COORDINATION IN COLLECTIVE PRODUCT INNOVATION

Jitesh H. Panchal
School of Mechanical and Materials Engineering
Washington State University, Pullman, WA 99163 (USA)
E-mail: panchal@wsu.edu

ABSTRACT

Collective innovation is based on connected, open, and collaborative processes to generate, develop, prioritize, and execute new ideas. While collective innovation is gaining significant attention by organizations, research on fundamental understanding of mechanisms enabling collective innovation is still in its infancy. One of the questions in enabling successful collective product innovation is: "How can activities of a large number of independent participants be coordinated?" Various researchers have studied coordination problems in traditional product realization processes, where the emphasis is on managing the dependencies between activities and resources. However, existing approaches for coordinating product development have limited applicability for collective product innovation because they are based on self-organizing communities as opposed to traditional hierarchies. To address this limitation, there is a need to understand how self-organization based coordination can be achieved in collective product innovation.

In this paper, two key aspects of self-organization based coordination are highlighted: decentralization and evolution. A conceptual framework for understanding self-organization based coordination in collective product innovation is discussed. The framework highlights the dependencies between products, processes, individuals and organizational structures, which are important for coordination in collective product innovation. Various coordination mechanisms are required to manage these dependencies, thereby achieving decentralized, evolutionary coordination. For illustrative purposes, examples of such mechanisms used in open-source software development are discussed. Finally, an agent-based model is presented to quantitatively study the mechanisms for achieving decentralized evolutionary coordination. The conceptual framework and the agent-based model are used to derive insights for designing novel coordination mechanisms.

Keywords: collective innovation, coordination, self-organization, decentralization, evolution, agent-based modeling

1. COLLECTIVE PRODUCT INNOVATION

Slawsby and Rivera [1] define collective innovation as a connected, open, and collaborative process to generate, develop, prioritize, and execute new ideas. Collective innovation is rapidly emerging as an important means of accelerating innovation in organizations. Some examples of collective product innovation projects include Linux [2], and Apache [3], open-source car [4], open prosthetics [5], and socially relevant design projects [6]. It is envisioned that collective innovation has the potential to transform the way products are developed [7]. The key characteristics that distinguish collective innovation from traditional product innovation are: a) the participation of a large number of autonomous individuals across organizational boundaries, b) absence of a central authority, and c) a lack of hierarchical control. These characteristics result in self-organization of participants, activities, and organizational (community) structures, as opposed to hierarchical structures in traditional product development.

Collective innovation is related to many recent concepts in innovation such as collective invention [8], user innovation [9, 10], open innovation [11, 12], community-based innovation [13], and commons-based peer-production [14]. Open innovation refers to the use of inflow and outflow of knowledge across organizational boundaries to accelerate innovation [11, 12]. The in-flow of knowledge can be from another organization or from a community of people. Hence, collective innovation can be used as a means for organizations to achieve open innovation. User innovation refers to the innovation carried out by the customers and end-users in product development processes [9, 10]. User innovation can also be carried out either by a single customer or by a group of users working collectively. Distributed innovation is a broader term that refers to innovation carried out by an independent and decentralized group of people. A prime example of distributed innovation is open-source software development, which is characterized by decentralized problem solving, self-selected participation, self-organizing coordination and collaboration, and free revealing of knowledge [15].
Distributed innovation is a special type of collective innovation. Benkler [14] defines commons-based peer-production and discusses the fundamental social and economic aspects that distinguish it from traditional and market-based innovation.

The paradigm of collective innovation has recently attracted attention in many research areas such as management [16], open-source software development [9, 17, 18], economics [19], organizational science [20], social science [14], and intellectual property management [21, 22]. Open-source software development represents the most well-known type of collective innovation. Significant qualitative and quantitative research has been carried out in understanding the factors behind the success of open-source processes. Other researchers have performed empirical studies on open-source projects to identify the decision-making mechanisms [23], community structures [20, 24-26], and evolution characteristics [27].

From the standpoint of engineering design methodology, one of the fundamental open questions in collective innovation is “How can activities of a large number the independent participants be coordinated?” While coordination of activities has been extensively studied in complex product development, the coordination of large number of independently acting individuals is still an open research issue because existing research efforts have mainly focused on well-structured, hierarchical organizations. Very few efforts in engineering design research have been devoted to understanding self-organization based coordination. This paper is a step towards addressing this research gap by providing a conceptual framework, and a supporting computational model for understanding self-organization based coordination in collective product innovation.

The overview of this paper is as follows. Literature review and background are presented in Section 2. Existing literature on coordination is provided in Section 2.1. Coordination in traditional product development is discussed in Section 2.2. An overview of self-organization and the key characteristics of self-organization-based coordination, namely decentralization and evolution, are discussed in Section 3. These characteristics of self-organization based coordination in collective innovation processes are discussed in Section 4. A model for simulating self-organized aspects of collective innovation processes and insights from the model are presented in Section 5. Closing comments are presented in Section 6.

2. AN OVERVIEW OF COORDINATION

2.1 Existing Literature on Coordination

According to the Merriam-Webster dictionary, coordination refers to “the harmonious functioning of parts for effective results”. Coordination has been studied in a variety of different fields, including economics, organization science, artificial intelligence, business processes, and product development [28]. In each of these fields, coordination has been used to represent specific types of problems. For example, in economics, coordination refers to making mutually consistent decisions. In product development and organization science literature, coordination refers to the management of dependencies between activities and resources. In artificial intelligence literature, coordination is generally used to study the interactions of independent agents.

Klein and Orsborn [29] discuss two types of coordination in economics literature – concatenate coordination and mutual coordination. Coordination problems refer to situations where individuals (or groups) can realize mutual gains by making mutually consistent decisions. A class of games in game theory, called coordination games, is used to model such scenarios and corresponding decision-making strategies [30]. Coordination games have multiple pure strategy Nash equilibria in which players choose the same or corresponding strategies. Coordination games have been used to model situations such as deciding which side of the road to drive on, and independently deciding where to meet a friend in the city [31]. In addition to being used to model social and economic coordination, coordination games have also been used to model scenarios involving the choice of technological standards [32].

Coordination has also been extensively studied in organization science. Alexander [33] presents three basic categories of inter-organizational coordination structures - markets, hierarchies, and solidarity-associations (see Figure 1). Markets and hierarchies are different in terms of the medium of interaction and form of control. In markets, coordination between independent and self-interested entities is through mutual adjustment on the basis of information on supply, demand, and prices. Market-based coordination is the foundation of classical economics. On the other hand, coordination in a hierarchy is based on unified governance with different levels of authority and control. The foundations of solidarity-associations are shared goals and trust. Coordination in solidarity frameworks is not through resource exchange, rather due to mutual obligation among participants. Examples of solidarity-associations include families/tribes, peer groups sharing disciplinary or professional values, and professional organizations. Alexander [33] highlights two key aspects of inter-organizational coordination: a) different coordination structures may be found at different levels, and b) the dynamic nature of coordination whereby coordination structures may shift among markets, hierarchies, and solidarity associations. Different coordination structures may exist at different levels such as macro-, meso- and micro-levels, depending on whether coordination is across organizations, between entities within an organization or between individuals. Even within the same organization, different coordination structures may be present. Further, the coordination structures are not static in nature. Organizations may dynamically evolve from one type of coordination structure to another. Coase [34] discusses the emergence of hierarchical forms of organization within markets using the notion of transaction costs. According to transaction-cost theory [34, 35], hierarchical forms of organization emerge if the transaction costs are too high. If the costs of transactions are low, then markets are the favored coordination structures. Coordination costs are a component of transaction costs,
Coordination mechanisms are important in large projects, where efforts have been carried out to coordinate product development activities. Klein and coauthors [43] utilize coordination mechanisms from Malone and coauthors [37] to manage dependencies in engineering change management processes. Adler [44] presents a taxonomy of project coordination mechanisms in design-manufacturing enterprises. The coordination mechanisms presented by Adler are a) standards, b) schedules and plans, c) mutual adjustment, and d) teams. These coordination mechanisms are adapted from Thompson's simple taxonomy of coordination mechanisms – standardization, plan, and mutual adjustment [45]. Adler [44] further classifies each of the four coordination mechanisms into three types based on the following three product development phases: pre-project phase, product and process design phase, manufacturing phase. Herbslab and Grinter [42] present three types of coordination mechanisms in product development – architecture-based coordination, plan-based coordination, and process-based coordination.

One common approach for achieving coordination in product development projects is the reduction of dependencies through product modularity. Literature on product architecture design [46-48] emphasizes these design principles to reduce the coordination overheads in product development projects. Literature on engineering change management [49] deals with achieving coordination between dependent product development tasks involving modifications in the product information. Coordination of activities and decisions has also been studied in concurrent engineering [50], design for manufacturing [44, 51], and design-for-X (DFX). Inter-disciplinary design teams represent a way of coordinating diverse activities. Approaches based on Design Structure Matrix (DSM) to study the inter-relationships between tasks have been proposed for coordinating activities [52]. Various coordination mechanisms have been implemented in software tools such as product lifecycle management, change management systems, etc.

Existing literature on coordination in product development is based on the hierarchical view of product development processes because of the interdependencies between different aspects of product development and the teams working on those aspects. Coordination mechanisms in product development can range from simple approaches to achieving information exchange to sophisticated software systems for coordinating large projects across continents. For a collocated team, coordination generally takes place through meetings and discussions. Herbslab and Grinter [42] suggest that the simplest coordination mechanisms include ad hoc communications at coffee machines. These mechanisms are carried out through face-to-face discussions in traditional settings and through forums, chats, and wikis in the emerging distributed settings.

Coordination is an important issue in large projects involving distributed multidisciplinary teams. Various efforts have been carried out to coordinate product development activities. Klein and coauthors [43] utilize coordination mechanisms from Malone and coauthors [37] to manage dependencies in engineering change management processes. Adler [44] presents a taxonomy of project coordination mechanisms in design-manufacturing enterprises. The coordination mechanisms presented by Adler are a) standards, b) schedules and plans, c) mutual adjustment, and d) teams. These coordination mechanisms are adapted from Thompson's simple taxonomy of coordination mechanisms – standardization, plan, and mutual adjustment [45]. Adler [44] further classifies each of the four coordination mechanisms into three types based on the following three product development phases: pre-project phase, product and process design phase, manufacturing phase. Herbslab and Grinter [42] present three types of coordination mechanisms in product development – architecture-based coordination, plan-based coordination, and process-based coordination.
development organizations, and highly structured view of processes. In contrast, collective innovation processes are driven to a great extent by self-organization of individuals based on their personal interests. For such processes, self-organization based coordination mechanisms are required. In the following section, an overview of self-organization and the key characteristics of self-organization based coordination are discussed.

3. SELF-ORGANIZATION BASED COORDINATION

3.1. An Overview of Self-Organization

Self-organization is defined as "a set of dynamical mechanisms whereby structures appear at a global level of a system from interactions among its lower-level components [53]". Self organization is "the process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover the rules specifying the interactions among the system’s component are executed using only local information, without reference to the global pattern" [54]. Self-organization has been observed in many physical, chemical, biological, economic, and social systems. Examples in biological domain, phenomena such as spontaneous folding of proteins and flocking behavior of birds and fish [55]. In economics, market economy is viewed as a self-organizing system [56, 57].

The emphasis in the study of self-organizing systems is on the emergence of patterns from the behavior and interactions of lower-level entities. Self-organization relies on four ingredients: a) positive feedback, b) negative feedback, c) amplification of fluctuations, and d) multiple interactions [53]. The behavior of entities may be attributed to physical behavior in the case of physical entities and decisions in the case of human participants. The behaviors of entities are based on local information available to them, which changes as the entities interact with each other. These changes in local information may result in positive or negative feedback; a balance between these two types of feedback results in self-organizing behavior [54]. Examples of positive feedback in self-organizing systems are as follows: in fish schools, individuals are attracted to the presence of other individuals, a night club with higher number of people attracts even more people, and social networking sites with greater number of users attract more users compared to the sites that have a small number of users. Negative feedback is generally due to constraints on available resources. For example, in a fish school, the number of fish is limited by the physical resources available, the number of people in a night club is limited by the amount of space, and the number of users on a social networking site may be limited by the maximum traffic on the website. Negative feedback balances the exponential growth resulting from positive feedback. Negative feedback is important in achieving self-organization because it counterbalances positive feedback and stabilizes the collective pattern.

Based on the type of information exchange, the interactions between the individuals can be defined as direct or indirect. Direct interactions involve direct information exchange between different individuals, which changes their local information, and hence, their decisions. For example, considering traffic as a self-organizing system, the interactions between individual drivers are direct interactions. In the case of indirect interactions, the individual actions affect the environment and modify it. For example, considering open-source software development as a self-organizing system, the interactions between individuals are through the modification of software code. Such indirect interaction of entities with the environment is referred to as stigmergy, which plays an important role in achieving coordination through self-organization mechanisms.

Self-organization differs from other modes of pattern formation such as leaders, blueprints, templates, and recipes, where pattern generation is governed by centralized control, and global knowledge. For example, blueprints involve a representation of relationships of parts in a pattern, a recipe is a set of instructions (i.e., process) through which a pattern can be achieved, templates are guides/molds that specify a particular pattern and steer the pattern formation. In contrast, self-organization is based on decentralized control by individual entities. Self-organization represents a way to produce stable and robust patterns through the process of emergence [54]. In Figure 1, markets and solidarity associations are coordinated through self-organization. To study coordination in collective innovation, the key objective is to determine the conditions under which these stable patterns emerge and the mechanisms for achieving those conditions.

As a summary, self-organization is characterized by two aspects – decentralization and evolution. In the following sections, these two aspects of self-organization are discussed in the context of coordination. In Section 3.2, decentralized coordination is compared with centralized coordination. Evolutionary coordination is compared with single-step coordination in Section 3.3.

3.2. Centralized vs. Decentralized Coordination

In centralized coordination, the overall system goals are decomposed into subsystem goals and corresponding activities by a central authority who possesses system-level (global) knowledge. For example, an orchestra is an example of centralized coordination, where a central conductor who directs the individual musicians’ actions through visible gestures. Other examples of centralized coordination mechanisms include process templates, design blue-prints, and leader-follower decision-making, hierarchical command and control.

On the other hand, in a decentralized system, the activities are driven by individual goals and local knowledge. The overall system level behaviors emerge from the individual actions and their interactions. The primary role of the decentralized coordination mechanisms is to ensure that the system-level behaviors are compatible with the system-level goals. Decentralized coordination is studied in depth in the fields of traffic control, economics, and robotics. In economics, the field of mechanism design is devoted to the design of mechanisms that can be used coordinate independent decision-makers. According to Hurwicz and Stanley, “a
decentralized mechanism is a formal entity intended to represent a system for organizing and coordinating economic activity” [58]. In economics and game theory, mechanism design involves the implementation of good system-wide solutions to problems that involve multiple self-interested agents, each with private information about their preferences [59]. A well-known example of decentralized coordination is the free-market in which the individuals are interested in taking actions which maximize their personal payoffs. Coordination between individual actors is achieved through price-based signaling. An example of a system-level goal is the maximization of social welfare. Policies and laws represent decentralized coordination mechanisms for achieving the system level goal. Emissions-trading [60] is another example of decentralized coordination mechanism to reduce environmental pollution at the lowest possible cost to the society. Similarly, in open-source software development, various decentralized mechanisms are used to achieve the system level goal of developing of a quality product, available freely to everyone, while individuals strive to achieve their personal goals. Detailed examples of decentralized coordination mechanisms in open-source software development are presented in Section 4.2. In a hybrid scenario, coordination is achieved through a combination of centralized and decentralized mechanisms.

3.3. Single-step vs. Evolutionary Coordination

Another classification of coordination is based on the dynamic nature of coordination. The mechanisms can be classified as single-step or evolutionary. In single-step coordination, the objective is to achieve coordination in a single set of decisions/actions by individuals. These mechanisms do not rely on the dynamics of processes. A popular example of single-step coordination is the coordination game referred to as the battle of the sexes [61]. Consider a couple who would like to go out for the evening but cannot communicate before making their decision. The husband would prefer to go to the football game while the wife would prefer to go to the opera. However, both would like to go to the same place, rather than to different ones. An example of single-step coordination in product development is a scenario where different individuals make changes to different aspects of a product. If the changes are made concurrently, there is a possibility of inadvertently undoing another individual’s changes. A simple coordination mechanism to resolve this problem is check-out/check-in mechanism where the information is locked when an individual is making any changes so that the other individual cannot make simultaneous changes.

In many cases, however, coordination is a dynamic process, which evolves over time. In such scenarios, as individuals make decisions based on their own preferences, they affect the environment, thereby affecting the future decisions made by other individuals. Depending on the dynamics of the processes, coordinated behavior may or may not emerge, i.e., the system level goals may or may not be met. The mechanisms that drive the evolutionary process towards coordinated action are referred to as evolutionary coordination mechanisms. Mechanisms that accelerate the shift from uncoordinated action to coordinated action are better than the ones that achieve coordination slowly. An example of evolutionary coordination is choosing which side to drive on. Assume that the personal objective of each individual is to avoid collision. If we assume that there is no person on the road, then the first few people can choose any side of the road. But after a few people choose a side, the incentives for new people to choose that side increase significantly. This results in an evolutionary coordination between people who are acting towards maximizing their own benefits. This type of process is called evolutionary self-organization based coordination. The price-based free-market mechanisms in economics are examples of such coordination mechanisms. Economic mechanism design deals with the design of such evolutionary economic mechanisms to achieve the overall system-level goals [58]. Axelrod [62] discusses the role of evolution in the emergence of cooperation in many problems.

4. COORDINATION IN COLLECTIVE PRODUCT INNOVATION


The key assumption in coordination mechanisms discussed in Section 2.2 is the hierarchical nature of product development processes with centralized coordination. Specifically, it is assumed that

- the processes are well defined in terms of activities and dependencies (both information and material),
- the individuals (or groups) and their competencies are aligned with the activities,
- the organization-level goals are aligned with individual goals,
- the organizational structures are aligned with the processes, and
- the processes are aligned with the product structures (in the case of product development).

Due to the implicit assumptions of alignment of individual goals with organizational goals, organizational structure with processes, processes with product structures, and activities with individual competencies, coordination in such processes is mainly focused on dependencies between activities and resources. However, these assumptions are not valid in collective product innovation due to the decentralized nature of the processes. There are various other dependencies that need to be accounted for in collective innovation (see Section 4.1.1). The processes generally do not have well-defined activities and interdependencies that are assigned to individuals. Instead, the activities are selected by individuals based on their own interests. In that sense, the process emerges based on the activities chosen by individuals. These processes are characterized by independent decision-makers who are driven by their personal interests and goals. The organization-level goals emerge based on the individual goals, instead of deriving individual goals from the organizational goals. In collective product innovation, organizational
structures are not pre-aligned with the processes. In fact, the hierarchical organizational structure rarely exist.

### 4.1.1. Decentralization in Collective Innovation Processes

The dependencies in collective innovation projects are shown in Figure 2. The discussion of this figure must start from the individual because the individuals are at the core of collective innovation. The individuals have their own goals that they are interested in fulfilling, and the competencies that they are interested in utilizing and enhancing. Based on these competencies, the individuals self-select (or define) the activities they would like to participate in. The self-selected activities carried out by many different participants result in interdependencies between activities. These interdependencies influence the processes that evolve from the activities, as opposed to being predefined, as in the case of traditional design processes. The interdependent activities result in the evolution of the product, the rate of evolution being dependent on the extent of interdependence. If the activities are independent, the rate of evolution is high, whereas if the interdependence is high, the rate of evolution is small (or even zero). The product also has an influence on the participants’ goals and preferences. For example, if the product is still in its infancy, the value of using and developing it further, and hence, the individual motivation may be low. On the other hand, if the product has been developed significantly, its utility to individuals is high, and the cost of participation is also low. Hence, individuals may be highly motivated to join such projects. The feedback loop represented by the chain of dependencies (participants → activities → processes → product → participants) significantly influences the evolution in collective innovation processes.

![Figure 2 - Dependencies in collective product innovation](image)

The interdependence between activities affects the communication between individuals, which in turn influences the structure of the communities in collective product innovation. Greater communication between individuals results in stronger ties, thereby resulting in a higher likelihood of collaboration. The architecture of collaborations between individuals defines the community structures. The community evolves based on the activities, and the information flow between activities. The participants determine the rules and policies based on which the community operates. These rules and policies are affected by the individual goals and preferences, and the entire community. These policies and rules in turn affect the evolution of the product structure, which consists of the product modules and the interdependencies between modules. The interdependencies between product modules further affect the interdependencies between activities. This completes the second feedback loop represented by the chain of dependencies (process → community → policies/rules → product → processes).

Due to the evolutionary nature of collective innovation scenarios, and the dependencies shown in Figure 2, there are many different avenues of coordination failures beyond the dependencies between activities and resources. For example, assume a completely open collective innovation scenario, where an individual initiates a project and carries out development activities on the product. However, other people don’t get interested in the product and the project does not proceed further. While this is not a coordination failure in a traditional sense [37], we believe that it is a failure of decentralized coordination between the preferences of different individuals, resulting in a dead project. This has been observed to a common phenomenon in many open-source projects. Consider another scenario where the activities of participants conflict with each other, i.e., the changes made by one participant are canceled/reverted by the changes made by another participant. This is also an example of failure of decentralized coordination resulting from the conflicting activities. Another example is a scenario where participants have conflicting goals, i.e., one person takes the project in one direction while other person takes it in a different direction. This is an example of coordination failure resulting from incompatible goals. Note that these coordination failures arise from the dependencies in collective innovation processes, shown in Figure 2, which do not factor in traditional product development processes. Due to the fundamentally different nature of collective innovation processes, the centralized coordination mechanisms are not effective. Hence, decentralized coordination mechanisms are necessary.

### 4.1.2. Evolution in Collective Innovation Processes

In addition to decentralization, another striking aspect of collective innovation processes is their evolutionary nature. As shown in the feedback loops in Figure 2, as the products evolve, the individual preferences may change. For example, as the product develops, more people may see personal benefits in using the product, and may be motivated to participate in the project. As the preferences change, the actions taken by people also change. The kinds of activities that people are involved in may evolve. These activities may either direct the processes towards coordination or away from coordination. For example, if the initial evolution of the product is such that a significant number of individuals with diverse needs find it useful and easy to extend it to satisfy their personal needs, there will be greater motivation for them to contribute. Since their contributions will be directed by different but complementary needs, individual contributions increase the rate of evolution. As the product evolution increases further, even more people may be interested in contributing their efforts, creating a positive feedback loop.
such a scenario, the achievement of individual goals is coordinated towards the achievement of the community-level goal of developing a product that is useful for a wide range of users for a low cost to the community. (Also note that the individual product changes may be coordinated by single-step coordination mechanisms such as check-out and check-in.)

The positive feedback is balanced by the negative feedback caused due to conflicting goals and product changes. For example, consider a scenario where the initial evolution is in the direction that the individuals who get motivated to use the product but not to contribute to it. In such a scenario, the group of individuals who initiated the development may also get discouraged to contribute any further because of the free-riders, thereby setting up a negative feedback loop resulting in an abandoned product. While this exemplifies the effect of only one feedback loop of evolution of product and its effect on the individual goals (see Figure 2), similar negative feedback may occur in the evolution of communities and rules. The interplay between the positive and negative feedback, mediated by the product information results in self-organization of collective innovation (see Section 3.1). The fundamental challenge is to manage the self-organization through evolutionary coordination mechanisms. Over the years, many coordination mechanisms have been established in open-source software development, which is a special example of collective product innovation. Examples of these coordination mechanisms are presented in the following.

4.2. Examples of Coordination Mechanisms from Open-Source Software Development

Since open-source software development represents a prime example of collective product innovation, and is fairly well known with many successful examples, we discuss the coordination mechanisms implemented in open-source software development processes. The following mechanisms are implemented in open-source software development:

1. **Inbuilt multiple parallel processing**: One of the reasons of coordination failures in product development is that multiple people have different conflicting ideas of possible solutions to a problem. In traditional hierarchical processes, coordination in such scenarios is achieved by limiting the efforts to one solution path. However, such a centralized coordination mechanism is not possible in collective product innovation projects that lack a hierarchical decision-making structure. As opposed to that, coordination between multiple ideas is achieved by letting the participants follow their own solution paths in parallel. The participants adopt the solutions that are better whereas the solutions that are inferior are ignored. Through this process, better solutions (product) emerge through decentralized activities of participants as opposed to being imposed by a central authority. Hence, it represents a decentralized coordination mechanism.

2. **Licensing schemes**: The licensing schemes in open-source processes represent mechanisms to coordinate the efforts of participants in achieving a positive growth of the products. One of the potential coordination failures in open-source projects is that different people may have different goals. Some participants may be interested in solving their own problems and may not have the motivation to contribute back to the community. This significantly reduces the growth of open-source software applications. While such a behavior may show short-term benefits to individuals, it is detrimental for the entire community. Very few individuals will have the motivation to contribute back, resulting in an undeveloped product. This is a problem of coordination of individual goals towards the achievement of community level objectives. Many different open-source licensing schemes have been considered to address this coordination problem. An example of an open-source license is GNU, which requires everyone who uses and extends the software under GNU license to share the updates on the code. Through the license-based mechanism, individual goals are coordinated to achieve the community-level goals of maximize ongoing use, growth, development and distribution of free software. This is an evolutionary process because “it ratchets up the process over time on a commons of raw material growth.” [23]

3. **Technical design (modularization)**: Code modularization is a mechanism that decouples the activities associated with different aspects of the project, reducing the interdependencies between activities, thereby helping in achieving coordination. The mechanism is similar to the modularization and standardization mechanisms to achieve coordination in hierarchical product development also.

4. **Formal governance structures and collective decision-making processes**: As the projects grow in size, some of the open-source communities develop formal organizational structures and adopt formal mechanisms for coordinating the decision-making processes. For example, Apache project has an e-mail based voting system based on minimum quorum consensus rule. Linux has a pyramid structure where the code changes are submitted by individuals and are evaluated by the core development team. Similarly, Linux BSD has an organizational structure consisting of concentric circles and Perl’s organizational structure is represented as a pumpkin-holder system [23]. Note that such formal governance and decision-making structures are different from the traditional hierarchical product development processes. This is highlighted by Weber [23] as follows: “that there is no hierarchy for division of labor – i.e., assigning tasks to particular individuals – lies at the heart of the open-source process. This can be a hierarchy of decision-making, for vetting and incorporating the results of distributed work, yet participation in that decision hierarchy is voluntary and remains voluntary for any individual developer, because it is always possible to exit and fork.”

5. **Code forking**: Code forking is a mechanism to coordinate different possible development paths that can be adopted in an open-source project. Forking involves taking the latest version of code and starting a parallel open-source development effort in a direction different from the existing project. Any participant has a right to fork the
code. Forking helps in exploring new solutions and avoids locking into an inefficient solution.

6. **Maintaining two parallel versions of code**: Open-source projects such as Linux maintain two concurrent versions of the code – one stable version and one experimental release. The stable version is used for production applications, whereas the experimental version is used for testing the latest enhancements and their stability. Without maintaining two parallel versions of the code, the robustness of the code would be low from a production standpoint. Low robustness would imply low acceptance by participants, which is a type of coordination failure.

7. **Coordination through norms**: The open-source communities also coordinate their activities based on norms around ownership (based on who initiate the project), decision-making roles, and technical rationality. Further, conflicting motivations and malicious behaviors are discouraged through mechanisms such as sanctioning (referred to as flaming and shunning).

Some of these mechanisms (e.g., modularization) are focused on achieving coordination through product structure, some (e.g., formal governance structures) on configuring suitable organizational structures, while others (e.g., maintaining two versions of code) are based on configuring the processes. These mechanisms represent some of the observed mechanisms found to be useful in open-source software development projects. In addition to these mechanisms, there are various other ways in which coordination can be achieved in open-source processes. Examples include moderation mechanisms in Slashdot and Wikipedia, as discussed in detail by Benkler [14]. Some of these mechanisms may be valid in general collective innovation projects, while new mechanisms need to be developed depending on other conditions such as the types of products, types of activities required, types of incentives, and most importantly, the potential coordination problems. Hence, there is a need to pursuing systematic design of coordination mechanisms for different collective innovation scenarios.

Further, it is important to understand that not all the coordination mechanisms used in open-source projects, and general collective innovation projects, are based on self-organization. The extent of self-organization is different in different collective innovation initiatives. Existing collective innovation initiatives have many differences in the product structure, the extent of decentralization, and the way in which the efforts are managed. In many scenarios, a hybrid approach of combining self-organization and structured processes is used. Some examples of scenarios involving hybrid coordination are presented in the following section.

### 4.3. Combination of Structured and Self-Organized Coordination

In Figure 2, a completely self-organization based scenario is illustrated. However, each of the dependencies can be coordinated either through structured processes or self-organized processes. The extent of structure or self-organization depends on the collective innovation scenario. In a scenario where the product structure, activities and the participants are all self-organized, the product architecture itself evolves based on the activities carried out by different people. The activities are also not well defined. The participants self-decide which activities they want to perform. This scenario is common during the initial phases of an open-source software development.

Consider another scenario where the product is structured but the activities around it are defined and created by independent participants. The development of a well-developed, modular, open-source product is an example of this scenario. This scenario is generally common during the later phases in open-source product development. For example, Drupal [63] has a well defined structure of the core and independent modules are built around it by different participants who self-define the activities and new modules to satisfy their goals. In such a scenario, the product is structured while the process and community are self-organized.

Finally, consider a scenario where the product is well structured and the corresponding activities, to complete the development, are also well defined. However, the activities can be left open for the participants to select and work on. Participants self-select the tasks that they want to work on based on their interests and abilities. An example of such a scenario is crowdsourcing [64] where the companies open up specific problems to the crowds in the form of competitions. In order to coordinate the efforts, a framework for matching the tasks with people is needed. Companies such as Innocentive [65] provide such a match-making framework. In this scenario, the community is self-organized but the product and the process are well structured.

A project does not necessarily adopt a single collective innovation scenario (structured/self-organized) throughout the development process. It may transition from one to other scenario as the project evolves. As the process evolves, generally, the project transitions from self-organized to more structured. For example, projects such as Linux started out as completely self-organized but as the product developed, significant amount of structure was adopted in the project. This is analogous to the transition from markets to hierarchies as explained by the transaction cost theory [34, 35] in organizational science and economics.

Having discussed the different aspects of self-organization based coordination in collective innovation, and some examples of coordination mechanisms in open-source software development, we proceed to discuss an approach to computational modeling of these processes to study their decentralized emergent behavior.

### 5. **SIMULATING THE SELF-ORGANIZED NATURE OF COLLECTIVE INNOVATION PROCESSES**

The top-down structured aspects of product realization have been modeled in the literature using various approaches such as IDEF [66], PERT and GANTT charts [67], design structure matrices [68-74]. The emphasis in these approaches is on minimizing factors such as cost, time, coupling, rework, and the effect of uncertainties; and maximizing factors such as concurrency, and utilization of available resources. As opposed to these objectives, the rationale for modeling of self-
organized processes is to identify conditions under which efforts of individual participants can be mobilized towards synergistic development of products. The objective in designing these processes is to identify conditions, including individual behaviors and community rules, under which targeted self-organization can be achieved and sustained. From the standpoint of factors affecting self-organization, the objective is to coordinate the positive and negative feedbacks in a manner that the emergent behavior of the overall system leads to sustained positive growth of the product.

One such approach for modeling the conditions under which self-organization in collective innovation processes is agent-based modeling. Agent-based modeling [75, 76] is a computational technique to simulate the individual behaviors of autonomous entities (called agents) and their interactions. The autonomous entities can range from physical objects such as atoms and molecules to biological entities such as plants, birds, and animals; and from autonomous robots to social entities such as people in an organization. Agents have their own behaviors and act based on the limited information available to it about its environment. Agent-based modeling is being increasingly adopted in the simulation of social systems because of its capability to predict the emergent behaviors in systems, and the availability of various commercial and open-source software tools such as Swarm [77], Repast [78], Ascape [79], Mason [80], AnyLogic [81, 82], and Echo [83].

Panchal [84, 85], and Le & Panchal [86] utilize agent-based models for simulating the emergent behavior in mass collaborative product development processes, which is a special type of collective innovation processes where the products, participants, and activities are all self-organized (see Section 4.3). An overview of the model and some results from the model are presented next.

5.1. Overview of the Agent-based Model

The agent-based model of mass-collaborative product development processes consists of two main components – the product model and the participant model. The product is modeled as a directed graph consisting of modules and interdependencies between modules. The modules are associated with two parameters – the growth rate and the percentage completion of the module at any given point in time. The interdependencies are modeled as the directional links in the graph and are used to capture the effect of information flow between modules. Coupled modules are represented as two dependencies. Each of the interdependencies is associated with a parameter called percentage rework, which represents the amount of rework that would need to be carried out in a target module if there is a change in the originating module. A screenshot of the model developed in Netlogo [87, 88] is shown in Figure 3.

Each participant is modeled as a decision-making agent involved in making decisions such as a) whether to contribute to the effort or not, b) which module to contribute to, and c) which participants to collaborate with. The participation decisions are based on factors such as cost (effort) and benefit parameters for each participant, the extent of completion of existing modules, and the number of participants working on individual modules. The participation scenario is modeled using the game of involuntary altruism [89]. The decisions are modeled as the mixed equilibrium strategy of the involuntary altruism game. Based on the mixed equilibrium strategy, each participant determines a probability \( \alpha_i^* \) based on which he/she participates in the product development effort. The probability is given by: \( \alpha_i^* = 1 - (c/v)^{1/N} \) where \( c \) and \( v \) are the cost, the value and the number of participants respectively.

The agent-based model captures the lower dependencies loop in Figure 2. The model is simplified by assuming that all the activities are similar in nature, hence any participant can work on any activity associated with a module. Two main dependencies that may result in coordination failure are: a) dependency of the product evolution on the individual participants' contributions, and the dependencies of the participants' contributions on the product's state. Based on these dependencies, the possible coordination failures are a) participants contributing to interdependent modules, resulting in rework and low product development rates, and b) low product development rates resulting in low benefits for participants and hence, small number of participants contributing to the effort. The first coordination failure is common in traditional product development processes, and is handled by careful planning of the sequence of tasks. However, in the case of collective innovation projects, these are decentralized coordination failures because the decisions to participate are made independently by different participants without the control of a central authority. These are also evolutionary coordination problems because as the product evolves, the individual decisions change, which results in further evolution of the product, creating a dynamic positive feedback loop. The key objectives of coordination mechanisms are to address these problems and to achieve sustained (and accelerated) growth of the product. In the following sections, we discuss how the model helps in identifying appropriate coordination mechanisms and the conditions under which sustained product growth can be achieved.

5.2. Evolution of the Product

Despite the simple nature of the model, it provides various insights about collective product innovation processes [84-86]. For example, it is observed that different kinds of
patterns, such as sequential and concurrent, can emerge even without explicitly planning for them [84]. The emergence of these patterns, for the product model in Figure 3, is shown in Figure 4. These patterns correspond to "attractors" of the dynamical system. Depending on the kind of product evolution pattern, the overall time for product development and the effort is different. This insight is useful in identifying the appropriate types of patterns that minimize the overall development time and effort for different classes of products.

The development efforts carried out prior to opening up the product for collective innovation have a significant effect on product evolution. The implication of this trend on the product development process is that with some initial work within a closed team before using collective innovation processes, significant amount of time can be saved. The model also provides insights into the specific modules that can be developed before opening up the process. This is an important coordination mechanism that addresses the coordination problem of low benefit to participants resulting in low participation rates. Such a mechanism is common in open-source product development projects where one individual (or a small group) develops an initial version of usable code before opening it up for contribution from other participants.

Using the model, Panchal [84] observes that a small number of the contributors (~20% for the specific case shown) are primary contributors and a large number of contributors are secondary contributors, which is analogous to the trends observed in open-source software development [90]. Panchal also notes that there are four distinct phases of participation – a) initial participation where participants (committers) who have personal benefits from contributing to the project start developing the product, b) slow growth phase where the same initial group of participants continues working on the project until significant enough progress is made and other participants see the value in it, c) accelerated growth phase when other participants see the value in the project, they join the effort in large numbers, and d) complete participation when the maximum number of people are involved in the project until the project is completed. The information about the different types of participants can be used to develop appropriate incentive structures to motivate the right group of participants at the right time during the product evolution process.

One of the important aspects is the effect of initial product architecture on product evolution. Modular architectures are better than integrated architectures for collective innovation processes because individuals can contribute to different aspects of the product without affecting the contributions of others. Modifying the initial product architecture is also an example of coordination mechanisms. The effects of the product modularity on the evolution time are quantitatively studied by Le and Panchal [86]. The authors use the Degree of Modularity (DoM) metric developed by Sosa and coauthors [91] to quantify the relationship between modularity and the evolution time. The authors execute the model for the design of a cell phone with different architectures. The example is derived from Holttta et al. [92, 93]. The reduction in product completion times with increasing modularity are shown in Figure 5. The model helps in determining how the initial product architecture can be modified (reduced) to expedite product evolution in the collective innovation process. Based on these results, the best product architecture which reduces the development time and effort, and does not compromise the product performance, can be determined.

Further enhancements in the model would include consideration of different types of activities, different participant competencies and interests, the evolution of the product architecture over time, the integration of decomposition-based processes with the evolutionary processes, etc. These additions to the model are expected to reveal greater insights on self-organization based coordination in such processes. Similarly, other features of collective innovation processes can be explored by including other aspects of the dependencies shown in Figure 2. For example, initial explorations of the interrelationship between the community structure and product development using the agent-based model are presented in the following section.

5.3. Co-evolution of community and product structures

The impact of organizational structure on the product structure is well known. According to Conway [94], "any organization that designs a system (defined more broadly than just information systems) will inevitably produce a design
whose structure is a copy of the organization's communication structure”. At the same time, the communication between different participants is based on the product structure and is driven by the dependencies between subsystems, implying the effect of product structure on organizational structure. Hence, the products and the organizational structures are highly interdependent in nature. This interrelated nature is particularly important in collective innovation due to the evolutionary nature. Since no organizational structure is imposed at the start of the process, the structure evolves as more participants join the product development effort and collaborate with existing participants.

Panchal [85] extends the agent-based model presented in Section 5.1 to model the interdependent evolution of the products and the organizational (community) structures. The participants are modeled as agents who make the following decisions: a) deciding whether to participate in the decision-making process or not, b) deciding which module to contribute to, and c) deciding which module to contribute to. The first decision is modeled using the game of involuntary altruism by considering the participants’ costs and benefits from participation. The second decision is governed by preferential attachment, i.e., participants generally contribute to modules that they have most experience with. A participant determines the module to work on with a probability that is proportional to the amount of contribution that he/she has made to that module. In addition to that, participants are also likely to participate in the development of dependent modules. Hence, the probability of contribution on a module is increased by a factor that is proportional to the strength of dependency between a pair of modules. The third decision is based on the observation in the open-source literature [25] that OSS developer-networks are scale free networks whose degree distribution follows a power law. The individuals decide to collaborate with specific participants with a probability, which is proportional to the target participants’ contribution to that module.

The collaborations between different participants are captured as weighted directed networks (see Figure 6) where the nodes represent participants and the arcs represent collaborations between participants. The weights on the arcs represent the number of times two individuals collaborate with each other. The community structure is analyzed using Social Network Analysis (SNA) metrics, specifically connectedness, hierarchy, efficiency, clustering, and centralization. An example of the community structure obtained from the model is shown in Figure 6. Further details are presented in [85]. The key advantage of the model is that it quantifies the effect of product structure and participants’ decisions on the community structure. It is used to predict the structure of the community and its growth, thereby helping in identifying ways to accelerate community growth. Finally, the model helps in understanding the effect of incentives to the participants on the overall time required for product development, the effort involved, and the distribution of effort among participants. These insights can be used to derive mechanisms for achieving self-organization based coordination in collective innovation processes.

6. CLOSING REMARKS

The study of coordination in collective innovation is important because it is an important enabler for the success of product development based on a large group of independent participants. While existing research on coordination in product development is mainly based on two types of problems - resource allocation and sequencing to minimize the effect of dependencies, the emphasis in collective innovation is on the self-organization of independent participants. In this paper, two key aspects of self-organization based coordination, decentralization and evolution, are discussed and contrasted with coordination in traditional product development. Decentralization refers to the lack of central control over the activities. Evolution refers to the changes in conditions over time, which may drive the system towards coordinated/uncoordinated action. The evolution is governed by various interdependencies between product structures, individual preferences, activities, and community structures. To illustrate these concepts, examples of coordination mechanisms used in open-source software development processes are presented. Scenarios involving combinations of self-organized and structured processes are also discussed.

In order to study the evolutionary nature of coordination in collective innovation processes, a simple agent-based model is presented. The model is used to study the evolution of products, individual preferences, and community structures. The objective in developing such a model is not to exactly replicate a complicated collective innovation scenario, but to understand the dynamics of the processes and to identify the conditions under which coordination emerges. Such conditions can then provide insights for managing real-world collective innovation problems. These conditions can also be used to develop software tools to support collective innovation. The research presented in this paper is a step towards addressing the long-term research goal, which is to establish methodologies for designing evolutionary coordination mechanisms for different collective innovation scenarios.

7. ACKNOWLEDGEMENT

The author gratefully acknowledges the financial support from the National Science Foundation through the CAREER grant # 0954447.
REFERENCES


