Strategy Dynamics in Markets of Software Components

Di Shang *Di.Shang@liu.edu*
Long Island University Brooklyn
New York, United States

Karl R. Lang *Karl.Lang@baruch.cuny.edu*
Baruch College, City University of New York
New York, United States

Roumen Vragov *rumvra@yahoo.com*
The Right Incentive LLC
Cold Spring, United States

Abstract

In this paper we propose a dynamic model of a software market for component reuse. We investigate the market dynamics using experiments with economically motivated human subjects. Our results suggest that the introduction of the software component market reduces production costs and increases vendor profits. The dynamic interactions in the component market helped vendors coordinate better their production decisions and resulted in production cost savings. The component market can thrive on a balance between competition and cooperation of software vendors. These experimental results could be applied with some modifications to the development of software products in general.
Keywords: Component markets, Strategy dynamics, Software reuse, Experimental economics

1. Introduction

An individual software component is a software module, or a web service, that encapsulates a set of related functions. Considered a key enabler of software reuse, component-based software development (CBSD) is often highlighted as a practice that reduces time to market and improves quality, and it has been recognized by software vendors as an effective approach to reduce production costs [2]. In the last decade, a large number of software component models have been developed using different principles and technologies [3]. Recently, Gartner suggested that "analytics vendors have started creating marketplaces for software components, such as analytical algorithms, to bring greater flexibility and choice to end users" [4]. This reflects the trends that web services and cloud services have opened new opportunities for dynamically integrating business processes across corporate boundaries on demand, and component-based, distributed computing architecture are pursued by major software vendors such as IBM, Google and Microsoft.

One important attribute of software components is their substitutability. A component can replace another if they meet the same requirements. Often, vendors might already have a platform on which they can integrate several components to derive products customized to their needs. Thus software vendors need to not only embrace component-based development practices but also to actively participate in software component markets that permit vendors to purchase and reuse available components in their products/services.

While there are significant benefits of using components in the development process, there has been little empirical evidence that directs software vendors towards exploring the appropriate production strategies in software component markets that utilize the economic potential of component reuse. Insights on the benefits of participating in component markets will help software vendors decide on their portfolio of products and components offered. In this research, we develop a simplified software component market for a laboratory experiment that

that supports software component reuse. We specifically aim to investigate *the dynamic interactions and how vendors coordinate their production decisions in the component market*. Our results suggest that the dynamic interactions in the component market helped vendors coordinate better their production decisions and resulted in production cost savings.

The paper is organized as follows: In the next section, we discuss related literature on software component-based development and software reuse. Then we present our theoretical analysis and the design of our economic experiment. The subsequent sections present the results of our experiment, and we conclude with discussing our contributions and future work.

2. Related Literature

An individual software component is a piece of software, a web service, or a module that encapsulates a set of related functions. For example, in web services or service-oriented architectures (SOA), a component is converted by the web service into a service and subsequently inherits further characteristics beyond that of an ordinary component [2], [16]. Component-based software development focuses on building large-scale software systems by integrating existing software components, often referred to as Commercial-Off-The-Shelf (COTS) components [8]. CBSD has been recognized to play various roles in addressing needs of particular applications [9]. While the reduced cost of component-based development comes from the relative ease with which components can be assembled, enabling reuse, such assembly may involve complexities of analysis, design, testing, and maintenance processes, common in conventional software development [3], [6,7]. In addition, prior literature on software product line engineering [10] has established the link between reusability of components in the product platform and the ability to increase product variety. Scholars suggest that software vendors can form B2B alliances with partner organizations to deliver a more customized solution to their customers [11].

Reusability is an important characteristic of a high-quality software component, and software components should be designed and implemented in such a way that many different programs can reuse them in different ways [15]. Creating and using software components has greatly expanded a company's ability to create massive, high quality, modern projects powered by components [1]. Software vendors are becoming increasingly involved in putting together components to build applications that solve specific problems. If a software development organization invests considerable resources into developing a flexible and robust software platform architecture that can accommodate the use of components to derive numerous product variants, time to market may accelerate considerably. Because software components are developed by vendors that are focused on developing components in a specific application domain, and since such components are used by large customer bases and would have been tested in a variety of conditions and environments, they are typically considered as relatively higher quality and more reliable alternatives to in-house developments from scratch. However, significant challenges remain due to supply constraints in currently existing software component markets. Vendors lack information on what component would sell in the market, how to effectively price them, and software buyers have difficulties in comparing and evaluating components that are on offer. In addition, most existing guidelines approach this problem from engineering rather than a business standpoint. While frameworks providing guidelines for development and acquisition of components are available, they do not provide insights on how market dynamics impact component supply and demand [13]. The growth of component integrators, component brokerages, and third-party component certifications in the software component market is a key indicator highlighting the need for research focusing on this area [14].

3. Theoretical Analysis

In a market environment where reusable and substitutable components are available, the basic problem that software vendors face is to find the cheapest way to develop a software product without compromising quality. A software product can be developed with the internal resources

of the vendor from scratch or sourcing components from the marketplace. The timing of the launch of new products is as important as the decision whether a launch should be done in the first place. In a software market with component reuse, the optimal time of launching a monopoly product is the point of time that the vendor has the most resource advantage. Since the quantity of components available in a market increases over time, the later a vendor launches their new product, the more alternatives they can choose to develop a product through the reuse of other products already available in the market. Moreover, ideally the price of a product will eventually approach its marginal cost from its high introductory price, so the later a vendor buys products from the component market, the lower price they will pay, and thus the lower cost their new products will take for the development. Therefore, in the software component market, vendors have to follow the dynamic changes of market demand and available resources to set the best product launch time.

For the time being let us assume that every vendor is a monopoly in the consumer market for the products they create. If the demand conditions in the consumer market do not change, the revenues from the consumer market will be relatively certain. However, vendors are not monopolies in the vendor market for software components. This is because the same component can be developed and used in several products by different vendors, which can then compete to provide this component. In a dynamic setting vendors in the vendor market always face a dilemma. Suppose that a vendor is considering developing *productA*. On the one hand the vendor is pressured to develop and offer *productA* as soon as possible so that they can sell the product in the vendor market before the competing companies have offered similar components to those found in *productA*. On the other hand, the company might have an incentive to wait until it is able to make *productA* at much lower cost by buying some low cost components in the vendor market. In most realistic settings it will take time before market participants discover the combination of components that spans the complete product space. Let us call this *set C*. There is always the possibility that vendors who can produce one of the products in *C*, focus their attention on the consumer market and ignore business opportunities in the vendor market. Vendors might also be afraid that competition in the component market will drive the revenues in that market towards 0. Therefore, vendors who have more accurate information are able to act strategically and quickly in the component market to take advantage of their informational and strategic advantage.

We can create a dynamic model in which one of the vendors (*P1*) can make *productA* from *set C*. At the same time, another component market participant (*P2*) can make *productB*, which shares some components with *productA* but is not a member of the *set C*. We define a multi-period game in which *Nature* moves first to decide which vendor has to make the first decision. One of the vendors moves second and decides whether to wait for low cost components in the component market (*Action W*) or to make the product using their own resources and offer it in the market at price *p1* or *p2*. Then the remaining vendor observes all actions that have happened so far and decides to wait until a better alternative is available (*W*) or to produce a competing product and offer it in the market for a price *p1* or *p2*. The modified extensive form of the game and its phase diagram are shown in Figure 1 where *R* signifies the expected revenue from the component market, and *c1* and *c2* are vendor *P1* and *P2*'s costs.

Given the structure of the game we can see that there are four basic states: price war, no market, vendor *P2* buys from vendor *P1 (Action B)*, vendor *P2* makes *productB* and sells it in the market. The most efficient state in the system is the one where vendor *P2* buys *productA* from vendor *P1* before going into a price war. If *c2-p1 >= R2-c2*, the most efficient state will be the sub-game perfect equilibrium of the game and the only stable state. Both vendors prefer to be in that state at the end of the game. Vendor *P2* will wait for vendor *P1* to build *productA* and will use *A*'s components to build *productB*. Vendor *P2* can also control to an extent the market power of vendor *P1* because, if the price for *productA* is too high, vendor *P2* can make *productB* and offer it on the market as a partial substitute for *productA*. Vendor *P2* can use the "threat" of a price war to keep vendor *P1* in check.

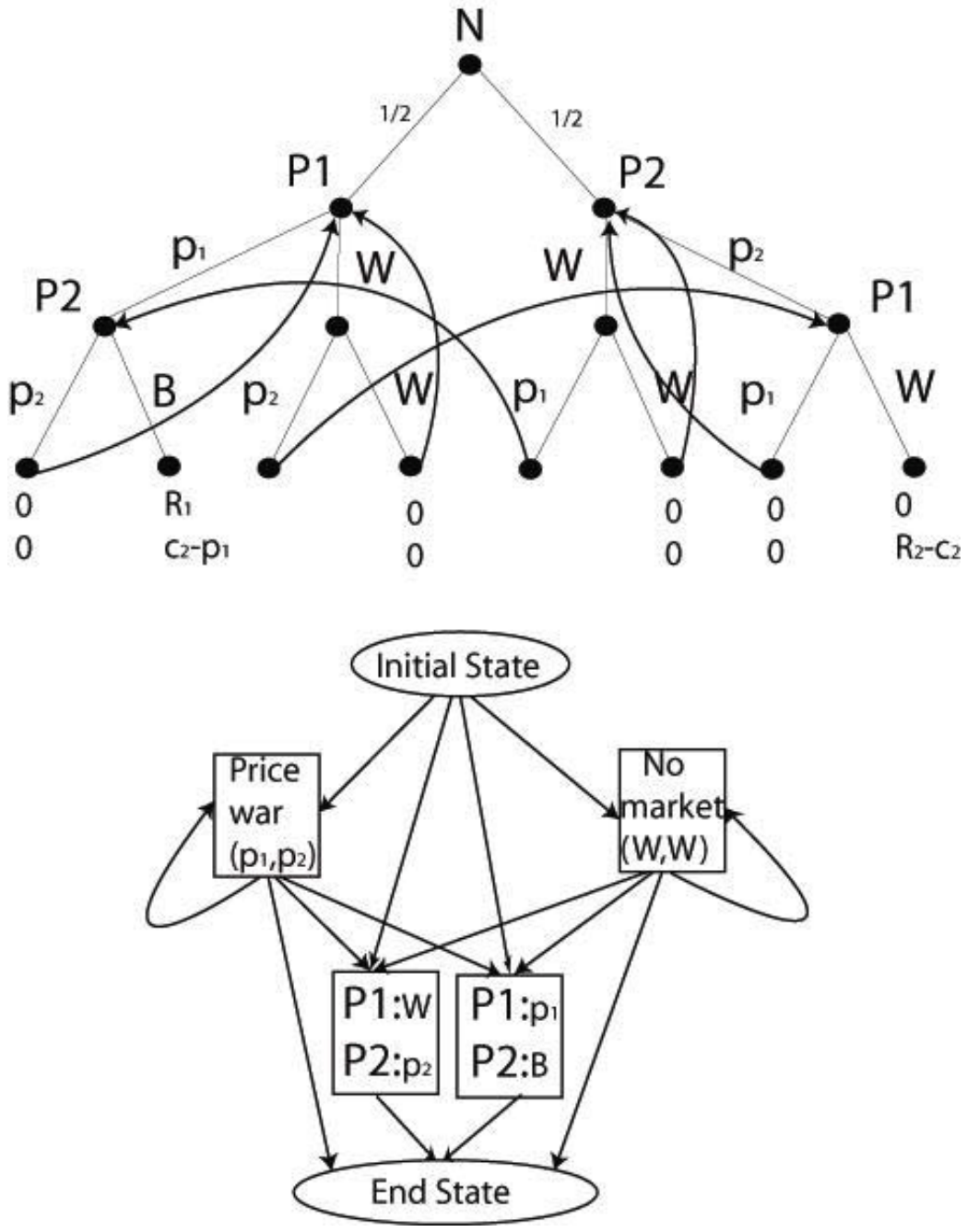

Fig. 1. The modified extensive form and the state diagram of the multi-period game. The arrows in the extensive form shows transitions from one period of the game to another. The payoffs pertain to the nodes only if they are final nodes.

Assuming that *Q* is the monopolistic quantity supplied to the market for the components in *productA*, the above inequality could be rewritten as *c2 >= Qp1/2*. If the number of vendors willing to buy the components in *productA* or *productB* is high, then the inequality above will not hold. Vendor *P2* will have an incentive to produce *productB* as soon as possible without waiting for vendor *P1* to build *productA*. The most efficient state can be maintained only if vendor *P1* can use the threat of a price war to discourage vendor *P2* from building and offering *productB*. The "no market" state is unstable because if one vendor knows for sure that the other one will not participate in the market, he will have an incentive to deviate from this state. The

"price war" state leads to 0 profits from the component market for both vendors because marginal cost is 0.

4. Experiment

The methodology of experimental economics is suitable for our investigation especially because private and unobservable customer and vendor preferences and costs can be induced and are directly available to the researcher in the laboratory [12]. Through laboratory experiments we are also able to relax many of the standard homogeneity, uniformity, and independence assumptions. Laboratory experiments also allow us to gather data from pricings and market designs that have never existed in practice.

Following the experimental economics approach, we present an experimental lab environment that captures the salient features of the software economy while simplifying the design in order to attain experimental control and streamline the cognitive demands on the participants. Our experimental environment represents a simplified, idealized economy of vendors and clients of component-based software applications. There are 4 vendors (*P1 – P4*) in the ecosystem and each of them has the capacity to develop 4 applications (*product1 – product16*). The software applications are component based, and each of them is designed for providing a specific business solution to the business clients (*C1-C10*). Among the applications, *product6, product11*, and *product16* are in *set C*. Each application has a given fixed cost (from a random cost schedule that ranges from 100 to 500) to develop the software and bring it to market. The marginal cost for producing additional units is set to zero. There are 10 business clients in the ecosystem, and each client demands for 5 applications. We model variation of demand for applications in a way that each application has different levels of demand in the marketplace, as suggested by [5]. The vendors are to sell applications for profit to the business clients.

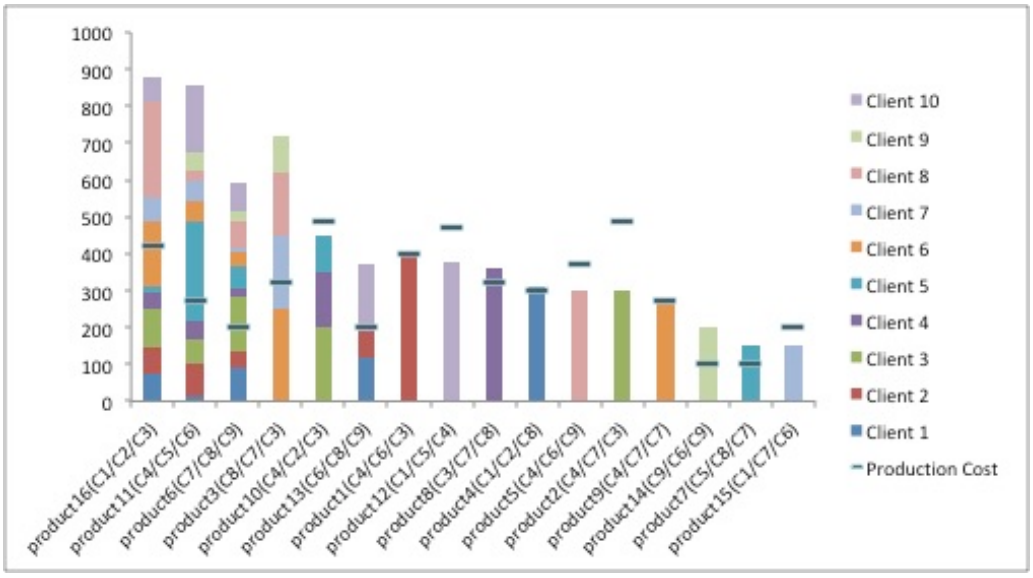

Fig. 2. The valuation of each software application to the clients and the production cost of each software application.

As illustrated in Figure 2, the applications vary in their profit potential, some enable high returns while others may not be profitable at all. The vendor can decide to offer none, 1, 2, 3, or all 4 of the applications to business clients.

Our experimental design comprises of two treatments. In the Baseline treatment, vendors sell complete component-based applications without a component market. In the Component Market Treatment, vendors offer the same software applications but now bundled with a component reuse license (that allows for decomposition of the software and reuse of individual components) in a component market. In this treatment, each vendor can decide to develop a

software application either with the internal resources of the vendor from scratch or by obtaining and reusing components available in the market. When a vendor purchases a software application from the component market, he can create applications on their own development platform based on the components available in the applications that they have purchased from the market. In our experimental design, each of the applications is comprised of 3 software components. Such a simplified design of software components has been adopted in previous research to increase experimental control allowing focus on the specific objective attributes of software components [1]. As shown in Figure 2, for example, the software application *product16* is comprised of the three components *C1*, *C2* and *C3*. Similarly, the software application *product11* comprises the components *C4*, *C5* and *C6*. Now let's assume vendor *P1* wants to develop a software application *product1* which comprises the components *C4*, *C6* and *C3*. Vendor *P1* has the choice to develop the software application with the internal resources from scratch, or he can purchase and reuse components from other software applications if available in the component market. If he purchases software application *product16* and *product11*, he could recombine component *C3* from *product16* with components *C4* and *C6* from *product11* to develop *product1*. For the sake of simplicity, we assume that vendors incur no cost in disassembling and re-assembling the components they have purchased on their own development platform

One hundred and twelve undergraduate students in a business school of a U.S. public university participated in our study. Subjects were randomly assigned to the roles of a vendor or a client and randomly grouped into sixteen cohorts of fourteen, four vendors and ten clients each. Each treatment was repeated four sessions, and each session used a different cohort of participants. After the subjects came to the experimental laboratory, they were seated at separated individual workstations that served as trading terminals for selling or buying products in the market. No communication among subjects was permitted during the experiment, other than posting the bids and asks in the trading periods. In both treatments vendors and clients interactively discover the prices of the software applications through a market mechanism, which is implemented as a version of a continuous double auction. Sellers in the market can submit asks or accept bids, but they are not allowed to increase their own asks. Likewise, buyers can submit bids or accept asks, but they are not allowed to decrease their own bids. Bids and asks at the same price are matched automatically by the trading system. These rules follow the ones used by Smith [12] in his original market experiments. We ran 4 sessions for each treatment and 10 repeated rounds in each session. Totally, we collected observations of 40 rounds of experiments for each treatment.

5. Results and Discussion

Our preliminary results suggest that vendors can reduce production costs and increase profits by participating in the component market, and that vendors who act strategically and adapt quickly in the component market are able to achieve more strategic advantage. Overall, we observed strong participation in the component market and adoption of software components reuse strategies. All the vendors offered some software applications in the component market and also purchased some applications for reuse from other vendors.

As shown in Figure 3 "*buy and reuse*", the most efficient state in our theoretical analysis, accounted for 48% (on average) of production of software applications in the component market treatment. For half of the software applications, the vendors shifted production strategy to buying components from the market for reuse from building them from scratch. As a result, the production costs of software applications were significantly reduced (one-way ANOVA, $p<0.01$). The component market achieved substantial production cost savings of 40% for procuring and reusing components. For the software applications that mainly developed from scratch – *product7, product11* and *product16*, there is no significant ($p> 0.1$) reduction of production costs in the component market treatment.

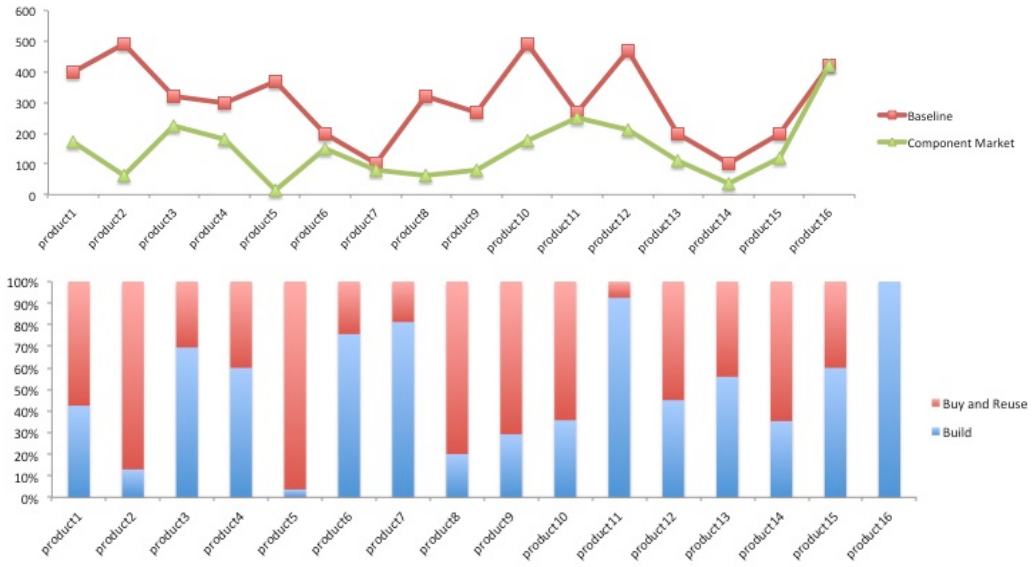

Fig. 3. The average production cost of each software application in the two treatments (upper chart) and the percentage of developed by reuse of each software application in component market treatment (lower chart).

The average profit per software applications in the baseline was -46 (which actually incurred a small loss to vendors), and it increased in the component market treatment to 48 and became profitable. As shown in Figure 4, not all the software applications increased profit for vendors in the component market treatment. The component market exhibited high demand for applications that included high-value components (*set C*), and the market helped vendors to discover which applications to build and which applications to buy from the market.

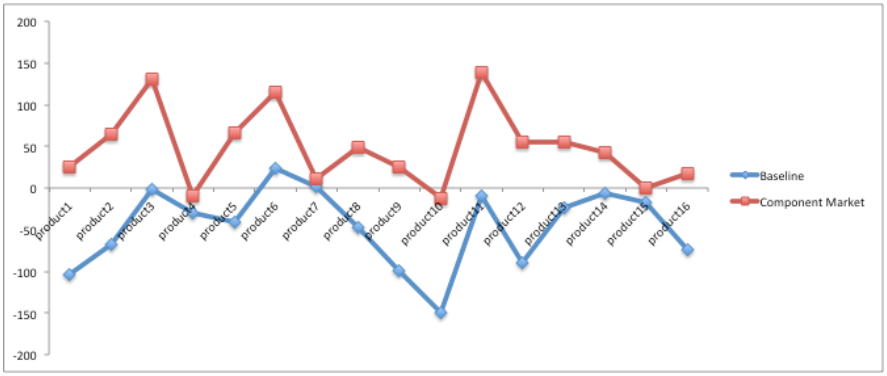

Fig. 4. The average profit of each software application in the two treatments.

The variation of profits of software applications explains the change of vendor profit as shown in Figure 5. In component market treatment vendor *P3* were able to sell *product11* (*set C*) in the component market and reuse software components purchased from other vendors to more efficiently develop the application (*product9*, *product10* and *product12*) needed for their business clients. As a result, they were able to achieve the highest profit in component market treatment comparing to in the baseline they scored the lowest profit. Overall, we see that vendor profit significantly ($p<0.01$) increased through participation in the component market. The dramatic change in profitability can be attributed largely to higher profits of most software applications in the component market treatment.

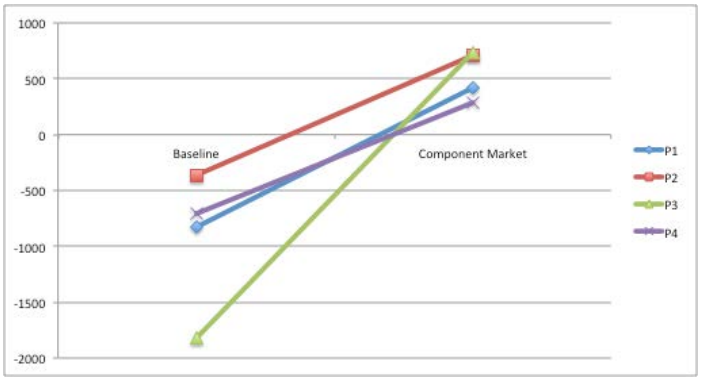

Fig. 5. The average profit per round of vendor P1-P4 in the two treatments.

In the component market, vendors have to follow the dynamic changes of market demand and available resources to set the best product launch time. Examining the product launch time, we see that products in set C (*product6*, *product11*, and *product16*) in general have early launch time than other products as shown in Figure 6. Vendors of these products have the most resource advantages, as they are potentially highly demanded in both the consumer market and the component market.

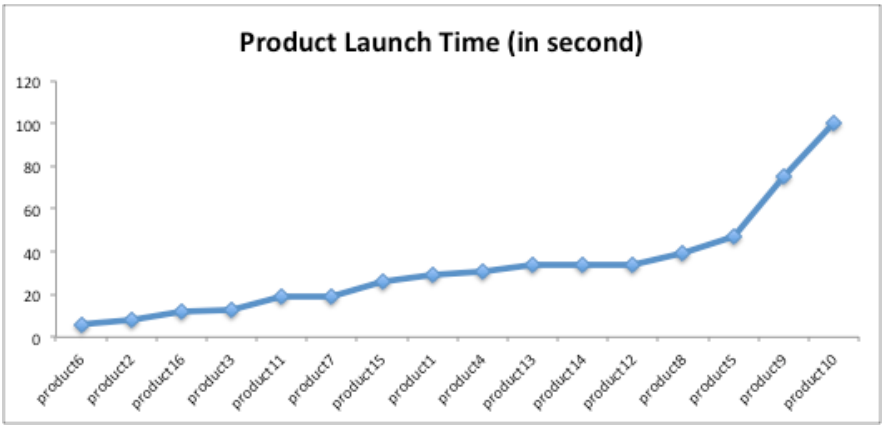

Fig. 6. The average product launch time of each software application in component market.

Results from our analysis demonstrated that introducing a component market for reuse created a complex dynamic system. There was enough activity in the component market to increase efficiency compared to the baseline. The average vendor profit attained in the component market treatment is significantly higher than in the baseline. In addition, there are many production strategies that vendors can take in order to increase their profits with the component market. Vendors who acted strategically and moved quickly in the component market were able to achieve more strategic advantage.

6. Conclusion

This paper proposed a dynamic model of a market for software components. We investigated the market dynamics using experiments with economically motivated human subjects. Our results suggest that the introduction of the component market for software reuse reduces production costs and increases vendor profits. The dynamic interactions between production strategies help vendors coordinate better their production decisions and result in production cost savings. Therefore, component market can thrive on a balance between competition and cooperation.

Several areas in software development, namely, components development and software reuse, can benefit from the findings of this study. Component marketplaces help organizations speed up their development processes and cope with the transformational changes introduced

with digital business. Empirical evidence on the effectiveness of component markets might encourage vendors to transform their current development process to focus more on incorporating reusable components from other software vendors.

Our study is motivated by the dearth of insight into the business side of component markets, while the technical side of component reuse has been well researched. The preliminary results reported here can only serve as an initial step towards a more complete understanding of how strategies in the component markets change in response to changing market dynamics. Thus, our next step is to extract more insights by analysing the data to complete the study. For instance, a deeper look into why vendor *P3* outperformed in the component market will uncover how to achieve strategic advantage in terms of product launch time and pricing strategies. Furthermore, a smarter market might be able to better help market participants discover quickly the members of the C set. Further research is required to investigate the impact of these factors.

References

1. Anguswamy, R., Frakes, W.: A Study of Reusability, Complexity, and Reuse Design Principles. In: Proceedings of ESEM'12, pp. 161-164. Lund Sweden (2012)
2. Bertoa, M.F., Troya, J.: Measuring the Usability of Software Components. Journal of Systems and Software. 79(3), 427-439 (2006)
3. Crnkovic, I., Sentilles, S., Vulgarakis, A., Chaudron, M.: A Classification Framework for Software Component Models. IEEE Transactions on Software Engineering. 37(5), 593-615 (2011)
4. Gartner (2017), "Gartner Says by 2020, At Least 30 Percent of Industrie 4.0 Projects Will Source Their Algorithms From Leading Algorithm Marketplaces", https://www.gartner.com/newsroom/id/3646717. Accessed June 22, 2018
5. Hong, S., Lerch, F.J.: A Laboratory Study of Consumers' Preferences and Purchasing Behavior with Regards to Software Components. The DATA BASE for Advances in Information Systems. 33(3), 23-37 (2002)
6. Kessel, M., Atkinson, C.: Ranking Software Components for Pragmatic Reuse. In: Proceedings of the Sixth International Workshop on Emerging Trends in Software Metrics, pp. 63-66. Florence Italy (2015)
7. Lakshmi N.V., Hendradjaya, B.: Some Theoretical Considerations for a Suite of Metrics for the Integration of Software Components. Information Sciences. 177(3), 844-864 (2007)
8. Mahmood, S., Niazi, M., Hussain, A.: Identifying the Challenges for Managing Component-based Development in Global Software Development: Preliminary Results. In Proceedings of Science and Information Conference (SAI2015), pp. 933-938. London UK (2015)
9. Morch, A.I., Stevens, G.: Component-Based Technologies For End-User Development. Communications of the ACM. 47(9), 59-62 (2004)
10. Pohl, K., Böckle, G.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
11. Sarker, S., Sarker, S., Sahaym, A., Bjørn-Andersen, N.: Exploring Value Cocreation in Relationships between an ERP Vendor and its Partners: a Revelatory Case Study. MIS Quarterly. 36(1), 317-338 (2012)
12. Smith, V.L.: Experimental Economics: Induced Value Theory. American Economic Review. 66(2), 274-279 (1976)
13. Ulkuniemi, P., Seppänen, V.: COTS Component Acquisition in an Emerging Market. IEEE Software. 21(6), 76-82 (2004)
14. Wohlin, C., Wnuk, K., Smite, D., Franke, U., Badampudi, D., Cicchetti, A.: Supporting Strategic Decision-Making for Selection of Software Assets. In: Maglyas, A., Lamprecht, A. (eds.) Software Business. Lecture Notes in Business Information Processing, Springer (2016)

15. Younoussi, S., Roudies, O.: All about Software Reusability: a Systematic Literature Review. Journal of Theoretical & Applied Information Technology. 76(1), 64-75 (2015)
16. Yu, Y., Lu, J., Fernandez-Ramil, J., Yuan, P.: Comparing Web Services with other Software Components. In: Proceedings of IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, USA (2007)