Economics of Collectives

Michael Weiss
SCE, Carleton University
Ottawa, Canada
weiss@sce.carleton.ca

ABSTRACT

The transition from a software product line to a software ecosystem, as reported by Bosch [5], takes place, when the product line company makes its platform available to developers outside the company. A similar transition takes place from a software ecosystem to a collective, when the platform is jointly created and owned by a group of members. Building on the literature on software product line economics, this research identifies three factors affecting the economics of collectives (level of contribution, number of members, and diversity of use), and develops a model linking those factors to three economic outcomes (time, quality, and cost).

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics; D.2.9 [**Software Engineering**]: Management

General Terms

Measurement, Management

Keywords

Software product lines, software ecosystems, collectives, product line economics, open source software

1. INTRODUCTION

The traditional view of software product lines is that a product line is managed entirely within a company. This view is challenged by two recent developments. One is the transition from software product lines to software ecosystems [5]. This transition takes place when a product line company makes its platform available to developers outside the company. These include internal developers (as a in a product line), strategic partners with long-term relationships, undirected external developers, and independent solution providers.

The other transition is one from software ecosystems to collectives. Many such collectives have recently been created, often going by different names such as ecosystems. Examples are the open source Eclipse project [28], or the closed source Artop ecoystem [19]. A collective is set up when a group of organizations wants to achieve a goal they cannot achieve on their own. For example, as a collective, a group of startups can deliver a complete solution (a "whole" product) to a customer. A collective can also collaborate to address common needs of its members, allowing the members to focus on features of their products that differentiate them.

It is often observed that somewhere between 50% and 90% of development effort is spent on creating software that does not differentiate a company from its competitors [30]. Only the remainder differentiates a company from its competitors. This observation has motivated companies to acquire the non-differentiating parts of their software stack elsewhere, for example, as COTS (common off-the-shelf software) or open source software (OSS). When such software is not available, or where a higher degree of control over the software is desired to be able to tailor it more effectively, organizations have joined efforts to create their own common stack in a collaborative effort, making the result available to each other, or even to anyone else who wishes to use it.

The objective of this paper is to study the economics of collectives. Specifically, we seek to identify the factors that affect the economics of collectives, and to create a model linking those factors to economic outcomes.

The paper makes several contributions to the areas identified in the call for this workshop. Its subject, collectives, consists of companies who seek to optimize their development efforts in order to work on features that create the most value for themselves and their customers. By developing propositions from three case studies of collectives the paper seeks to understand how the composition of a collective affects the achievement of the business goals of their members. Specifically, the propositions link three characteristics of collectives (level of contribution, number of members, and diversity of use) to three variables used in [2, 9, 23] to model the economics of product lines (time, quality, and cost).

The paper is organized as follows. Section 2 reviews the literature on the evolution of software product lines to software ecosystems and collectives, and on the economics of software product lines. Section 3 describes the research method used in this paper and introduces the case studies. Section 4 identifies factors that affect the economics of collectives, and develops a model that links factors and economic outcomes. Section 5 discusses the findings and concludes the paper.

2. LITERATURE REVIEW

2.1 From software product lines to software ecosystems and collectives

As companies develop multiple products in the same domain, they benefit from organizing their software development activities as a product line. A product line provides a platform (also known as core asset base) shared by a set of related products that are developed by an organization [4]. The shared platform identifies points of commonality and variation. Products are created "on top" of the platform by reusing its core assets. The motivation for a product line is to reduce the cost of developing new products, while increasing their quality and reducing the time to market. By taking a product line approach a company can manage product diversity and reuse more systematically [30].

A current trend is the transition from software product lines to software ecosystems [5, 17]. In a software ecosystem the company owning the platform opens the platform to external developers, for example, through APIs, services, or plug-ins. As a result, the overall product is no longer assembled by the company that provides the platform, but by customers instead.[1] The shift to software ecosystems is a shift in ownership of the composition of the final product. But moving towards a software ecosystem also allows customers to participate in the creation of products that better satisfy their needs. Thus, externalizing the platform provides the platform owner with new growth opportunities. However, there are also costs to opening the platform, including the cost to create external interfaces, constraints on the further evolution of those interfaces, and the challenges that come with managing the relationships with external developers.

Creating a software product line, and even more so a software ecosystem, however, increases the effort required for software development by as much as 1.5-3 times [8]. Traditionally, only large companies were able to justify this kind of up-front investment. The option to build reusable assets is not available to small companies, given the time and resource pressures under which they operate, and their focus on managing cash flow. The solution lies in taking the concept of software ecosystems one step further, to a collective. In a collective, development is no longer carried out by individual organizations, but by a group of organizations. Members of a collective share the cost and risk of developing a shared code base, but also reap the benefits that come from being able to reuse code and the increased quality that results from collaborative development [38]. Pooling resources allows companies to focus on the added value that differentiates them [37]. A shared platform also attracts further companies that might base their products on it, growing the total market available to the platform [24]. There are several successful implementations of this model, such as the open source Eclipse project, or the closed source Artop ecoystem, both of which are described in detail in Section 3.

A collective can achieve things that its individual members cannot achieve on their own. For example, as a collective, a group of startups can deliver a complete solution to a customer, whereas individually they are only able to deliver pieces of the solution, which the customer has to integrate. Joining forces makes the group of startups much more competitive against large system integrators. Collectives can also collaborate to address common needs, allowing their members to focus on features of their products that differentiate them. The more members a collective has, the more its members are able to share the load of meeting common needs. However, such collaboration is also fraught with problems, for example, the coordination overhead that results from dependencies between subtasks.

A key characteristic of collectives is that they are voluntary organizations. Membership in a collective is a function of how well the collective helps its members meet their business goals. As contributors to the collective, members gain access to the total value generated by the collective. As long as the total value is higher than the cost of contribution, previous research [1] has shown that members benefit from joining. Conversely, existing members of a collective are not interested in members who do not add value to the collective. Thus, collectives often impose conditions on membership such as asking members to commit resources.

The link between software product lines and collectives has been previously explored by studies that ask under which conditions open source projects can be considered product lines [7, 31]. The first study [7] examines how the Eclipse project implements SEI's software product line practice areas [8, 27]. Eclipse is found to have many of the characteristics of a product line. Eclipse products are developed by using and creating plug-ins, which can be end-user products or core assets. The plug-in development guide serves as a production plan for an Eclipse product. Eclipse also has an explicit organization to manage the product planning process, including architecture, planning and requirements councils, as well a management organization, the Eclipse Foundation. The second study [31] notes that the examined open source projects (Eclipse, Mozilla, and Linux) follow best product development practices (such as release management and technical roadmaps) that are most suitable for a distributed development style. Thus, while clearly not all open source projects are product lines, certain open source projects share characteristics with software product lines, even though they employ different processes.

Figure 1 shows the transitions between the three models. The transitions occur along two dimensions. The first transition, from software product lines to software ecosystems, is one from an internal to an external activity, as the platform is made available to external developers. The second transition, from a software ecosystem to a collective, is one from a hierarchical to a network type of governance as described in [22]. The locus of creation and evolution of the platform shifts from a single platform owner to a network. The network collectively creates and owns the platform.

2.2 Economics of software product lines

Key motivations for creating a product line are: reducing cost, enhancing quality, and reducing time to market. All economic models of product lines include at least the reduc-

[1]Customers are either knowledgeable end-customers or intermediaries who provide services to end-customers.

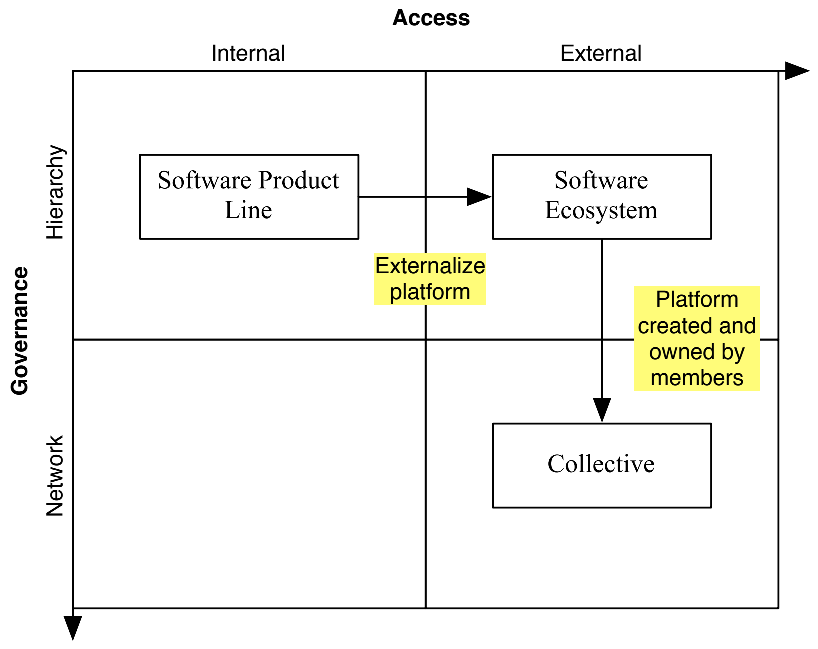

Figure 1: Transitions from software product lines to software ecosystems and collectives

tion of cost. When assets from the platfom are used, a cost reduction is achieved for each product. However, in order to be able to reuse assets, an up-front investment must be made to create the reusable assets. In most software products, a break-even point between the up-front investment and the savings from reuse is achieved around 2-3 products [23].

A cost model for product lines has been proposed in [2, 9]. It has the following cost components:

- C_{org}: cost for an organization to adopt a product line approach (creation of processes, training, etc.)
- C_{cab}: cost to build the platform (core asset base)
- C_{unique}: cost to build product-specific parts (usually a small portion of the complete product)
- C_{reuse}: cost to reuse common parts (including finding, tailoring, and testing the part in the new context)

The cost of creating a product line that consists of n products (p_i) can then be estimated as:

$$C = C_{org} + C_{cab} + \sum_{i=1}^{n} (C_{unique}(p_i) + C_{reuse}(p_i))$$

For a more detailed description of the model see the Structured Intuitive Model for Product Line Economics (SIMPLE) model [9], which has been integrated into the SEI Framework for Software Product Line Practice [27]. The SIMPLE model only incorporates cost factors, neither quality nor time. A research agenda for software product line economics that considers other factors than cost (such as quality, time to market, and risk) is provided in [26].

3. RESEARCH METHOD

This research identifies factors affecting the economics of collectives and creates a model linking those factors to economic outcomes. It combines action research with an analysis of case studies from the literature. In action research, the research is done by or in collaboration with practitioners [16]. I have been actively involved in a collective to create a shared platform for collaboration systems (TFN 200). I have developed first-hand insights into the Eclipse project based on meetings with members of the Eclipse Foundation and members of the Eclipse ecosystem. The description of the Artop ecosystem was taken from the literature [19].

The unit of analysis is a collective as defined in the previous section. The collectives described in this section of the paper form a convenience sample. I had direct access to two of the collectives (Eclipse and TFN 200), and identified a third collective (Artop) from the literature, which complemented the other cases. A summary of each case was prepared that described its purpose, governance structure, and software architecture. Factors and economic outcomes were identified in an iterative manner. Propositions linking factors to economic outcomes were developed through a combination of observations about the case studies, findings in the literature, and logical deduction.

3.1 Eclipse

Eclipse is an open source community focused on building an open software development platform [28]. The Eclipse project was founded in 2001 as a spin-out of technology that IBM had acquired from Object Technology International.

We use the term "spin-out" to refer to a case where a company externalizes an internal development project [34]. Initially, the Eclipse community was primarily driven by IBM and other software vendors. With the creation in 2004 of an independent, non-profit governance body, the Eclipse Foundation, IBM relinquished its control over the project and allowed other players, including IBM's competitors, to become equal members of the community [29].

The Eclipse Foundation is responsible for the technical infrastructure, coordination of the development process, handling of IP rights, and promotion of Eclipse and its wider ecosystem. The role of the Eclipse Foundation is administrative; it does not set the direction of the project or develop code. The direction of the project is set by strategic members of the collective. To become a strategic member, a company has to pay a membership fee and commit resources to the development of the platform. The Eclipse project is organized as a set of top-level projects with subprojects.

Eclipse has a well-defined process for how members can engage with the collective [12]. Three councils, responsible for requirements, planning and architecture, guide the projects. The requirements council collects, reviews, and prioritizes incoming requirements. The planning council manages the release train. The architecture council defines and evolves the architecture of the Eclipse platform. Individual projects are overseen by project management committees. The councils are comprised of strategic members and representatives of the project management committees.

Eclipse is designed to be highly extensible. At its core is a minimal runtime that provides tools for extension management. All functionality of Eclipse (even "core" functionality such as basic UI elements) is implemented in the form of plug-ins. Plug-ins are the basic distribution unit of functionality in Eclipse. A plug-in can declare extension points (aka variation points), and implements extensions (aka variants) to the extension points of other plug-ins.

3.2 Artop ecosystem

The AUTOSAR Tool Platform (Artop) is the platform for an ecosystem in the area of automotive tool development [19]. The project was created in 2008 with the goal to promote and support the AUTOSAR software architecture for complex E/E (Electrical/Electronic) systems. Artop is not an open source project, and members of the collective have to agree to the terms of the Artop license before using the software. Recently, portions of Artop were relased as the open source project Sphinx as part of Eclipse. Internally, Artop relies on the Eclipse Modeling project for modeling, model transformation, and code generation.

Artop is managed by the Artop User Group. This group has members from 120 different companies. The direction of the project is controlled by a subset of design members, who also contribute the majority of the core assets to the platform. Developers can also join as consulting members. In this role, they are allowed to contribute to the core, as well as to create their own subprojects. A final group of members are adopters, who do not contribute directly to the platform, but use it to develop tools.

Artop is based on several principles [19]: low entry barriers to make it easy for members to join; commercial-friendly licensing: the Artop Software License allows other to include the software in commercial products, but also requires contributions to Artop itself to be released back to the platform; technical focus: Artop focuses on platform functionality and leaves end-user functionality to its members; and awareness for competitive differentiators: Artop provides the common parts for an AUTOSAR tool, so tool vendors can focus on developing differentiating end-user features.

3.3 TFN 200

The TFN 200 is a next generation collaboration system released under an open source license. It is designed by a collective that comprises academics and students, several startups, an economic development agency, and early customers. Its goal is to provide a shared platform that can be extended by members to develop their own products in a shorter time frame and to deliver a more comprehensive solution than they could provide on their own. Achieving alignment with the business goals of the members was an important design goal of the TFN 200. The collective was launched by Carleton University in March 2011.

The governance structure of the TFN 200 involves three councils responsibile for architecture, opportunity development, and infrastructure. The architecture council guides the development and evolution of the platform. The opportunity council identifies opportunities – products or services that can be derived from the platform – and extracts and priorities requirements for the shared platform from the opportunities. Opportunities need to be backed up by commitments; they only get to move forward, if there are members of the collective willing to fund or assign resources to them. The infrastructure council provides a shared testing and development platform that is available to all members.

The platform has a plug-in architecture to facilitate its extension by third-party features. As in the Eclipse platform, every component of the TFN 200, except for a small runtime, is realized as a plug-in, maximizing the level of modularity for evolving the platform to meet the common needs of the collective. To encourage the development of commercial products on top of the TFN 200 platform, while ensuring that changes to it are made available to all members of the collective under the same terms as the platform, an the platform is licensed under the Eclipse Public License.

4. ECONOMICS OF COLLECTIVES

This section examines factors that affect the economics of collectives. From the analysis of the cases, three factors were identified as characteristics of collectives: level of contribution, number of members, and diversity of use. Level of contribution is amount of work contributed by a member of the collective to the core asset base. Contributions are not limited to code, but can include requirements, designs, test cases, or feedback. The number of members is the size of the collective. Diversity of use is a measure of the range and variety of contexts of use for the platform.

Figure 2 shows a model that links these factors to economic outcomes. Traditional cost-benefit models of product lines [2] only model the impact on cost, not other benefits from creating a product line such as time to market or quality. The three economic outcomes selected are: time, quality, and cost. Time is either time to market or the coordination overhead. Quality refers to the quality of the core asset base or the quality of the product. Cost is either the cost for the organization of the collective, the cost to create the core asset base, the cost to reuse assets, or the cost to create a unique asset not based on the platform. In the following, I will discuss nine propositions anchored around this model.

4.1 Level of contribution

The level of contribution is not evenly distributed among members of a collective. Instead, as studies of open source projects show, the distribution typically follows a power law, where a small number of members account for a majority of the contributions [10]. Some members may be in a better position to create a specific core asset, because the skills required are not generally available, or they may have a more urgent need than other members for a specific asset to be available in the asset base. When the development of core assets is driven by "lead users" [33], those assets will be available earlier than other planned assets. In the TFN 200 project, the development of a voice component requires specialized expertise. The members of the collective driving its development also have a particular interest in streaming voice to many simultaneous users. A wider project scope than this initial focus will be explored at a later stage.

Proposition 4.1. (Early bird) *Time to market decreases with the level of contribution as a result of better alignment between contributed assets and the contributor's needs.*

In the literature on small groups, trust has been been noted as a determinant of effective team collaboration [10]. Successful leaders make a strong contribution and hold a central position in the community [13]. Projects run by leaders who have demonstrated their technical skills and who have a record of past successes are generally more likely to succeed [20]. Trust can be increased by developing key functionality early in a project to demonstrate that the project is doable and has merit [36]. With the initial release of the Eclipse source code in 2001, IBM triggered contributions from other companies.[2] The creation of an early prototype of the TFN 200 by a small core team galvanized the collective and prompted signficant follow-up contributions from other members. The higher the trust among the members of a collective, the less they require formal coordination.

Proposition 4.2. (Track record) *Coordination overhead decreases with the level of contribution as a result of the increase in trust it creates between the members.*

Through their level of contribution a member can ensure fit of the core assets with their business goals. Members who contribute most to a specific asset can be expected to benefit when reusing the asset. A study of open source development [32] found that contributors obtain private benefits from the development of shared assets that are not available to "free riders", who only use the assets. These include learning, sense of ownership and control, and feedback from others on the code that was revealed. Contributors are also in a better position to tailor their code to their individual needs, as the code that was revealed for general use may not be a good fit with somebody else's needs. In the TFN 200 platform, features to be added are selected from opportunities identified by members as creating business value.

Proposition 4.3. (Fit for use) *The cost to reuse assets in the core asset base and the cost to develop unique, product-specific assets both decrease with the level of contribution.*

4.2 Number of members

When members of a collective contribute to a core asset base, they develop a shared platform. The purpose of the shared platform is to provide non-differentiating functionality to members of the collective, so that each member can focus on its differentiating features [19]. For example, the Artop platform provides functionality needed by every AUTOSAR tool such as an implementation of the AUTOSAR meta-model, serialization, and editing capabilities [25]. A decision on whether to include a contribution in the shared platform is made on the basis of how well the contribution is aligned with the goals of the other members of the collective. If a contribution were only to benefit a single member, then it would not be included in the platform.

Proposition 4.4. (80/20 rule[3]) *The time to market decreases with the number of members. Members can focus on the development of value-added features.*

Each member added to a team introduces coordination overhead [6], time not spent productively towards achieving the task of the team. The capacity of team members to interact with one other in meaningful way is also limited [21]. Conversely, a smaller number of collaborators allows members to interact more frequently with each other. This creates stronger ties among the members and increases commitment and identification with the collective and its goals [14]. The effort to coordinate activities can be controlled by restricting access, that is, strategically selecting members for specific interactions [18]. In OSS projects, restricting access to core members reduces the amount of coordination required when members collaborate on a section of the project [35]. The Eclipse project is organized into top-level projects, each of which has multiple subprojects. Only a subset of project members are active in any specific subproject. The plug-in architecture of the TFN 200 makes the system near-modular, that is, plug-in developers rarely have to coordinate their work with other plug-in developers.

Proposition 4.5. (Too many cooks) *Coordination overhead increases with the number of members working on the same section of the core asset base.*

[2]A study of open innovation projects (such as Eclipse) found that outsiders will invest as much as the founding company itself, if the company voluntarily reveals knowledge and continues to commmit resources to the project [29].

[3]States that as much as 80% of the code in a stack may be non-differentiating, leaving only 20% as value-added.

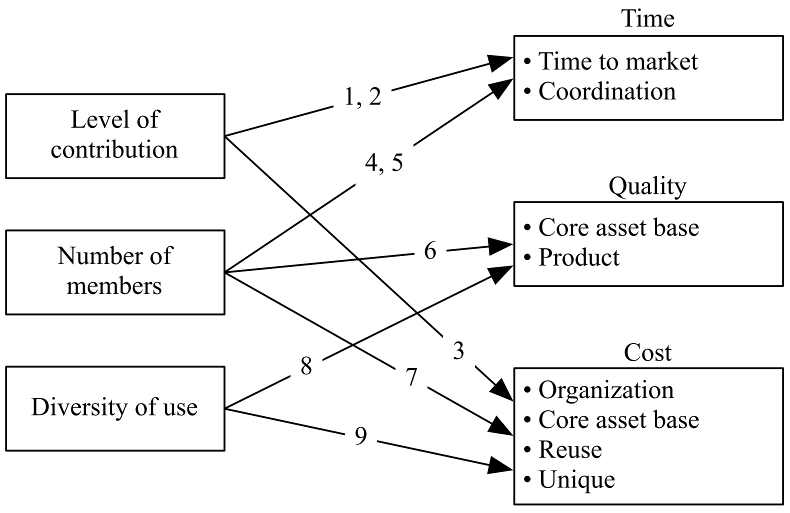

Figure 2: Model of the factors that affect the economics of collectives

A high level of quality in the core asset base attracts new members to the collective. Products built on top of a high quality base will also be of higher quality. In a collective of small companies like the TFN 200, individual members do not have the resources to build a system to the level of quality provided by the platform. From proposition 4.2 it is also apparent that a collective needs to receive enough initial contributions in order to reach an acceptable level of quality that will attract more new members. A study of embedded systems companies using Linux [15] showed that these companies were motivated to reveal their changes to Linux to receive technical support from other companies. The Artop license asks developers to contribute modifications to the Artop platform itself back to the community [19].

Proposition 4.6. (Many eyes) *The quality of the core asset base increases with the number of members who provide feedback on the assets in the core asset base.*

A collective-driven approach to developing a core asset base is more efficient than for each member of the collective to develop a full software stack in isolation [19]. Instead of creating their own versions of commodity features, members can focus on developing features that differentiate them from each other. The effort for maintaining the software stack as it evolves is also significantly reduced. Changes in underlying technologies can be spread among members. If members have existing investments in their own software stacks, switching to a platform developed by a collective may be expensive at first, but will pay off in the long term [19]. In the Artop case, tool vendors differentiate themselves through the value they offer to end users. Common tool components such as integration with other backend systems for subsystem design or simulation are provided by Artop [19].

Proposition 4.7. (Sharing the load) *The cost of contributing to the core asset base decreases with the number of members who provide resources.*

4.3 Diversity of use

Each time the core asset base is put to use in a new context, new aspects of the base will be exercised. Each new context of use may uncover errors or omissions that had not been identified before. This increases the chance of correcting errors, thus increasing the quality of all products that depend on the asset base [23]. For example, the Artop platform provided core assets that allowed individual members to deliver a better integration with other systems, functionality that was either missing or incomplete from their solutions before. Diversity of use also helps evolve the variability of the asset base, as new contexts of use may require variation points that had previously not been exposed. It has been argued that a single company can explore more design options than a collective and thus produce code of higher quality [1]. However, this argument hinges on the assumption that the collective does not have customers, which is not true in the general case, as in the examples studied in this paper.

Proposition 4.8. (Many faces) *The quality of the core asset base increases with diversity of use. Each new context of use will further harden the asset base.*

Diversity of use is driven by the diversity of needs of the members of the collective. The literature on team diversity predicts that increasing knowledge diversity in a team positively affects the range of information accessible to the team, but also negatively affects how a team can integrate information [11]. At early stages of growth, the availability of multiple perspectives that come with diversity of use benefits a collective. Decisions about what functionality to include in the core asset base will be made from a broad understanding of product needs. At later stages, too much diversity may, in fact, hinder the evolution of the core asset base in a cohesive manner. When initially released, the Eclipse project provided core components for a Java-centric development environment. It subsequently grew in diversity to include components for tool integration, modeling, and web applications that could be applied across a range of domains. Today, Eclipse can perhaps be best characterized as a collection of vertical solutions for specific domains. About one half of the Eclipse projects today are technology specific. The diversity of Eclipse projects has increased significantly, and as a group the projects are far less cohesive now.

PROPOSITION 4.9. (Multiple perspectives) *The cost of creating the core asset base first decreases, then increases with diversity of use. At low diversity of use, the collective benefits from a broader range of perspectives. When diversity of use is high, the collective will appear less cohesive.*

5. CONCLUSION

This paper builds on the literature on software product line economics to identify factors affecting the economics of collectives, and develops a model linking those factors to economic outcomes. The paper identifies two factors that could not be observed in either software product lines or ecosystems: level of contribution, and number of members. It shows how these factors suggest new ways to reduce time to market, enhance quality, and lower cost by using a collective. The third factor (diversity of use) is also present in the other models of organizing product lines, but has a more pronounced effect in the context of collectives. In a collective, the members are peers and offer a wider range of perspectives on the product domain that may not be available to a platform owner in a traditional product line approach.

The propositions developed can inform the construction of a quantitive model of the economics of collectives. There are two paths to operationalize the model in this paper. One is to refine the model described in [2] to incorporate the new factors and relationships predicted by the propositions in this paper. The other is to collect data for a sample of collectives and to estimate the parameters for a linear regression model based on the propositions. For instance, the Eclipse project makes statistics on the number of companies contributing to its subprojects and their level of contribution publicly available.[4] The diversity of use of Eclipse subprojects can be estimated from product announcements and listings in the Eclipse marketplace. Another opportunity for future work is to describe each of the propositions in the form of a pattern or best practice. The short names given to the propositions (for example, Early bird, or Many eyes) are meant to suggest possible names for these practices.

6. REFERENCES

[1] C. Baldwin, and L. Clark. Architecture of participation: does code architecture mitigate free riding in the open source development model?. *Management Science*, 52(7), 1116-1127, 2006.

[2] G. Böckle, P. Clements, J. McGregor, D. Muthig, and K. Schmid. Calculating ROI for software product lines. *IEEE Software*, 21(3), 23-31, 2004.

[3] S. Brown, and K. Eisenhardt. Product development: Past research, present findings and future directions. *Academy of Management Review*, 20(2), 343-378, 1995.

[4]Each of the Eclipse subprojects can be considered a collective in the sense described in this paper.

[4] J. Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach.* Addison-Wesley, 2000.

[5] J. Bosch. From software product lines to software ecosystems. *International Software Product Line Conference*, 111-119, 2009.

[6] F. Brooks. *The Mythical Man Month.* Addison Wesley, 1972.

[7] G. Chastek, J. McGregor, and L. Northrop. Observations from viewing Eclipse as a product line. *Workshop on Open Source Software and Product Lines*, colocated with the *International Software Product Line Conference*, 2007.

[8] P. Clements, and L. Northrop. *Software Product Lines: Patterns and Practices.* Addison Wesley, 2002.

[9] P. Clements, J. McGregor, and S. Cohen. The structured intuitive model for product line economics (SIMPLE). Technical Report, CMU/SEI-2005-TR-003, 2005.

[10] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open source software development: What we know and what we do not know. *ACM Computing Surveys*, 44(2), 2012.

[11] K. Dahlin, L. Weingart, and P. Hinds. Team diversity and information use. *Academy of Management Journal*, 48(6), 1107-1123, 2005.

[12] Eclipse Foundation. *Eclipse Development Process.* http://www.eclipse.org/projects/dev_process/development_process_2010.php, 2010.

[13] L. Fleming, and D. Waguespack. Penguins, camels, and other birds of a feather: Brokerage, boundary spanning, and leadership in open innovation communities. 2005-04, opensource.mit.edu, 2005.

[14] M. Granovetter. The strength of weak ties: A network theory revisisted. *Sociological Theory*, 1, 201-233, 1983.

[15] J. Henkel. Selective revealing in open innovation processes: The case of embedded Linux. *Research Policy*, 35, 953-969, 2006.

[16] K. Herr, and G. Anderson. *The Action Research Dissertation.* Sage, 2005.

[17] S. Jansen, A. Finkelstein, and S. Brinkkemper, A sense of community: A research agenda for software ecosystems. *Companion of the 31st International Conference on Software Engineering*, 187-190, 2009.

[18] C. Jones, W. Hesterly, and S. Borgatti. A general theory of network governance: Exchange conditions and social mechanisms. *Academy of Management Review*, 22(4), 911-945, 1997.

[19] C. Knüchel, M. Rudorfer, S. Voget, S. Eberle, R. Sezestre, and A. Loyer. Artop – an ecosystem approach for collaborative AUTOSAR tool development. *International Congress on Embedded Real Time Software and Systems*, 2010.

[20] K. Luther, L. Caine, K. Ziegler, and A. Bruckman. Why it works (when it works): success factors in online creative collaboration. *International Conference on Supporting Group Work*, ACM, 1-10, 2010.

[21] J. Orton, and K. Weick. Loosely coupled systems: A reconceptualization. *Academy of Management Review*, 15(2), 203-223, 1990.

[22] G. Pisano, and R. Verganti. Which kind of collaboration is right for you? *Harvard Business Review*, 86(12), 78–86, 2008.

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

[24] D. Riehle. The economic case for open source foundations. *Computer*, January, 86-90, 2010.

[25] M. Rudorfer, S. Voget, and S. Eberle. Artop Whitepaper, *Artop User Group*, https://www.artop.org/artop_whitepaper.pdf, 2010.

[26] K. Schmid. An initial model of product line economics. *International Workshop on Product Family Engineering*, LNCS 2290, 38-50, Springer, 2001.

[27] SEI. *Framework for Software Product Line Practice.* Version 5.0, http://www.sei.cmu.edu/productlines/tools/framework, last accessed in May 2011.

[28] D. Smith, and M. Milinkovich. Eclipse: A premier open source community. *Open Source Business Resource*, July, 2007, www.osbr.ca.

[29] S. Spaeth, M. Stuermer, and G. v. Krogh. Enabling knowledge creation through outsiders: towards a push model of open innovation. *International Journal of Technology Management*, 52(3/4), 411-431, 2010.

[30] F. van der Linden. Applying open source softeware principles in product lines. *Upgrade*, X(2), April, 32-40, 2009.

[31] J. van Gurp. OSS product family engineering. *International Workshop on Open Source Software and Product Lines*, colocated with the *International Software Product Line Conference*, 2006.

[32] E. von Hippel, and G. v. Krogh. Open source software and "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2), 209-223, 2003.

[33] E. von Hippel. Horizontal innovation networks – by and for users. *Industrial and Corporate Change*, 16(2), 293-315, 2007.

[34] J. West, and S. Gallagher. Challenges of open innovation: the paradox of firm investment in open-source software. *R&D Management*, 36(3), 319-331, 2006.

[35] M. Wasko, G. Sagers, and M. Dickey. Network governance in open source software development projects. Working Paper, Florida State University, 2010.

[36] M. Weiss. Performance of open source projects. *European Conference on Pattern Languages of Programs*, CEUR, 566, 2009.

[37] M. Weiss. Profiting from open source. *European Conference on Pattern Languages of Programs*, 2010.

[38] M. Weiss. Control and diversity in company-led open source projects. *Open Source Business Resource*, April, 29-32, 2011, http://www.osbr.ca.