Communications of the Association for Information Systems

Volume 51 Article 36

2022

A Market-Based Approach to Facilitate the Organizational Adoption of Software Component Reuse Strategies

Di Shang
Department of Management, Coggin College of Business University of North Florida, d.shang@unf.edu

Karl Lang
Zicklin School of Business, Baruch College City University of New York

Roumen Vragov
Queensborough Community College City University of New York

Recommended Citation

Shang, D., Lang, K., & Vragov, R. (2022). A Market-Based Approach to Facilitate the Organizational Adoption of Software Component Reuse Strategies. Communications of the Association for Information Systems, 51, pp-pp. Retrieved from https://aisel.aisnet.org/cais/vol51/iss1/36

Communications of the Association for Information Systems

A Market-Based Approach to Facilitate the Organizational Adoption of Software Component Reuse Strategies

Richard Shang
Department of Management, Coggin College of Business
University of North Florida
d.shang@unf.edu

Karl Lang
Zicklin School of Business, Baruch College
City University of New York

Roumen Vragov
Queensborough Community College
City University of New York

Please cite this article as: Shang, Richard; Lang, Karl; Vragov, Roumen: A Market-Based Approach to Facilitate the Organizational Adoption of Software Component Reuse Strategies, *Communications of the Association for Information Systems* (forthcoming), In Press.

Communications of the Association for Information Systems

Research Paper ISSN: 1529-3181

A Market-Based Approach to Facilitate the Organizational Adoption of Software Component Reuse Strategies

Richard Shang
Department of Management, Coggin College of Business
University of North Florida
d.shang@unf.edu

Karl Lang
Zicklin School of Business, Baruch College
City University of New York

Roumen Vragov
Queensborough Community College
City University of New York

Abstract:

Despite the theoretical benefits of software component reuse (and the abundance of component-based software development on the vendor side), the adoption of component reuse strategies at the organizational level (on the client side) remains low in practice. According to research, the main barrier to advancing component-based reuse strategies into a robust industrial process is coordination failures between software producers and their customers, which result in high acquisition costs for customers. We introduce a component reuse licensing model and combine it with a dynamic price discovery mechanism to better coordinate producers' capabilities and customer needs. Using an economic experiment with 28 IT professionals, we investigate the extent to which organizations may be able to leverage component reuse for performance improvements. Our findings suggest that implementing component reuse can assist organizations in addressing the issue of coordination failure with software producers while also lowering acquisition costs. We argue that similar designs can be deployed in practice and deliver benefits to software development in organizations and the software industry.

Keywords: Software Reuse, Software Platform, Software Licensing Model, Pricing Strategy, Software Development.

[Department statements, if appropriate, will be added by the editors. Teaching cases and panel reports will have a statement, which is also added by the editors.]

[Note: this page has no footnotes.]

This manuscript underwent [editorial/peer] review. It was received xx/xx/20xx and was with the authors for XX months for XX revisions. [firstname lastname] served as Associate Editor.] **or** The Associate Editor chose to remain anonymous.]

1 Introduction

Krueger (1992) defined software reuse as the process of creating software systems from existing software rather than building them from scratch. Mørch et al. (2004) explained that component-based software development (CBSD) is a key enabler of software reuse and focuses on building large-scale software systems by integrating existing commercial-off-the-shelf software components. CBSD has long been recognized as an effective method for lowering production costs, shortening time to market, and improving quality (Szyperski, 2002). Many software component models have been developed over the last decade using various principles and technologies (Crnkovic, Sentilles, Vulgarakis, & Chaudron, 2011). Since then, the software development ecosystem has changed considerably. Many systems today are highly distributed and take advantage of publicly available APIs/services. Web services and cloud services have created new opportunities for dynamically integrating business processes across corporate boundaries, and major software vendors such as Amazon AWS, IBM, Google, and Microsoft are pursuing component-based distributed computing architectures.

The recent developments in cloud computing services present an emerging trend of containerization of software. Containerization of software is a modern approach to transform applications into collections of smaller services or components which are portable, scalable, efficient, and easier to manage. A container is defined as "a standard unit of software that packages up code and all its dependencies in order for the application to run quickly and reliably from one computing environment to another."[1] Such independence makes containers more portable, scalable, reusable, and of higher quality. In the past, one challenge in CBSD has been the requirement that developers have to navigate through dependencies to identify components that are relevant to the task and to decide how components should be reused (Szyperski, 2002). To function individually or collectively, software components rely on various libraries, operating systems, and architectures. This dependency raises search costs because clients must consider not only required functions but also the dependencies on which they rely. The dependency also increases customization costs because clients must set up all required dependencies to make the components work in their systems or platforms. By contrast, containerization enables applications to be "written once and run anywhere." This portability is crucial in terms of software development and reusability of software components. The adoption of containerization is growing rapidly in this industry segment. According to Gartner (Pettey, 2019), "by 2022 more than 75% of global organizations will be running containerized applications in production, which is a significant increase from fewer than 30% today."

Despite the economic potential of CBSD, we observe that the software component market has never fully matured. There are only a few software component marketplaces that operate on a small scale. ComponentSource (www.componentsource.com), for example, is the most well-known marketplace we could find. Surprisingly, this fact indicates that Ravichandran and Rothenberger's (2003) prediction from almost 20 years back that "black-box reuse with component markets could be the silver bullet solution that makes software reuse a reality, …" has still not come to fruition. At the same time, the organizational adoption of software development utilizing external software components is also still low in business practice (Clements & Gacek, 2001; Stefi, Lang, & Hess, 2016; Bastos, da Mota Silveira Neto, Oleary, de Almeida, & de Lemos Meira, 2017)[2].

Often, client organizations might already have a development platform on which they can integrate several components to derive products customized to their needs. Customers can, for example, integrate their on-premises applications with cloud services or edge computing devices on the Amazon AWS platform. Software vendors offer preconfigured software solutions in both Amazon Machine Image and software as a service formats, all of which are hosted on AWS. Customers can customize software products and build business solutions on top of the AWS platform. Containerization breaks a complex application into a series of specialized services, and new cloud-based applications can be built from the containerized microservices.

In the software industry, the reuse of software is regulated by an End User License Agreement (EULA). Similarly, in the software container market, industry practice follows the same conservative approach to software reuse, as codified in a typical EULA of software container vendors: Customer will not "…, (h)

[1] Docker.com, last retrieved on January 21, 2022.
[2] We want to point out that we are not talking about adoption of CBSD tools and approaches in general. Most large tech companies and software vendors, Microsoft, Google, SAP, adopt CBSD in-house to componentize their applications/services for costs and scalability. We are only talking about the external component reuse of clients.

modify, copy, or create derivative works based on a service or any part, feature, function or user interface …, (j) disassemble, reverse engineer, or decompile a service, or access it. …"[3] From those EULA terms, we can see that vendors favor strong copyright protection and treat software reuse primarily as a loss of control and their ability that they may not capture sufficient value from their creations.

Understanding the benefits of reuse licensing schemes for tradeable software components, as opposed to integrated monolithic products, should help software vendors devise more effective product development strategies. This necessitates that software vendors not only embrace component-based development practices but also implement component reuse licensing schemes that allow users to reuse and integrate these components into their offerings. Introducing and implementing a flexible licensing model can be a first step toward realizing the potential advantages of software component reuse. Such an approach could foster component reuse and play a key role in making component reuse strategies viable for business clients. In other areas, an increasing number of online platform-based market mechanisms have emerged and are now commonly used by businesses to sell complex products to other businesses, often on a large scale (for example, the Google AdSense/AdWords auction platform for selling digital advertisements, the Alibaba platform that globally connects manufacturers and suppliers with business customers, among many others).

In this paper, our specific research question is: *What is the impact of introducing software component reuse licenses in a given market exchange design on production cost, seller profit, product selection?*

In a controlled experiment in the laboratory, we examine the effect of introducing a specific reuse license that permits clients who purchase software on a platform to reuse components of it in other software developed at the client site. We use a previously tested experimental environment described in Lang, Shang, and Vragov (2015) to implement our software platform design, which models a simplified marketplace of competing software vendors, business clients, and component-based software products. We recruit IT professionals, who serve as proxies for software client organizations, as the participants in our experiment.

Our findings suggest that supply increases when reuse is permitted and that vendors and clients can mitigate the coordination failure problem and achieve higher component utilization and component reuse rates, as well as lower acquisition costs. We argue that similar platform designs with reuse licensing could be easily deployed in the software industry. Efficient platform designs can have significant implications for the software industry and help evolve component reuse into a robust industrial process for software development and fostering software reuse strategies.

This paper is structured as follows. Section 2 describes background and related research to position this work. Section 3 discusses our research methodology and experiment designs. Section 4 presents the study's findings, followed by a discussion of the implications and limitations of the findings in Section 5. Section 6 brings the research to a close.

2 Theoretical Background

2.1 Component-based software development

Considered as a key enabler of software reuse, 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 reducing production costs (Sherif & Menon, 2004; Bertoa, Troya, & Vallecillo, 2006).

Although software components are frequently discussed from a purely technical perspective, Atish and Hemant (2013) considered them to be business components that represent "self-contained piece(s) of software" with specific functionality that can be assembled and integrated to develop complete business solutions. An individual software component is a software module that encapsulates a set of related functions. For example, in web services or service-oriented architecture (SOA), a component is converted by the web service into service and subsequently inherits further characteristics beyond that of an ordinary component (Yu, Lu, Fernandez-Ramil, & Yuan, 2007; Sinha & Jain, 2013).

[3] Extract from end-user license agreements of certain container vendors, e.g., Cloudwave (https://d7umqicpi7263.cloudfront.net/eula/product/94bd3d30-b66e-45d9-b15e-09d1b2846d4d/9387c44d-40f4-4a7c-8a57-a81cca9d0bdf.pdf) and Datadog (https://d7umqicpi7263.cloudfront.net/eula/product/d882c81b-9d58-4e1e-954c-1a6126a36317/bd7b7359-ba96-4c68-ab4b-fcd6031cc1fd.pdf), last retrieved on January 21, 2022.

In the past, one challenge in CBSD is that it requires developers to navigate through dependencies to identify components that are relevant to the task and to decide how components should be reused (Szyperski, 2002). Dependency becomes a hidden cost of reusable architectures when the availability of external resources had changed (Wąsowski, 2020). The recent development of software containerization and microservices aims to alleviate the dependency constraint. They are lightweight approaches to virtualizing applications that have resulted in a significant increase in cloud application management. A container holds packaged self-contained, ready-to-deploy parts of applications, and if necessary, middleware and business logic to run the applications. Container engines are the foundation of tools like Docker, which act as portable means of packaging applications (Pahl, Brogi, Soldani, & Jamshidi, 2017). Microservices is a software development technique that divides applications into multiple self-contained components that are served via APIs in a loosely coupled but coordinated manner. The main advantage of this approach is that each microservice is built independently of others, which can boost productivity and result in more resilient applications (Newman, 2015).

Time to market may accelerate significantly if a software development firm invests considerable resources into developing a flexible and robust software platform architecture that can accommodate the use of components to make numerous product variants. Clients may already have a development platform on which they can integrate several components to create products tailored to their specific requirements, particularly in the development of SOA, web services, open-source architectures, and cloud computing services. Containerization of software on cloud platforms is an enabler of such flexible customization because customization can be well described and isolated from each other (Schmid & Rummler, 2012).

The availability of software components for integration and reuse in a wide range of user-developed products and services can have significant implications for the software market and industry, similar to those seen in some manufacturing industries where manufacturers provide toolkits to facilitate user-led product modifications and innovations (Baldwin & von Hippel, 2011). Previous research has identified supply constraints and acquisition costs of external software components for reuse as the major barriers for developers and clients to adopt software component reuse strategies (Ravichandran & Rothenberger, 2003). The high acquisition cost is largely due to a lack of coordination between suppliers and buyers. Vendors are unsure of what components to offer or how much to charge for them, and business customers are unsure of where to look for useful components or how much to pay for them (Hong & Lerch, 2002). Consequently, the software industry suffers from a scarcity of and incorrectly priced components for sale to customers contemplating or implementing software component reuse strategies.

2.2 Software component reuse and software licensing

Software components are a subset of digital products, so they bear the same characteristics as any other digital product. Once created, software components can be replicated and distributed at near-zero cost and used in new software development projects. Meanwhile, components that perform the same functions can be substituted with each other with necessary modifications for dependencies and adaptation to the new working environment setting. The containerization of software makes it even easier to reuse and substitute software components (or microservices). Because containerized applications are abstracted from the host operating system, they are portable and can run uniformly and consistently across any platform or cloud.

Using software components has expanded a company's potential ability to develop large-scale, high-quality business solutions powered by modularized components. Reusability is an important feature of a high-quality software component, and software components should be designed and implemented in such a way that they can be reused by many different programs. Previous studies have emphasized the importance of software reuse, emphasizing the direct and indirect costs of reinventing software. Systematic software reuse is one of the most effective software engineering approaches for obtaining benefits related to productivity, quality, and cost reduction (de Almeida, 2019).

Prior literature on software product line engineering (Pohl, Böckle, & van der Linden, 2005) has established the link between the reusability of components in the product platform and the ability to increase product variety. To achieve time to market, consistency across products, cost reduction, better flexibility, and better management of change requirements, software product lines emphasize proactive software reuse and adaptation, interchangeable components, and multiproduct planning cycles (Bastos et al., 2017). Previous research also pointed out that software product line requires anticipating the future software compositions and thus endows flexibility only in predefined ways; thus, not every change or customer demand can be satisfied (Clements & Gacek, 2001).

Most previous research has examined software reuse within organizations where initially the firm would develop an in-house repository of software components to be reused across different projects (e.g., Mørch et al., 2004; Lakshmi Narasimhan & Hendradjaya, 2007; Mahmood, Niazi, & Hussain, 2015). In such cases, a rich collection of reusable components is necessary for component-based development. Lemley and O'Brien (1997) noted the importance of markets for software components, impact of standardization of software components, and network externalities—increase in socioeconomic value of software components as their widespread use increases. According to Ravichandran and Rothenberger (2003), component-based development becomes truly cost effective only when software components can be purchased in the market rather than developed in-house. But the vendors' inability to understand the market needs and to identify a selection of components for which there is demand has been a big barrier for software producers and has been creating severe supply constraints for client organizations that pursue reuse strategies, even for just building mature intrafirm component reuse platforms (Ulkuniemi & Seppanen, 2004). Previous research indicated that in business practice, organizational adoption of software development utilizing external software components also remains low (Clements & Gacek, 2001; Stefi, Lang, & Hess, 2016; Bastos et al., 2017). Although available reuse frameworks do provide technical guidelines for the development and acquisition of components, they do not offer insights on how market dynamics determine component supply and pricing. Effective market designs that facilitate price discovery have not drawn much interest and attention from researchers and practitioners (Hong & Lerch, 2002).

Open-source development is fundamentally based on reusing, cocreating, and modularizing software code (Lerner & Tirole, 2002). The openness of this process harnesses the power of a community of external professionals who contribute to the further development of the software code. Han et al. (2012) examined the economic value of open innovation alliances (OIAs) and discovered that the degree of openness of the OIAs is significantly related to the amount of returns accruing to the partnering firms. Consequently, several software firms have been active in promoting open-source development as evident in Microsoft's acquisition of Revolution R, IBM's acquisition of Red Hat, and Google's acquisition of Kaggle. Meanwhile, because clients require software applications for a specific market niche, open-source software components may fall short of providing all of the necessary functionality (Ven & Mannaert, 2008).

Permission for the reuse or modification of software components is regulated under software licenses. Choudhary (2007) found that software licensing models have impacts on market performance in terms of investment in product development, software quality, producer profits, and social welfare. Proprietary software and free and open-source software (FOSS) are two common types of software licenses which are associated with different business models. Buyers of proprietary software are typically not permitted to modify or reuse a software product or embedded modules. FOSS software grants the customer both modification and reuse rights and bundles the modifiable source code with the software (Sen, 2007; August, Shin, & Tunca, 2013).

Although the provision of open-source licensing rights to proprietary components is considered as a risk in that firms may lose control of their components, releasing software as black-box components will mitigate such risks and particularly reduce customization costs (Ravichandran & Rothenberger, 2003). In black-box components, only binary code is available in executable form, allowing for adaptations via extension languages and APIs while keeping source code inaccessible. Because a component acts without changing its source code and communicates with other components via interfaces over protocols, black-box reuse is particularly suitable for software components (e.g., microservices). Although software licensing has been extensively studied in the literature (Di Penta, German, Guéhéneuc, & Antoniol, 2010), our study captures the black-box reusability of software components that have not been considered in prior literature in software licensing. Understanding the impact of such black-box reuse licensing is especially important in today's software markets, where many software firms have shifted their software product provisioning to the web or cloud. Users of these web or cloud-based software can integrate different components for specific business functions on vendor platforms, such as Amazon AWS and Microsoft's Azure, despite not having access to the source codes.

Modern economic growth theory (Weitzman, 1998; Romer, 1986) and the "private-collective" innovation model (von Hippel & von Krogh, 2003) emphasize the strong economic dynamics of reuse and recombination of existing "components" and advocate reusing, cocreating, and modularizing software code (Lerner & Tirole, 2002). Researchers have advocated for a balance between intellectual property rights protection for producers and client flexibility in using the components (Lemley & O'Brien, 1997). O'Hern and Rindfleisch (2010) described open-source software practices to explain the highest level of cocreation as a collaboration of various actors (original software designers and subsequent ones who modify and extend the software) that develop software (in an ongoing process). Open-source software is premised on an open license (typically a form of the General Public License, GPL) that lets developers reuse, modify, and extend source code. Open-source software development is used in practice to develop upgrades and new releases of a particular software product or to customize software solutions (at a client site).

3 Experimental Design

Table 1. Key Design Elements of our Experimental Settings

Element	Description
Software product	The software products in our study refer to containerized software applications, implemented with three components (microservices) each. These containerized applications are portable and able to run uniformly and consistently across any platform or cloud.
Software component	The software components are microservices that are self-contained components of each application and serve via APIs in a decentralized manner. Microservice Architecture's focus is to modularize the application by building independent services and ensuring the scalability of the product.
Software component reuse	Software component reuse refers to taking a specific component from one software product and using it in another product (Sommerville, 2016). In our study, each application's microservices are reused to create new software applications at the client site. It is the systematic, black-box reuse of components from external sources/markets (Flakes & Terry, 1996).
Software platform	The platforms refer to platforms (e.g., cloud computing services platforms) where vendors sell and clients buy the software. On the platforms, the clients can customize and integrate the components purchased from the marketplace.
Supply and demand	Heterogeneous demand—each software product has different levels of demand and valuation to clients. Each software product incurs a fixed production cost with zero marginal cost for producing additional units.
Price discovery	Vendors and clients interactively discover the best prices for the software products through a standard, continuous double auction mechanism.

Our research is intended as a market design study in that we propose and evaluate specific reuse licensing model for software components combined with a specific price discovery mechanism. We provide human subjects a simple experimental software component market to see to what extent software traders in the laboratory can realize the economic potential of a specific market design. This research logic is adapted from the experimental economics literature (Davis & Holt, 1993; Smith, 1994) and is also increasingly used in the information systems field, especially on the topics of auction design, electronic market design, and smart markets (Kauffman & Wood, 2008; Gupta, Kannan, & Pallab, 2018). Approaches based on experimental economics have been applied to complex business contexts such as business negotiations, business contracts, and market design (e.g., Milgrom, 2004). Wohlin et al. (2012) have similarly discussed the need for experimentation in software engineering to understand and identify relationships between various factors.

Our study aims to investigate a market design feature that drives software component reuse and component-based development. To do so, we must abstract CBSD sufficiently and limit the complexity of the experimental task so that participants can complete it within the time constraints we have. To facilitate experimental control, we use abstract software products in our experimental settings. The simple design captures the salient features of the market environment that we must model in our lab market. In our case, that is (1) software products, (2) CBSD, and (3) software reuse at the component level (reuse of components to make new software products). Table 1 lists the key design elements of our experimental settings.

Like any experiment, the design requires the experimenter to make several design choices that will limit the generalization of the findings. The design should be simple (typically modeling an abstract/idealized and simplified market exchange economy) and focus on modeling the salient features (as theorized in the literature) of the market (products to be traded, product features, trade participants—buyers/sellers, trading rules and the price mechanism, and supply and demand schedules) while abstracting away details that are not part of the theory. To strengthen the validity of the experiment, we made design decisions carefully by looking at (1) the literature for theoretical guidance on the design parameters, (2) other related experimental studies to borrow best research practices, and (3) practical considerations (simplicity, budget, and realism). If certain parameters could not be predetermined through those considerations, we conducted pilots to compare and choose from some combinations as much as budget and resource constraints allowed (Davis & Holt, 1993).

3.1 Experimental context—reuse licensing model

The context for our experiment (see Appendices A1 and A2 for the experimental instructions for the vendor and client roles) is that of a digital platform where vendors sell and clients buy software, with the possibility of utilizing a reuse licensing model as a complement or alternative to open-source licensing models. Experiment participants take part in a simulated platform-based software marketplace for software vendors and business clients, which we designed to improve component supply and encourage client side reuse strategies. Uncertainty about how to best design a component so that it can be reused in different projects and uncertainty about market demand has been identified as the key factors that inhibit component production (Ravichandran & Rothenberger, 2003). We incentivize production by providing profit opportunities in our design, and we reduce uncertainty by using demand information shared in the marketplace in the form of bids and asks.

We assume the existence of effective standardization mechanisms and search mechanisms (e.g., digital catalogs) for vendors and clients in our experimental environment. The development and widespread adoption of component infrastructure standards demonstrate the increasing maturity of component technologies by providing mechanisms to standardize interfaces and thus allowing for easier component integration and recombination. Additionally, component standardizations and component certifications significantly reduce the search costs for software components (Bass & Buhman, 2001). In terms of microservices, the separation from the host operating system allows containerized applications to be portable and run uniformly and consistently across any platform or cloud. On the client side, we assume that clients will be able to customize and integrate the components purchased from the marketplace via the platform. Vendors sell applications that can be decomposed into basic components (e.g., microservices) that make up the application. By utilizing their published API interfaces, our proposed reuse licensing model allows clients to reuse the components to build other application variants. Clients can, in a sense, reuse and create their applications using the components available in the applications they have purchased from the marketplace.

3.2 Supply and demand

Following standard principles of experimental economics (e.g., Friedman & Sunder, 1994), we captured the essence of software ecosystems (i.e., component-based software products, software component reuse, and vendor and client heterogeneity) while removing complexities that are present in real-world markets but are not linked to our specific research question. Software is an easily modifiable information product with a high fixed cost, a low marginal cost, and variable demand (Choi, Stahl, & Whinston, 1997). We created sufficiently abstract software products and client reusability so that participants could complete the tasks in a reasonable amount of time.

Our experimental environment represents a simplified, idealized economy of vendors and clients of 16 component-based applications (A1–A16) built from nine basic software components (C1–C9) as shown in Table 2. Each application is made up of three black-box software components (microservices) and is intended to support a specific business function. Previous research has used simplified software component designs because they help increase experimental control and allow the experimenter to focus on the specific theoretically salient attributes of software components (e.g., Hong & Lerch, 2002; Anguswamy & Frakes, 2012). Each application has a given fixed production cost (from a random cost schedule that ranges from 100 to 500) to develop the software and provide it on the platform. The marginal cost for producing additional units is set to 0. There are 10 customers in the economy who represent corporate clients, and each client must acquire five specific applications, each with a predefined component structure[4], either through direct purchase in the marketplace or through client side development with component reuse in the reuse treatment). We model variation of demand for applications in a way that each application has different levels of demand in the marketplace, as suggested by Hong and Lerch (2002). The vendor's task is to sell applications for profit in the marketplace. As shown in Table 2, the profit potential of the applications varies; some allow for high returns while others may not be profitable at all. The vendor can choose to offer none, one, two, three, or all four applications in the marketplace.

Table 2. Valuation and Cost Schedule of 16 Component-Based Applications

Software products	**Client valuation to product (e.g., CV1 refers to client 1's valuation)**										**Total valuation**	**Production cost**
	CV1	CV2	CV3	CV4	CV5	CV6	CV7	CV8	CV9	CV10		
A1(C1/C2/C3)	75	70	105	45	15	180	65	255	5	65	880	420
A2(C4/C5/C6)	15	85	70	50	270	50	60	25	50	180	855	270
A3(C7/C8/C9)	90	45	150	20	60	40	10	70	30	80	595	200
A4(C8/C7/C3)						250	200	170	100		720	320
A5(C4/C2/C3)			200	150	100						450	490
A6(C6/C8/C9)	120	70								180	370	200
A7(C4/C6/C3)		400									400	400
A8(C1/C5/C4)										380	380	470
A9(C3/C7/C8)				360							360	320
A10(C1/C2/C8)	310										310	300
A11(C4/C6/C9)								300			300	370
A12(C4/C7/C3)			300								300	490
A13(C4/C7/C7)						280					280	270
A14(C9/C6/C9)									200		200	100
A15(C5/C8/C7)					150						150	100
A16(C1/C7/C6)							150				150	200

3.3 Experimental treatments

Our experimental design comprises of two treatments. In the baseline treatment, vendors sell component-based applications without the proposed reuse licenses. Vendors sell software applications bundled with a component reuse license that allows for black-box reuse of components in any application at the client's site in the component reuse treatment. When a client buys a software application from a vendor, he also buys a black-box reuse license that is linked to the software components. Clients are given reuse rights to create applications on the platform on the basis of the components available in the applications that they have purchased from the market. For example, as shown in Table 2, the software application A1 comprised the three components C1, C2, and C3. Although the functionalities of these components cannot be changed because the source code is not available, they can be reused in the development of other product variants by utilizing their published interfaces. Similarly, software application A2 is made up of the components C4, C5, and C6. Assume a client wants to create a software application A7 that includes the components C4, C6, and C3. They have the choice to purchase it from the market directly, or they can reuse components from other software applications if available. An attribute of software components is their substitutability that a component can replace another if the components meet the same requirements. If they have the necessary component reuse rights for A1 and A2, they could

[4] In our design we feature multiple customers and multiple applications. Few experimental economics lab studies have more than 10 buyers or 5 sellers and 5 different products. Research has shown that just 5 buyers or sellers are sufficient to generate efficient, competitive market behavior in lab experiments (Davis & Holt, 1993).

recombine component C3 from A1 with components C4 and C6 from A2 to make A7. For the sake of simplicity, we will assume that vendors incur no costs in disassembling and reassembling the components they have purchased.

3.4 Price discovery

Vendors and clients interactively discover the best prices for the software applications and their embedded components through a standard continuous double auction (CDA) mechanism. Clients can submit bid prices or accept ask prices, and vendors can submit ask prices or accept bid prices. The trading system automatically matches bids and asks at the same price.

Among different auction mechanisms (English, Dutch, silent, sealed bid, etc.), CDA is the most efficient one in two-sided markets. Centralized CDA mechanisms are truly dynamic and quite complex. This has resulted in the relative lack of theoretical models that explain their efficiency and convergence properties. Many models have been tested and evaluated in the laboratory, but none have emerged as a standard (see Parsons et al. (2011) for a review of theoretical, computational, and experimental results, as well as Friedman and Rust (1993) for discussion of most models). Nonetheless, the general agreement is that the CDA mechanism has shown remarkable robustness of operational efficiency and price convergence under many underlying economic environments in a diverse collection of controlled laboratory experiments (Smith, 2003).

We used VB.net to create the entire experimental platform, including the CDA mechanism. Our experimental system, like a stock market system, automatically posted, updated, and matched bids–asks; reported transactions; and calculated participants' gains or losses.

3.5 Participant pool and participant compensation

We had an opportunity to run a laboratory study while we had access to a limited number of industry executives and used a small convenience sample of 28 IT professionals for our experiment. They were recruited from a graduate course in an information systems program at a public university in the United States, and they served as organizational proxies in the experiment, making organizational software procurement and development decisions. As shown in Table 3, our participants had work experience in the software industry in the position of software developer, project manager, system analyst, or others. Participants were randomly assigned to the roles of a vendor or a client. We did not find any significant differences among the cohorts in terms of gender, age, or experience.

Table 3. Job Positions of the Participants

Job position	Number of participants
Software development	8
Project manager	7
System analyst	5
IT professional (help desk, IT associate, IT coordinator, etc.)	8
Total	28

The participants who voluntarily participated in the experiment were given a consent form that clearly states that they are free to stop or withdraw from the experiment at any time, following standard Institutional Review Board requirements. If participants did not wish to participate in the study, they were given an alternative way to earn course credit.

Participants were incentivized with performance-based course credit. After each experiment session, participants' total profit (gain or loss) from the experimental market was calculated and ranked. Participants who made a larger total profit received more course credits. Furthermore, to encourage participants to maximize their market profits, a bonus credit was awarded to the participants with the highest profit in each of the client and vendor groups. We checked the distribution of vendor/client profit in the experiment, and it does not show skewness that might be caused by the credit incentives.

3.6 Experimental procedure

The experiment is a mixed design. The baseline and reuse treatments are with two between-subject groups (half the participants participated in the baseline treatment, and the other half, in the reuse treatment). Both the baseline and reuse treatments went through eight rounds (periods), resulting in within-subject repeated measures. To reduce the effects of repeated experimental rounds, we rotated the 10 consumer demand schedules between rounds, so that each participant received a different demand schedule (with different product assignments and valuations).

After the participants came to the experimental laboratory, they were seated at separated individual workstations that served as trading terminals for selling or buying products in the marketplace and provided with instructions about their specific experimental task as buyer or seller (see Appendix A). During the experiment, participants were not allowed to communicate with one another except to post bids and asks during trading periods. In both treatments, vendors and clients discover the prices of software applications interactively via a standard trading mechanism, which we implemented as a version of CDA. Sellers can submit asks or accept bids. Similarly, buyers can submit bids or accept asks. Bids and asks at the same price are matched automatically by the trading system. We conducted eight rounds for each of the two experimental treatments: the baseline (no reuse licenses) and the component reuse treatment (with reuse licenses). Each vendor received their production schedule and associated fixed costs for producing four software products at the start of each round. They knew neither the other vendors' production schedules nor any client side demand schedules. Throughout the round, they monitored the marketplace with an online trading system (as shown in Appendix A) and check the bids that clients who were interested in their products were posting. Vendors decided whether and when to produce and market a software product based on market signals about client demand and known production costs. Vendors may have decided to only sell products on which they expect to profit.

The clients began each round by receiving their private demand schedule, which told them the software products they needed to buy in the present round and what the products were worth to them. Each client received a different set of five products that had different valuations. They were unaware of the other clients' demand schedules or vendor cost figures. They could, however, see ask prices for products currently available in the marketplace, enter new bids into the system at any time, and receive instant feedback when transactions were completed (also shown in Appendix A). The allocation of products and associated valuations changed for clients from round to round.

3.7 Measurements of market performances

We analyze the impacts of the component reuse license on market performances with four variables, the number of software applications obtained by clients, the acquisition cost of clients, production costs of vendors, and vendor profit (as shown in Table 4).

Table 4. Measurements of Market Performances

Variable	Definition	Measurement
NA_r	Total number of software applications obtained by clients in round *r*	$NA_r = \sum_{i=1}^{10} N_{ir}$ N_{ir}: number of applications obtained by *a* client *i* in round *r*
AC_r	Average acquisition cost per software application of clients in round *r*	$AC_r = \frac{1}{NA_r}\sum_{n=1}^{NA_r} P_{nr}$ P_{nr}: *the* price paid by clients for application *n* in round *r*
PC_r	Total production cost of vendors in round *r*	$PC_r = \sum_{j=1}^{16} C_j * B_{jr}$ C_j: product cost of application *j* in round *r* B_{jr}: dummy variable; 1 if application *j* is offered by vendors in round *r*, 0 otherwise.
TP_r	Total profit of vendors in round *r*	$TP_r = \sum_{n=1}^{NA_r} P_{nr} - PC_r$

4 Results

In our analysis, we conducted the repeated measures analysis of variance (ANOVA) test to statistically control within-subject repeated measures. The experimental round, the within-subject factor, showed no significant effect ($p > 0.19$) on the production cost, profit, etc. Because our sample size is small, Mann–Whitney nonparametric U-test is also conducted, and the results show significant ($p < 0.01$) impacts of reuse license on the dependent variables listed in Table 5.

4.1 Market performances

The results from our experiments suggest that introducing the proposed reuse licensing model does increase component supply in the market and reduce the acquisition costs for clients. At the same time, it increases vendor profits. Overall, we see a high level of participation in software component reuse strategies among market participants.

Table 5. Experimental Results

		Mean (standard deviation)		**Change**	**ANOVA p-*value***	***df* Residuals[5]**	**Effect size (omega squared)**
		Baseline treatment	**Reuse treatment**				
Client	Acquisition cost per software application	65.88 (4.44)	52.02 (2.28)	−21%	0.025	12	0.80
	Total number of applications obtained	21 (2.1)	32.1 (2.78)	+52.9%	0.01	12	0.84
Vendor	Total profit	−22.75 (137.32)	449.26 (173.98)	+472.01	0.01	12	0.68
	Total production cost	1406.25 (31.3)	1221.12 (48.99)	−13%	0.02	12	0.76

The experimental marketplace with component reuse licensing achieved substantial acquisition cost savings of 21% for procuring component-based software applications when a reuse license was available to the clients. As shown in Table 4, the acquisition cost per application was reduced from 65.88 in the baseline treatment to 52.02 in the reuse treatment. Clients were able to reuse software components in the reuse treatment to more efficiently create the applications they required for their business. Thus, this reduction of acquisition cost was mostly through a remarkable 52.9% increase of applications that clients obtained (an average of 9.11 applications were obtained by client side component reuse to create applications in the reuse treatment).

On the vendor side, we see that the total profits increased with the option to bundle applications with component reuse licenses to 449.26 from −22.75 in the baseline treatment, which represents a small loss. The total vendor profit (−22.75) was a small loss, that is, less than the monopolistic equilibrium value of 550 (see A3). Consequently, our participants were unable to capitalize on the full profit potential offered by our market design. We attribute this outcome to investment risk related to the demand uncertainty and high fixed production costs that the vendors were working with. There were a few software applications with low demand that resulted in losses because the vendors were unable to accurately estimate client demand. When we examine the potential causes of the dramatic increase in profitability, we discover that it can be attributed to better-informed production investment decisions. The total production investment (total cost of bringing products to market) for vendors in the baseline treatment is 1406.25, and it decreased in the reuse treatment by 13% to 1221.12. In the reuse treatment, we see that there is a high demand in the market for applications that include high-value components for which clients are willing to

[5] We conducted ANOVA using R. The calculation of df of residuals in R includes the df of between-subjects group (1), the *df* of within-subject (1), and the interaction of group and within-subject (1).

pay a premium. Therefore, the market signals helped vendors discover which applications cater to the clients' needs. In the reuse treatment, none of the applications offered resulted lost money for vendors.

4.2 Component utilization

From a software marketplace perspective, we also see an improvement in component utilization in the treatment group. The platform's 16 software applications are all built from the same 9 basic software components. Component utilization is defined as the actual reuse of a component. We keep track of how many and how often clients use components. The number of copies of a component used in applications obtained by clients is used to calculate component utilization. As shown in Figure 1, the utilization of all the components in the reuse treatment is better than the utilization of components in the baseline treatment. On average, allowing for reuse at the client site improved the utilization of components by approximately 41% from the baseline treatment. The majority of the improvement in component utilization resulted from component reuse at the client site. On average, the treatment results in 26% client site component reuse (clients reuse the component to create applications).

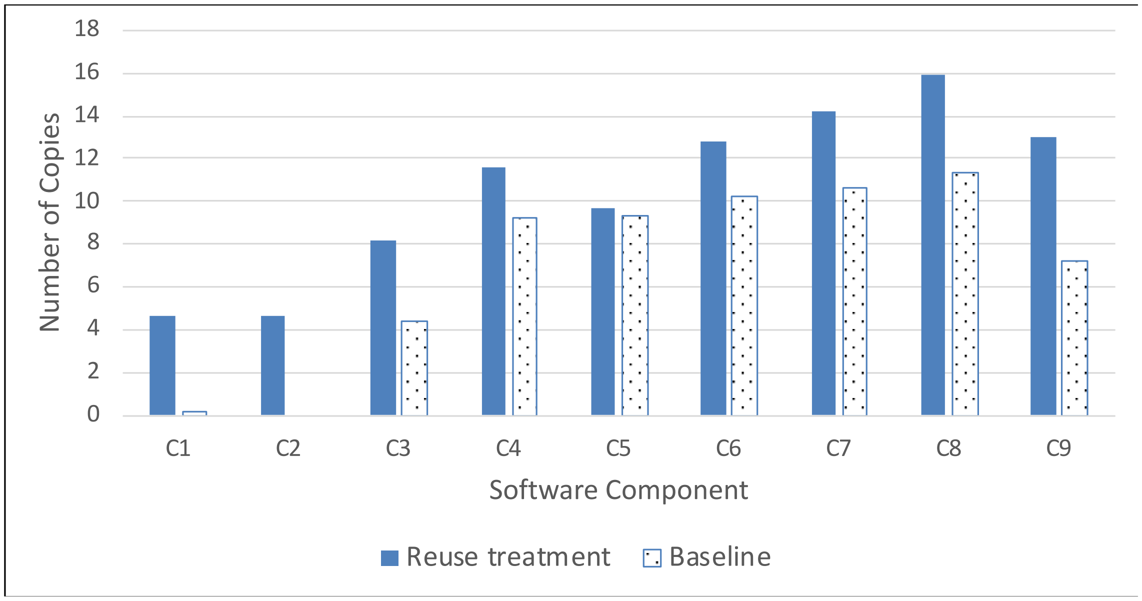


Figure 1. Component utilization at client site

4.3 Postexperiment questionnaire

In a postexperiment questionnaire, we asked the 28 IT professionals to answer five open-ended questions after they participated in an experimental session. We asked them to assess the licensing model for component reuse based on their observations of the market from the experiments. The authors created the open-ended questions on the basis of their observations, literature review, and experience. The experiment participants' responses and comments were transformed and organized using excel tables, and then, responses were categorized into the opinions listed in Table 6. Individual responses were identified from the transcripts with references to the individual statements to ensure traceability (Yin, 2003). All the responses were then used to identify results and draw conclusions.

Table 6. Summary of Postexperiment Interview

Opinion	**Number of participants holding the opinion (out of 28 participants)**
C1. The proposed component reuse benefits both clients and vendors	20
C2. Software should be more customized for the needs of clients	24

C3. The cost of the software will drop if clients have the option to reuse part of an application	20
C4. Standardization and centralization are the keys to the success of implementing this market	16
C5. The proposed market will increase competition in the software industry	8
C6. Such a market can be implemented in real	23

Our analysis of the responses presents some initial evidence that our proposed design is in principle feasible and potentially useful for the software industry. Because software components are reutilized, the professionals generally agreed that such a reuse license benefits the software industry (Table 6). “Commercial software vendors have an opportunity to learn from the open-source community and leverage that knowledge for the benefit of its commercial clients,” some said. Most clients thought that component reuse would allow them to be flexible, up to date with the latest technology, and potentially decrease their software investment.

Twenty-three participants thought that it would be possible to implement such a marketplace for component reuse. They did, however, emphasize that standardization and centralization are critical to the success of implementing such a market. It is much more difficult to build the actual system by combining various applications developed by various vendors. To reuse a component for different systems, the component must be compatible with others. Microservices and containerization of software have opened up new opportunities because containerized applications are portable and able to run uniformly and consistently across any platform or cloud. At the same time, vendors were concerned that the reuse license “brings competition among vendors” or, as one participant put it, “it is extremely difficult to get all vendors under one roof,” because software vendors may be familiar with traditional mechanisms for selling software licenses and it may be difficult to get them to participate in a single marketplace.

5 Discussion

5.1 Theoretical and practical implications

This paper contributes some initial empirical evidence that software marketplaces with black-box reuse can provide component liquidity, stimulate software reuse strategies, and reduce acquisition costs of software applications for clients as well as generate profits for vendors. According to our findings, software vendors should be encouraged to participate in software marketplaces that use flexible licensing models: Vendors providing clients with reuse rights for components will result in significant benefits for both vendors and clients. Although software development organizations have been investing in component-based development efforts on the basis of their ability to reduce time to market and to deliver higher quality products, there has been little attention on the potential implications of providing reuse of components to clients. Our findings indicate that this will benefit both parties. Vendors must balance the demand for a specific product offering with the costs of making it available as a stand-alone product. Many products are not available because there is insufficient demand to cover the costs of manufacturing and distributing them. Moreover, anticipating all variants and future software compositions is expensive and creates a large cost upfront. This implies that given a software development organization’s investment in CBSD processes, they can set aside their concerns regarding the potential loss of control of their components by using the black-box approach to reuse and then provide clients in the market with rights to reuse components to build other product variants. Our findings show that the benefits of CBSD extend not only to higher quality, flexibility, and lower building and maintenance costs but also to providing reuse rights to other market participants.

Clients should transform development processes to adopt software component reuse strategies and participate in the new software marketplaces. Our findings encourage organizations to modify their current development process to emphasize (cloud) platform-based approaches that incorporate reusable software components from various software vendors. Containerization makes it possible for applications to be “written once and run anywhere.” Clients can focus on the planned reuse of software products on platforms that are incorporated with the flexibility to accommodate component variants in the derivation of members of product lines. When such software platforms are built with standardized interfaces that allow clients to incorporate software components to create product variants, we can expect a significant increase in product variety.

Software platforms exist when they make it possible for vendors and clients to produce value-creating product and service transactions so they can do their business more effectively than they can without a middleman (Kauffman, Li, & van Heck, 2011). By promoting more flexible licensing agreements, software platforms can also play an important role in fostering innovation and software development. Platforms should focus on optimizing the value created by the ecosystem to maximize growth as systems for matching vendors and clients who transact with each other using system resources (Parker, Van Alstyne, & Jiang, 2017). The current industry practices, as we observed on the AWS platform, have been promoting value creation by implementing flexible pricing options include free trial, hourly, monthly, annual, and multiyearly. According to our findings, providing flexible licensing agreements will result in significant benefits for both vendors and clients. Furthermore, for software platforms, the creation of standards that allow for the flexible, rapid, and seamless integration of components, even by clients, will be crucial. The rapid improvements in network connectivity and computing power have brought various application integration solutions as we observe on cloud computing services platforms.

5.2 Limitations

Experimental findings always must be presented cautiously, and alternative explanations should be considered, especially when generalizing beyond the chosen parameter configuration. There is a tradeoff between internal and external validity in all experimental research (Smith, 2002). Experiments are strong in the former and usually allow for causal claims to be made, but they are weak in the latter because these causal claims are limited to the specific experiment. Within the constraints of our experiment’s design, our participants demonstrated their ability to develop successful reuse strategies in the lab. It remains unclear from our study if these findings would transfer to organizational settings where additional, organizational factors not modeled in the experiment could come into play and potentially change the outcome.

Our findings are based on one small scale, idealized economic experiment and as such should be interpreted with caution. We made several design decisions that limit the findings’ generalizability. First, we enlisted the help of a small group of IT professionals who were enrolled in a graduate university program. In this sense, our sample was not drawn at random from a population of IT professionals. Second, we implemented a simplified software economy in an artificial laboratory environment. Although we did make careful design choices on the basis of the literature and practical considerations, it is never possible to test all variable and parameter choices before moving to data collection. Consequently, promising untested parameter choices should be recommended for future research. Third, we conduct a brief experiment with only eight rounds. The effects of a market design frequently take time to become apparent. Future replications that assess the long-term effects of market design are required.

Despite the considerable market gains generated by the reuse licensing model, our experimental setting is not able to achieve the full projected surplus. As detailed in Appendix A3, the projected total surplus (the sum of all market participants’ profits) in the baseline treatment is 1750, and the projected total vendor surplus (vendor profit) is 550. The total vendor profit in our experimental setting is far lower than our projected figure. From an economic perspective, losses are just negative profits, and reducing losses is equivalent to increasing profits. Hence, it is appropriate for our study to use the experimental evidence and interpret loss reductions as profit increases in our findings. Businesses do take losses in reality, but in the long term, this may not be sustainable. Although our current parameterization does allow participants to find profitable decisions as they did in some rounds, they overall performed below optimal. A different parametrization could be used to make it easier for experiment participants to generate profits. However, whether or not absolute profits are positive, the main point is that the participants were able to improve financial performance. From economic theory, we know that in the presence of perfectly competitive markets, these surpluses should be achieved in practice. Nevertheless, our laboratory market and naturally occurring software markets are far from perfect and thus a consistent theoretical model that predicts how market participants will interact does not exist. It is not entirely surprising that the results are far from perfect. The double auction mechanism performed poorly in a laboratory study (Van Boening & Wilcox, 1996) with a market structure similar to ours. At the moment, it is unknown what type of market institution can achieve superior performance in the market structure under consideration. The fact that our market results are far from perfect means that the market structure we explore needs a different market mechanism than the double auction to achieve better results in either treatment.

5.3 Future research

Experimental research is by nature cumulative and our study can only be a starting point into the inquiry. It is widely accepted that no single experiment can claim external validity of its findings, that is, the confidence that the experimental findings can be applied outside of the laboratory. When considering a line of research (i.e., several related experimental studies that repeat and vary a basic experimental design), external validity can and should be established (Hamermesh, 2007). Thus, replicating results when reusing the same design and producing consistent results when varying design parameters and variable configurations will increase and eventually establish the external validity of experimental studies. It takes a series of related studies to replicate the core findings and investigate boundary conditions by refining and extending the research design, which includes changing some parameter settings to different values. Following that, we will go over some potential future research directions.

In our experiment, we chose to explicitly model the acquisition costs. In the literature, other reuse cost factors have also been discussed, including identification costs, modification costs, integration, and maintenance costs. A few cost estimation models, such as reuse based, risk-based, and product-line-based, have been developed to determine the cost of reuse (e.g., Frakes & Terry, 1996; Mili, Chmiel, Gottumukkala, & Zhang, 2000; Rothenberger & Nazareth, 2002). We propose that future research consider running additional experiments that change our design setting and include identification costs, modification costs, and integration costs to see if our basic findings can be replicated and held in different and more elaborate settings.

CBSD is more complex and richer than our simple three-component software products. Although software component standardization and certification have significantly reduced component search costs, identification costs remain significant when developing a software application that necessitates the procurement of numerous software components. In the context of microservices, the modification costs appear very low because microservices are portable and able to run uniformly and consistently across any platform or cloud. However, reuse of microservices will involve integration costs and those should be carefully assessed on the basis of cost-benefits, potential failure, product line model, etc. In our study, we assume that there will be no integration costs and that the clients will know exactly what business applications they will develop. It is worthwhile to investigate various scenarios in which, for example, software component reuse and recombination are used as an inventive process for new products. The integration costs would then be significant, potentially affecting market outcomes.

Many other designs are possible besides the specific setting that we studied in this paper. Future research should consider running additional experiments that change some design parameters such as supply and demand schedules, different market mechanisms other than CDA, reuse license specifics, number of products, and number of market participants. We also suggest considering complementing laboratory experiments with field experiments and case studies, which would allow us to examine the adoption of software reuse strategies in an everyday context. Other intriguing research directions include developing new experiments that explicitly incorporate open-source code libraries and cloud-based development platforms.

6 Conclusions

We propose an economic solution and offer a specific software component reuse licensing scheme to facilitate better coordination between vendors and clients. Our experiments in a simplified lab environment show that by participating in the software marketplace, software vendors and clients can overcome coordination failure and achieve higher IT capabilities at client sites. Using a small-scale economic experiment, we can provide evidence to support the economic impacts of “black-box” reuse of software components, which was advocated by Ravichandran and Rothenberger (2003). Although they theoretically predict the potential benefits of market approaches on the software market, our study is the first that examines a specific market design—a component reuse license—to further improve the market performance and promote component-based development. Moreover, with our experimental study, we were able to test our market design that built on literature using assumptions that are less restrictive than in previous theory.

We believe that findings from our study provide some valuable insights into the importance of designing appropriate software licensing models for purposes of promoting software component reuse strategies. By selling them in liquid marketplaces, software vendors will increase component reuse rates and make software reuse more cost effective. Coordination failure occurs in a market with uncertain demand when software vendors are unable to focus on developing software applications with the appropriate market components. It is critical to have adequate mechanisms in place to facilitate an efficient price discovery process. Vendors will not capture the possible benefits of black-box reuse if the market does not reflect the underlying value of reuse. Consequently, appropriate mechanisms should make price discovery easier and allow vendors to “find it easier to discover demands for software applications,” as one of our participants stated in the postexperiment questionnaire. Consequently, the platform will direct vendors to concentrate on the creation of software applications that generate value that can be shared by vendors and customers.

There has been limited empirical evidence on the impact of specific market designs on software component reuse. The experiments show that the market failure that is evident in the current software component markets can be overcome with an appropriately designed economic solution, a market design that includes a more openly structured reuse license than what is currently available in the commercial component market. The software reuse literature has mostly looked at technical fixes, organizational factors, or user attitudes to explain the software component market failures. Our research provides an alternative point of view, proposing an economic solution to the problem. Simultaneously, our limited experiment outlines a new line of investigation and calls for more in-depth experiments on this issue to replicate and strengthen the validity of our findings. It may encourage researchers to tackle the economical dimension of reuse activities by exploring market mechanisms.

References

Anguswamy, R., & Frakes, W. B. (2012). A study of reusability, complexity, and reuse design principles. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*.

Atish, P. S., & Hemant, J. (2013). Ease of reuse: An empirical comparison of components and objects. *IEEE Software, 30*(5), 70-75.

August, T., Shin, H., & Tunca, T. I. (2013). Licensing and competition for services in open source software. *Information Systems Research*, *24*(4), 1068–1086.

Baldwin, C., & von Hippel, E. (2011). Modeling a paradigm shift: From producer innovation to user and open collaborative innovation. *Organization Science*, *22*(6), 1399–1417.

Bass, L., & Buhman, C. (2001). *Volume I: Market assessment of component-based software engineering*. Carnegie Mellon Software Engineering Institute.

Bastos, J., da Mota Silveira Neto, P., Oleary, P., de Almeida, E., & de Lemos Meira. S. (2017). Software product lines adoption in small organizations. *Journal of Systems and Software*. *131*, 112-128.

Bertoa, M. F., Troya, J. M., & Vallecillo, A. (2006). Measuring the usability of software components. *Journal of Systems and Software*, *79*(3), 427–439.

Choi, S., Stahl, D., & Whinston, A. (1997*). The economics of electronic commerce*. Indianapolis, IN: Macmillan Technical Publishing.

Choudhary, V. (2007). Comparison of software quality under perpetual licensing and software as a service, *Journal of Management Information Systems*, *24*(2), 141-165.

Clements, P., & Gacek, C. (2001). Successful software product line development in a small organization, in *Software product lines: Practices and patterns* (pp. 485–512). Addison Wesley Longman.

Crnkovic, I., Sentilles, S., Vulgarakis, A., & Chaudron, M. R. V. (2011). A classification framework for software component models*. IEEE Transactions on Software Engineering*, *37*(5), 593–615.

Davis, D., & Holt, C. (1993). Experimental economics: Methods, problems, and promises. *Experimental Economics*, *8*(2), 179-212.

de Almeida, E. S. (2019). Software reuse and product line engineering. In: Cha S., Taylor R., Kang K. (eds) *Handbook of software engineering*. Springer.

Di Penta, M., German, D. M., Guéhéneuc, Y. G., & Antoniol, G. (2010). An exploratory study of the evolution of software licensing. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*.

Frakes, W., & Terry, C. (1996). Software reuse: Metrics and models. *ACM Computing Surveys*, *28*(2), 415-435.

Friedman, D., & Rust, J., (1993). *The double auction market: institutions, theories and evidence*, Perseus Publishing, Cambridge.

Friedman, D., & Sunder, S. (1994). *Experimental methods: A primer for economists*, Cambridge University Press, NY.

Gupta, A., Kannan, K., & Pallab, S. (2018). Economic experiments in information systems*, MIS Quarterly*, *42*(2), 595-606.

Hamermesh, D. S. (2017). Replication in labor economics: Evidence from data and what it suggests. American Economic Review, *107*(5), 37–40.

Han, K., Oh, W., Im, K., Chang, R., Oh, H., & Pinsonneault, A. (2012). Value co-creation and wealth spillover in open innovation alliances. *MIS Quarterly*, *36*(1), 291-325.

Hong, S. J., & Lerch, F. J. (2002). A laboratory study of consumers' preferences and purchasing behavior with regards to software components*. ACM SIGMIS Database*, *33*(3), 23–37.

Kauffman, R. J., & Wood, C. (2008). Research strategies for e-business: A philosophy of science view in the age of the Internet. In R. Kauffman & P. Tallon (Eds.), *Economics, information systems, and electronic commerce: Empirical research*. New York: M. E. Sharp.

Kauffman, R., Li, T., & van Heck, E. (2010). Business network-based value creation in electronic commerce. *International Journal of Electronic Commerce*, *15*(1), 113-144.

Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys*. *24*(2), 131–183.

Lakshmi Narasimhan, V., & Hendradjaya, B. (2007). Some theoretical considerations for a suite of metrics for the integration of software components. *Information Sciences*, *177*(3), 844–864.

Lang, K., Shang, R., & Vragov, R. (2015). Consumer co-creation of digital culture products: Business threat or new opportunity? *Journal of the Association for Information Systems*. *16*(9), 766-798.

Lemley, M. & O'Brien, D. (1997). Encouraging Software Reuse. *Stanford Law Review*, *49*, 255.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics*, *50*(2), 197-234.

Mahmood, S., Niazi, M., & Hussain, A. (2015). Identifying the challenges for managing component-based development in global software development: Preliminary results. In *2015 Science and Information Conference*.

Milgrom, P. (2004). *Putting auction theory to work*. Cambridge University.

Mili, A., Chmiel, S., Gottumukkala, R., & Zhang, L. (2000). An integrated cost model for software reuse. In *Proceedings of the 2000 International Conference on Software Engineering*.

Mørch, A. I., Stevens, G., Won, M., Klann, M., Dittrich, Y., & Wulf, V. (2004). Component-based technologies for end-user development*.* *Communications of the ACM*, *47*(9), 59-62.

Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media, Inc.

O'Hern, M. S., & Rindfleisch, A. (2010). Customer co-creation: A typology and research agenda. *Review of Marketing Research*, *6*, 84-106.

Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2017). Cloud container technologies: A state-of-the-art review*. IEEE Transaction on Cloud Computing*, *7*(3), 677-692.

Parker, G., Van Alstyne, M., & Jiang, X. (2017). Platform ecosystems: How developers invert the firm. *MIS Quarterly*, *41*(1), pp. 255-266.

Parsons, S., Rodríguez-Aguilar, J., & Klein, M. (2011). Auctions and bidding: A guide for computer scientists. *ACM Computing Survey*, *43* (10), pp. 1-59.

Pettey, C. (2019). 6 Best Practices for Creating a Container Platform Strategy. Gartner Report, April 23, 2019. https://www.gartner.com/smarterwithgartner/6-best-practices-for-creating-a-container-platform-strategy/

Pohl, K., Böckle, G., & van der Linden, F. J. (2005). *Software product line engineering: Foundations, principles and techniques*. Springer.

Ravichandran, T., & Rothenberger, M. A. (2003). Software reuse strategies and component markets*.* *Communications of the ACM*, *46*(8), 109–114.

Romer, P. (1986). Increasing returns and long-run growth. *Journal of Political Economy*, *94*(5), 1002-1037.

Rothenberger, M., & Nazareth, D. (2002). A cost-benefit-model for systematic software reuse. In *Proceedings of the 2002 European Conference on Information Systems*.

Schmid, K., & Rummler, A. (2012). Cloud-based software production lines*.* In *Proceedings of the 16th International Software Product Line Conference.*

Sen, R. (2007). A strategic analysis of competition between open source and proprietary software*.* *Journal of Management Information Systems*, *24*(1), 233–257.

Sherif, K., & Menon, N. M. (2004). Managing technology and administration innovations: Four case studies on software reuse. *Journal of the Association for Information Systems*, 5(7), 247-281.

Sinha, A. P., & Jain, H. (2013). Ease of reuse: an empirical comparison of components and objects*. IEEE Software*, *30*(5), 70–75.

Smith, V. L. (1976). Experimental economics: Induced value theory*. American Economic Review*, *66*(3), 274–279.

Smith, V. L. (1994). Economics in the laboratory. *Journal of Economic Perspective*. *8*(1), 113-131.

Smith, V. L. (2002). Method in experiment: Rhetoric and reality, *Experimental Economics*, *5*(2), 91-110.

Smith, V. L. (2003). Constructivist and ecological rationality in economics. *American Economic Review*, *93*(3), 465–508.

Smith, V. L. (2010). Theory and experiment: What are the questions? *Journal of Economic Behavior & Organization*, *73*, 3-15.

Sommerville, I. (2016). *Software engineering*. 10th Edition. Pearson.

Stefi, A., Lang, K. R., & Hess, T. (2016). A contingency perspective on external component reuse and software project success. In *Proceedings of 22nd Americas Conference on Information Systems.*

Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley.

Ulkuniemi, P., & Seppanen, V. (2004). COTS component acquisition in an emerging market*. IEEE Software*, *21*(6), 76–82.

van Boening, M., & Wilcox, N. (1996). Avoidable cost: Riding the double auction roller coaster. *American Economic Review*, *86*(3), 461-477.

Ven, K., & Mannaert, H. (2008). Challenges and strategies in the use of open source software by independent software vendors. *Information and Software Technology*, *50*(9-10), 991-1002.

Wąsowski, A. (2020). Dependency bugs: The dark side of variability, reuse and modularity. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*.

Weitzman, M. L. (1998). Recombinant growth. *Quarterly Journal of Economics*, *43*(2), 331-360.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Kluwer Academic Publishers.

Yin, R. K. (2003). *Case study research: Design and methods*. Sage Publications.

Yu, Y., Lu, J., Fernandez-Ramil, J., & Yuan, P. (2007). Comparing web services with other software components. In *Proceedings of 2007 IEEE International Conference on Web Services*.

Appendix A: Experimental Instructions

A.1 Instructions to Vendors

General Overview

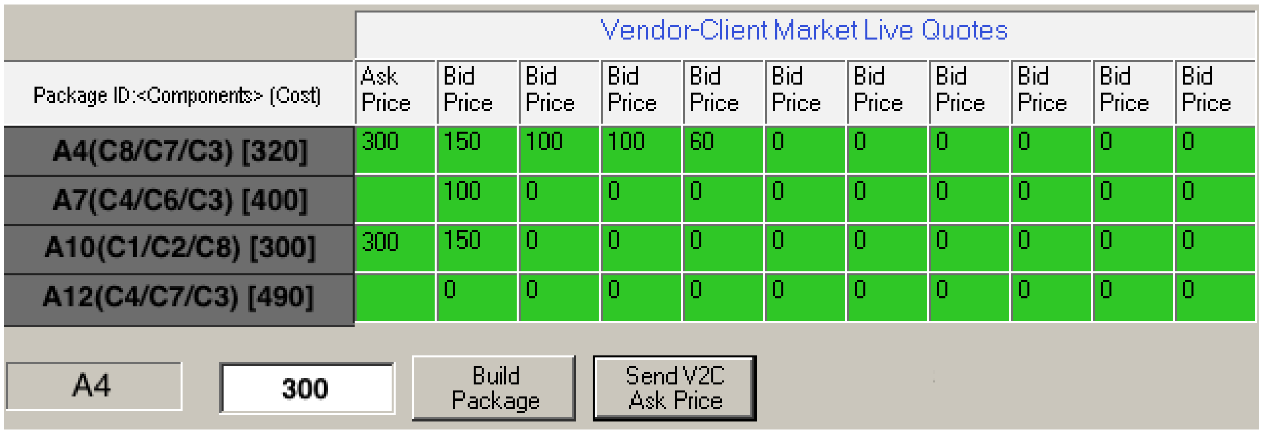
You are an executive of a software company. Currently your company has the copyrights over four specific e-commerce applications. The software applications are component based, and each of them is designed for supporting a different business function. For example, software application A1(C1/C2/C3) provides a business solution, and it includes three reusable software components, C1, C2 and C3. Each application has a given fixed production cost to develop the software and bring it to market. For example, the application A1 will cost you 180 to produce. Your task is to sell each of the 4 software applications for the best (highest) possible price in a vendor-to-client marketplace (within a sales period, simulated by a round of approximately five minutes). Each application may have different levels of demand in the marketplace. Some s can sell multiple copies. You can decide to offer none, 1, 2, 3, or all 4 of your applications in the market.

Reuse and Recombination (for reuse treatment)

You sell your software s with a reuse license with which the customers can reuse and integrate the components to develop software s for their specific needs. Once a customer develops a software application on his own, his demand for this software application will be satisfied.

Your Profit

The application profit you earn is equal to the price the buyers pay for the application minus your initial production costs. E.g., if you made application A1 for 180 (from scratch) and sold 1 copy each to two clients for 100 and another copy to a third client for 50, your profit will be (100 + 100 +50) - 180 = 70. The profit you earn for the round is equal to the total sales you made, less the total costs of producing your applications. Your total profit will be the profits totaled over all rounds in the experiment.



	Vendor-Client Market Live Quotes										
Package ID:<Components> (Cost)	Ask Price	Bid Price	Bid Price	Bid Price	Bid Price	Bid Price	Bid Price	Bid Price	Bid Price	Bid Price	Bid Price
A4(C8/C7/C3) [320]	300	150	100	100	60	0	0	0	0	0	0
A7(C4/C6/C3) [400]		100	0	0	0	0	0	0	0	0	0
A10(C1/C2/C8) [300]	300	150	0	0	0	0	0	0	0	0	0
A12(C4/C7/C3) [490]		0	0	0	0	0	0	0	0	0	0

A4 300 Build Package Send V2C Ask Price

Figure A1. Experiment interface for vendors

During the experiment vendors are working with a trading screen as shown in Figure A1. Vendors can see the fixed production cost of the four software applications (in this example, A4, A7, A10 and A12). They can monitor demand from clients (bid prices) as they are posted in the market. Vendors can decide whether or not produce any of the products based on their assessment of the demand. When they produce an application and make it available in the market, they post an ask price. The information is broadcast to clients in real time. In the example, the vendor produced software application A4 given the market showed a total demand of 410 (150+100+100+60) at the moment. The vendor posted an initial ask price of 300. Vendors can lower their ask prices at any time after they post the initial ask prices. The vendor in this example can set the ask price of A4 at different points so that one copy can be sold for 150, two for $100 and one more for $60. That is, at this point A4 can generate 410 revenue, which translates into 90 profit. The producer has not offered A7 or A12. While there are outstanding bids for A7 and A10, it is not certain that the demand is strong enough to recoup the production cost. The vendor may wait for higher bids, or shelve the product and let the demand go unfulfilled. If the vendor produces an application but fails to cover the cost from sales, the vendor will incur a loss.

The system keeps track of inventory, sales, and profits. The profits of the current round as well as the accumulated total profits over all completed rounds are shown to the participant. This provides participants with instant feedback on their decisions and performance in the experiment.

A.2 Instructions to Clients

General Overview

As an IT executive of a firm, you need to acquire five specific software applications for your firm. The software applications are built from components. Each application supports a different e-commerce business function. For example, application A1(C1/C2/C3) is designed to provide a business solution, and it includes three reusable software components, C1, C2 and C3. Each application creates a given business value to your firm. For example, software application A1 creates 180 value. The value given is specific to your firm. It may be different for others.

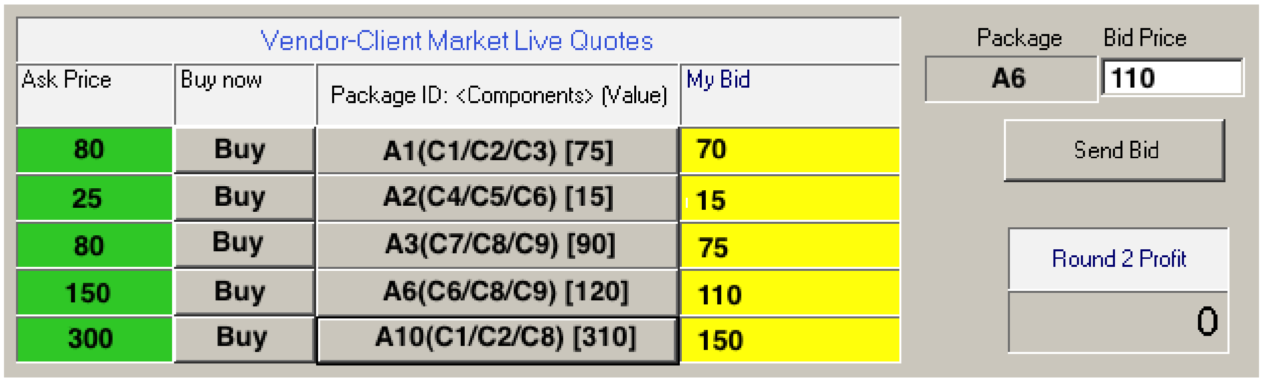
Your task is to acquire each of the 5 software applications for the best (lowest) possible price in the marketplace. You need to complete your purchases within a procurement period that is simulated as approximately five minutes. Your goal is to generate as much business value (profit) as possible. Your profits are equal to the values of the applications minus the amounts you paid for them.

Reuse and Recombination (reuse treatment only)

You can recombine the components you have available. E.g., if you have already purchased application A1(C1/C2/C3) and application A2(C4/C5/C6), you can extract at no additional cost (from A1 and A2) and recombine component C3 from A1 with components C4 and C6 from A2 to make A7.

Your Profit

The application profit you earn is equal to the value of the application bought minus the price you pay for it. For example, if application A1 is worth 180 and you pay 90, you will earn 180 - 90 = 90. For the whole round (procurement cycle) your earnings are equal to the values of all the applications you bought minus the total amount you pay for them. E.g., if you bought the 5 applications A1 to A5 for 140, 50, 50, 150, and 300 respectively, you earn a profit of (180+80+65+180+380) - (140+50+50+150+300) = 295.



Vendor-Client Market Live Quotes

Ask Price	Buy now	Package ID: <Components> (Value)	My Bid
80	Buy	A1(C1/C2/C3) [75]	70
25	Buy	A2(C4/C5/C6) [15]	15
80	Buy	A3(C7/C8/C9) [90]	75
150	Buy	A6(C6/C8/C9) [120]	110
300	Buy	A10(C1/C2/C8) [310]	150

Package	Bid Price
A6	110

Figure A2. Experiment interface for clients

Figure A2 shows an instance of the trading screen for clients. This client needs to acquire five software applications – A1, A2, A3, A6 and A10. The goal is to buy them in the market for less than the budget of the client (the sum of valuation prices that are shown in parentheses). Only those products for which an ask price is listed are currently available for purchase. In this case, all five applications are offered. Clients can enter new bids into the system at any time. For example, if the client increases his bid for A3 to 80, the client will obtain A3 and thus generate 10 profit (90 minus 80). For products A1, A2 and A6, however, their ask prices are currently above the client's valuation prices. The buyer could wait for ask prices to drop or raise his own bids.

A.3 Predicted equilibrium prices and quantities in the baseline case

Our study (like most market design studies) focuses on the market-level variables (total surplus, product variety) of the proposed reuse license and their impact on market performance, and we provide micro (firm)-level measurements (acquisition and production cost, profit) mainly to be able to offer a thorough explanation and interpretation of how the impact occurs under the two market settings. With an experimental study, we are able to investigate and advance the discussion on component reuse by “assembling a collection of people, putting them in the situation of interest, and seeing what they do” (Lucas 1986, page 421).

Table A1. Predicted Equilibrium Prices and Quantities in the Baseline Case

Product	Price	Quantity	Product	Price	Quantity
A6	120	2	***A6***	60	5
A4	170	3	***A15***	100-150	1
A13	270-280	1	***A10***	300-310	1
A2	50	7	***A1***	65	7
A14	100-200	1	***A9***	360	1

Table A1 shows the predicted equilibrium prices and quantities for this market environment. The theoretical prediction is based on the concept of static Nash Equilibrium with perfect information. That is, given these prices, none of the market participants can unilaterally improve his or her payoff by deviating from them, when all participants know all values and costs in the environment. Notice, that only 10 out of the 16 products are produced in the monopoly equilibrium. The other six products should not get made in this market environment, although there is some demand for them.

Total surplus (TS) is the sum of the profits of all market participants. In our case the projected total surplus is 1750. Assuming that the bilateral price negotiation in the niche markets with only a single consumer will result in a transaction price in the middle of the cost-value interval of the profitable products, we can also calculate the distribution of surplus between buyers and sellers. Total consumer surplus (TCS) is then 1200, and total producer surplus (TPS) is 550. The projected deadweight loss due to monopoly power is 420, which is the total unrealized consumer value due to monopoly power. Because we use a continuous double-auction as the institution of exchange we can expect some deviation from the theoretical equilibrium in the experimental setting.

Table A2. Pareto Optimal Prices and Quantities in the Reuse Treatment

Product	Price	Quantity
A2	50	10
A3	480	7
A1	65	9

In the reuse treatment, clients are allowed to modify a product after they buy it. In a sense, clients are given complete freedom to create other products that are either not offered on the market or are too expensive. Notice that all the products can be developed by modifying and combining component C1 to C9. Table A2 shows the Pareto optimal prices and quantities in the reuse treatment. Only three most important products should be developed and offered in this environment. Total consumer surplus is 1080, and total producer surplus is 3555.

Appendix B: Questionnaire – Open Ended Questions

1. Strategically what differences do you find between the first and the second market simulation? What are the implications for vendors and clients?

2. In your mind, how will the software industry as a whole be impacted if something like the second simulation is implemented in practice?

3. Based on your experience/knowledge on outsourcing and reuse methodology how would clients and vendors react to a market for component reuse rights?

4. Can you identify the potential benefits, risk and limitation of these kinds of markets (with component-based reuse strategies) for both vendors and customers?

5. Do you think it is possible, or likely, that such markets will be implemented in reality?

We conducted a qualitative analysis of the post-experiment questionnaires to further understand the value and challenge of our proposed reuse licensing model. It is exploratory in nature.

The open-ended questions are designed by the perception of the authors based on observations, literature review and experience. The answers and comments of the experiment participants were transformed and organized using excel tables, and then responses were categorized into to the opinions listed in Table 5. Individual responses were identified from the transcripts with references to the individual statement in order to ensure traceability (Yin, 2003). All the responses were then used to identify results and draw the final conclusions.

About the Authors

Richard Di Shang. Dr. Shang is an Assistant Professor of Business Analytics at the Coggin College of Business, University of North Florida. His research applies data analytics and experiments to discover insights in the contexts of IT services, information goods, and online marketplaces, and public health. Dr. Shang received his Ph.D. in Business (Information Systems) from Baruch College at the City University of New York.

Karl Reiner Lang. Dr. Lang is a Professor of Information Systems at Baruch College, CUNY, as well as the Executive Officer of the PhD Program in Business. Dr. Lang specializes in the areas of IT strategy, digital business, cognitive computing, globalization & technology, and issues relating to newly arising information society. He serves on the editorial boards of the journals Information & Management and Financial Innovation. Dr. Lang holds a PhD in Management Science from the University of Texas at Austin.

Roumen Vragov. Dr. Vragov is an Assistant Professor at Queensborough Community College and the founder of The Right Incentive LLC, a company engaged in IS & Economics consulting for businesses, government, and educational organizations. His interests include analysis, design, and development of electronic exchanges. Dr. Vragov received a PhD in Economics with a Minor in Management of Information Systems from the University of Arizona.