors in the ecosystem new opportunities, 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 [14]. 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 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 promise 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 enable developers to receive feedback from end-users directly, 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. Interception and resolution of these challenges by seniors allows 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 are effectively shielding their developers from any challenges arising from context specific requirements.

Personal network: Mapping of emergent requirements to actors in the ecosystem becomes difficult, especially because they are often cross-cutting over several players in the ecosystem. It is important that the context is well understood and that requirements are systematically forwarded to relevant other players in the ecosystem. Our interviews indicate that at the moment, 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, thus being able to discover and resolve cross-cutting 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 even actively track open communication channels of other actors to identify cross-cutting 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 on a large part on their personal experience and network and we found no systematic approach for capturing and managing context. According to our interviewees, 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.

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 Figure 2), developing teams struggle with the complexity of the software ecosystem and with the consequences of practices on Scope Level 1 and 2. For example, *protecting developers* (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 with setting up a representative testframe or reproducing 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: Especially junior developers heavily rely on their managers and senior developers in the team to navigate 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."

Happy-or-gone: Priorities become even more surprising by the huge amount of different information channels, which are only partly open, and due to the lack of systematic feedback about stakeholder satisfaction.

Strong sales and support processes are in place to understand stakeholder needs and user experience teams can provide further input, but there is a lack of transparency and systematic approaches for channeling stakeholder feedback. If there is a problem, development teams usually receive consumers' complaints, but if there is no complaint, they do not know if a consumer is happy with a service or 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 relying 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 Figure 2. The PMC works with so called plan items to create a release plan. Plan items are special workitems or tasks in an issue tracker. They 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 who then define *tasks* based on these stories and assign them to developers. During this final refinement, most of the cross-cutting 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 is not sufficient to address the software ecosystem challenges.

Bottom-up requirements flow: In order to deal with challenges discussed on both Scope Level 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 or to discuss their needs in the comment stream of a workitem.

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

This 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 formal processes or other systematic approaches, *personal network* and experience of team leads and senior developers are 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 cross-cutting concerns as *bottom-up requirements flow* (on the right in Figure 2).

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

V. Discussion: RE-Related Tradeoffs in Open Commercial Software Ecosystems

The observations and findings in our study are of course specific to the CLM ecosystem. However, they reveal underlying tradeoffs that openness brings to how requirements are managed within software ecosystems. We discuss, for each tradeoff, the different forces that need to be balanced and ways in which IBM teams approached these challenges. We therefore intend that these tradeoffs and strategies have implications for RE practices in software ecosystems in general.

Tradeoff 1: Act Openly vs. Act Proprietarily

The open-commercial approach in the CLM ecosystem means that customers and end-users, in fact every stakeholder interested in the development, has access to the issue tracker and can see the current progress of the project. 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 jazz.net. For the development, it is important to know the context of this report: how is it embedded in the customer's context 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 are 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 discussion in open communication channels is 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.

Tradeoff 2: Act Globally vs. Act Locally

The CLM ecosystem is in direct or indirect competition with other lifecycle management solutions, e.g. *Visual Studio Application Lifecycle Management* [31] or *SAP Solution Manager* [32]. 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 [21]. Context in the CLM ecosystem depends on various factors, and descriptions of workitems frequently showed 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 3^{rd} 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 was 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 [11], 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 [1].

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.

Cross-cutting concerns: Empowering local teams typically increases the need to deal with cross-cutting concerns and hidden technical dependencies [33]. In the CLM ecosystem, even if discovered, such cross-cutting 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 timezones 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.

General solution strategy: *Rely on 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 cross-cutting 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 cross-cutting concerns between actors or even teams [33].

Threats to Validity

To mitigate threats to *construct validity* [26], we examined the CLM ecosystem using terms defined in the extensive treatment of ecosystems edited by Jansen et al. [4], and triangulated our findings with insights from multiple methods: participatory observation, interviews and repository analysis.

With respect to *external validity*, our study has been of a particular domain of action and our findings should be regarded as tendencies rather than predictions [34]. 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.

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.

VI. Related work

In understanding ecosystems, one would draw on three fields in software engineering: open source software [30], modelling and architecture (e.g. software evolution, architecture, and product lines [35]), and managerial perspectives (e.g. business aspects and co-innovation [6]). Software ecosystems imply some degree of openness and different strategies exist from widely proprietary ecosystems based on a semi-open partnership program to pure open source ecosystems [3], [7]. Related works [35]–[37] discuss how to analyze software ecosystems and relationships among their actors. While influenced by these studies, we focus on understanding implications these relationships have on RE practice and vice versa.

Literature discusses almost as many proprietary ecosystems as free-and-open-source ecosystems [10]. RE practice in traditional proprietary software projects (as e.g. described in [28], [29]) 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 [30]. 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.

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 [38]. 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 [39]. For commercial

software ecosystems this offers an exciting opportunity to improve scalability by facilitating decentralized 'just-in-time' RE and to support agile development [40].

VII. Conclusion

In this paper we described RE challenges and practices within the CLM software ecosystem, and identified two trade-offs that openness brings about in software ecosystems. Open information channels support both global strategic and local just-in-time action, but the openness conflicts with the need to act as a reliable business partner and according to non-disclosure agreements about intellectual property and confidential information. Both global and local action is 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. 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/actors.
- Systematically transform requirements flows into strategic and technological decisions to position actors in the software ecosystem.

Acknowledgement

We thank the various participating IBM teams, and all researchers and practitioners for their feedback on this work: the SEGAL group at University of Victoria, IBM research, and the Division of Software Engineering in Gothenburg. This research was partly funded by the NECSIS Network, Canada.

References

[1] P. Feiler, R. P. Gabriel et al., *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute, 2006.

[2] A. Gawer, *Platforms, Markets, and Innovation*. Edward Elgar, 2009.

[3] J. van Angeren, J. Kabbedijk et al. *Managing software ecosystems through partnering*, in [4], 85–102.

[4] S. Jansen, M. A. Cusumano, and S. Brinkkemper, Eds., *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Cheltenham, UK: Edward Elgar, 2012.

[5] R. Frost, "Jazz and the eclipse way of collaboration," *IEEE Software*, vol. 24, no. 6, 114–117, 2007.

[6] S. Jansen, S. Brinkkemper, and A. Finkelstein, *Business Network Management as a Survival Strategy*, in [4], 29–42.

[7] S. Jansen, S. Brinkkemper et al. "Shades of gray: Opening up a software producing organization with the open software enterprise model," *The Journal of Systems and Software*, vol. 85, p. 1495–1510, 2012.

[8] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Leveraging transparency," *IEEE Software*, vol. 30, no. 1, 37–43, 2013.

[9] O. Barbosa, R. Pereira et al. *A Systematic Mapping Study on Software Ecosystems through a Three-dimensional Perspective*, in [4], 87–129.

[10] K. Manikas and K. M. Hansen, "Software ecosystems: A systematic literature review," *Systems and Software*, vol. 86, 1294–1306, 2013.

[11] S. Fricker, "Requirements Value Chains: Stakeholder Management and Requirements Engineering in Software Ecosystems," in *Proc. of Requir. Eng.: Foundation for Softw. Quality*, Essen, Germany, 2010, 60–66.

[12] S. Jansen, A. Finkelstein, and S. Brinkkemper, "A sense of community: A research agenda for software ecosystems," in *Proc. of Int'l Conf. on Softw. Eng.*, 2009, NIER Track.

[13] D. Damian, S. Marczak, and I. Kwan, "Collaboration Patterns and the Impact of Distance on Awareness in Requirements-Centred Social Networks," in *Proc. of Int'l Requir. Eng. Conf.*, 2007.

[14] K. Manikas and K. M. Hansen, "Reviewing the Health of Software Ecosystems - A Conceptual Framework Proposal," in *Proc. of Int'l Wksp on Softw. Ecosys.*, Potsdam, Germany, 2013, 33–44.

[15] C. Castro-Herrera, C. Duan, et al. "Using Data Mining and Recommender Systems to Facilitate Large-Scale, Open, and Inclusive Requirements Elicitation Processes," in *Proc. of Int'l Requir. Eng. Conf.*, Barcelona, Spain, 2008, 165–168.

[16] K. Schneider, "Focusing Spontaneous Feedback to Support System Evolution," in *Proc. of Int'l Requir. Eng. Conf.*, Trento, 2011, 165–174.

[17] K. Schneider, S. Meyer, et al. "Feedback in Context: Supporting the Evolution of IT-Ecosystems," in *PROFES*, 2010, 191–205.

[18] W. Maalej and A. K. Thurimella, "Towards a Research Agenda for Recommendation Systems in Requirements Engineering," in *Proc. of Int'l Wksp on Managing Requir. Know.*, 2009, 32–39.

[19] O. Brill and E. Knauss, "Structured and Unobtrusive Observation of Anonymous Users and their Context for Requirements Elicitation," in *Proc. of Int'l Requir. Eng. Conf.*, Trento, Italy, 2011, 175–184.

[20] O. Liskin, C. Herrmann, et al. "Supporting acceptance testing in distributed software projects with integrated feedback systems: Experiences and requirements," in *Proc. of Int'l Conf. on Global Softw. Eng.*, 2012.

[21] N. Seyff, N. Maiden, et al. "Exploring how to use scenarios to discover requirements," *Requir. Eng.*, vol. 14, no. 2, 91–111, 2009.

[22] E. Knauss, D. Damian, et al. "Detecting and Classifying Patterns of Requirements Clarifications," in *Proc. of Int'l Requir. Eng. Conf.*, Chicago, USA, 2012, 251–260.

[23] D. Damian and D. Zowghi, "RE challenges in multi-site software development organisations," *Requir. Eng.*, vol. 8, 149–160, 2003.

[24] D. Damian, "Stakeholders in global requirements engineering: Lessons learned from practice," *IEEE Software*, vol. 24, 21–27, 2007.

[25] T. Tuunanen and M. Rossi, "Engineering a Method for Wide Audience Requirements Elicitation and Integrating It to Software Development," in *Proc. of Hawaii Int'l Conf. on System Sciences*, vol. 7, Jan. 2004.

[26] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Emp. SE*, (14) 131–154, 2009.

[27] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, 86–94, 2006.

[28] G. Ruhe, *Product Release Planning: Methods, Tools and Applications*. CRC Press, 2010.

[29] S. Robertson and J. Robertson, *Mastering the Requirements Process*. Addison-Wesley, 1999.

[30] W. Scacchi, "Understanding requirements for open source software," in *Proc. of Design Reqts. Wksp*. Springer LNBIP 14, 2009, p. 467–494.

[31] J. Rossberg and M. Olausson, *Pro Application Lifecycle Management with Visual Studio 2012*, 2nd ed. Apress, 2012.

[32] M. O. Schäfer and M. Melich, *SAP Solution Manager*. SAP Press, 2011.

[33] N. Sekitoleko, F. Evbota, et al. "Technical Dependency Challenges in Large-Scale Agile Software Development," in *Proc. of Int'l Conf. on Agile Softw. Dev.*, Rome, Italy, 2014.

[34] G. Walsham, "Interpretive case studies in is research: nature and method," *Eur. J. Inf. Syst.*, vol. 4, 74–81, 1995.

[35] J. Bosch, "From Software Product Lines to Software Ecosystems," in *Proc. of Int'l Conf. on Softw. Product Lines*, 2009.

[36] S. Jansen and M. A. Cusumano, *Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance*, in [4], 13–28.

[37] K. Manikas and K. M. Hansen, "Characterizing the danish telemedicine ecosystem: Making sense of actor relationships," in *Proc. of MEDES'13*, Neumünster Abbey, Luxembourg, 2013, 211–218.

[38] J. C. S. d. P. Leite and C. Cappelli, "Software transparency," *Business & Information Systems Engineering*, vol. 2, no. 3, 127–139, 2010.

[39] T. Kilamo, I. Hammouda, et al. *Open source ecosystems: a tale of two cases*, in [4], 276–306.

[40] M. Lee, "Just-in-time requirements analysis—the engine that drives the planning game," in *Proc. of Int'l Conf. Extreme Programming and Agile Proc. in Software Eng.*, Alghero, Italy, 2002, 138–141.