2004

Lean object-oriented software development

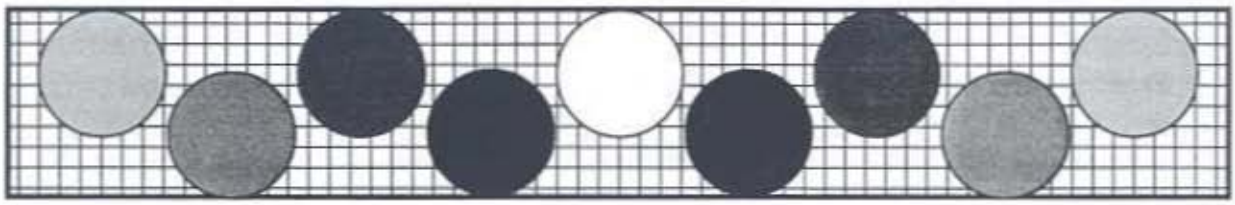
Jack Cook

Victoria Semouchtchak

Lean Object-Oriented Software Development

Jack Cook, *Rochester Institute of Technology*

Victoria Semouchtchak, *Rochester Institute of Technology*



Software failures can be dramatic, expensive, and catastrophic. The London Stock Exchange was developed eleven years late and 13,200% over budget (Corr 2002). A catastrophic software failure in February 1998 interrupted the New York Mercantile Exchange and phone service in several East Coast cities (NIST 2002). All industries need software development process improvement. Of 800 business technology managers responding to an *InformationWeek* survey, 97% reported problems with software bugs in the past year and nine out of 10 reported higher costs, lost revenue, or both as a result (Hayes 2002, p. 40). A study commissioned by the Department of Commerce's National Institute of Standards and Technology (NIST) estimated that software bugs cost the U.S. economy $59.5 billion annually, or approximately 0.6% of the gross domestic product (NIST 2002). Organizations are so dependent on software for virtually every aspect of business that any failure carries the threat of irreparable harm, especially for systems that operate around the clock (Hayes 2002). Development, production, distribution and after-sales support of products and services all depend on software.

Software systems should satisfy customer requirements at the agreed cost within the established timeframe. However, it is very difficult to estimate the cost and time needed to develop new and increasingly complex software applications. Hence, poorly understood projects are consistently late and over budget. The Standish Group (Corr 2002) found that 31% of all software development projects are cancelled due to defects. The average cost for "failed" projects is 189% of the original estimate, almost double what was budgeted. Average time for software development projects is even more than double the original estimate. Finally, an average project delivers only 61% of the specified functions and requirements. Only 16% of software development projects are successful. Process throughput times and the quality of finished goods would be unacceptable if manufacturing performed this poorly.

Software sales in the U.S. exceed $180 billion annually, supported by approximately 1.3 million software engineers and computer programmers (NIST 2002). Due to the difficulties of monitoring progress, defining the problem as well as alternative solutions, the degree of precision required, and the rapid pace of technological advances, many software developers have accepted that projects will be late and over budget. But software is more than code. It has a life cycle similar to manufactured products. Maintenance costs for software often exceed 60% of the total cost. Imagine the nightmare if cars were not designed to be maintained over the life of the product. Software is no different.

The high demand for software, shortage of skilled developers, limited financial resources, and the need for quality systems, requires the software development process be re-examined. Current economic conditions and increased competition make the case for change. Increased IT outsourcing will force developers to reconsider how they create systems. Better tools and management techniques can help if they reduce complexity and support the software project throughout its entire life cycle. Applying lean manufacturing techniques to software development is one alternative. The transformation of object-oriented software development (OOSD) from traditional to lean using techniques and examples from lean manufacturing is innovative and challenging.

OOSD clearly has parallels to manufacturing, and these similarities facilitate the application of lean techniques. Lean thinking is not a quick fix for software quality. The journey is long and

difficult and may not suit every company. This paper explains major lean techniques and their successful application to software development. It is aimed at decision-makers who initiate software development projects, the users assigned to assist developers, and the project leaders charged with the task of developing systems. The introductory sections discuss basic lean techniques, their benefits, and requirements. Subsequently, the need for lean thinking in the current software development environment is established. Finally, preliminary steps and recommendations for implementing lean procedures in application development are provided.

Lean Manufacturing techniques

The four principle concepts of lean are eliminate waste, standardize work, produce zero defects, and institute one-piece flow (Piszczalski 2000). The APICS online dictionary (apics.org) defines "lean" as a

> *philosophy of production that emphasizes the minimization of the amount of all the resources (including time) used in the various activities of the enterprise. It involves identifying and eliminating non-value-adding activities in design, production, supply chain management, and dealing with the customers. Lean producers employ teams of multi-skilled workers at all levels of the organization and use highly flexible, increasingly automated machines to produce volumes of products in potentially enormous variety. It contains a set of*

Figure 1. Lean Manufacturing Principles

principles and practices to reduce cost through the relentless removal of waste and through the simplification of all manufacturing and support processes.

Ultimately, lean improves productivity, quality, and flexibility

Lean requires strong support from all organizational levels. It requires changing the company's vision and scope as well as its management style. Management often resists change, insisting that benefits, potential savings, and quality improvements are clear. Employees, on the other hand, are frequently more concerned with the ease of implementation and operation of new processes and equipment than with overall company savings. Employees need to be included in the transformation and decision-making processes, resulting in "ownership" and an increased sense of vitality, necessity, and importance in the organization. Employee involvement empowers, enables, and encourages continuous feedback, cooperation, and suggestions. Eliminating waste (muda) necessitates employee involvement. Muda is everywhere — in the unnecessary complexity of a product, the labor used in production, the space taken up by the materials and the output, the defects, the raw materials, the inventory on hand caused by overproduction, the distance traveled by a product from materials to the final output, the time, and the skills.

- **Benefits**

Lean manufacturing encourages agility and responsiveness to customer demand, leading to greater profitability in a world where customer requirements are demanding and ever-changing. Ultimately, production costs are reduced by eliminating waste, decreasing defects, standardization, reduction of process variations, production control with a lot size that lowers overproduction, and lastly, continuous improvement to exceed customer expectations. In summary, lean practices are a set of valuable strategies and tools to advance current manufacturing processes and achieve maximum productivity with increased customer satisfaction.

- **Requirements**

Implementing lean successfully requires a clear understanding of lean principles (see Figure 1) and a methodical, step-by-step approach. Employees must be educated about the major lean theories and principles as well as trained to perform ad hoc problem solving, identify process waste and defects, and collaborate in multifunctional teams. An organization must implement statistical process control. Measurements recorded at strategically located points in production identify irregularities that require immediate attention. Moreover, employees must track their own performance and be informed concerning the company's progress in meeting its lean goals. Commitment of a substantial amount of time and money is required. Therefore, management needs to be aware that the transformation is not going to be immediate and cost beneficial right away. Most important, improperly implemented lean initiatives are expensive and involve complicated corrective action.

Drawing the Parallels

- **Current Software Development Environment**

Industry-Wide standards and company-specific uniform coding conventions. Commitment to standardizing the software development process is required. Only 24% of organizations implementing the Capability Maturity Model have reached level 3, and only about 6% have achieved level 5 (Hayes 2002). Third-party class libraries increase productivity and software quality by allowing developers to focus on unique aspects of an application, essentially ignoring standard infrastructure issues. Although third-party component markets exist (e.g., componentsource.com), issues that must be resolved before developers broaden their use include (1) certifying components and possibly their developers, (2) creating appropriate controls to prevent reverse engineering and subsequent reengineering, (3) instituting numerous pricing policies (e.g., outright purchase, pay-per-use, and time-expired use), and (4) devising suitable licensing agreements that document "appropriate" usage and developer liability in case of defective components among other things (Vitharana 2003). Defects would include hacker-prone, corrupt, or virus-infected components (Vitharana 2003).

Unnecessary, single-use or unused code. Currently, many systems contain poorly documented and inefficient code. Programmers, pressured by deadlines, tend not to produce reusable code since such code takes more time to initially develop. The absence of suitable incentives to

promote reuse of existing objects is also a common roadblock. Since reusability is a major way to cut costs and shorten product development time in the long run, management must establish such incentives.

Information encapsulation. OOSD incorporates the concept of information encapsulation, an important step from structured programming. Information encapsulation hides the nuts and bolts of an object from the user. The user utilizes an object's interface (e.g., a button, checkbox, or menu) as a means to an end without necessarily understanding its internal methods. Coded, functioning objects are unavailable to the user to edit.

- **Manufacturing and OOSD parallels**

Manufacturing parallels OOSD in many ways not exploited thus far. Although other software development methodologies exist, OOSD is increasingly used by many companies today. It allows programmers to work with higher levels of abstraction, making objects more reusable. OOSD encourages developers to create objects that collaborate to produce software that better models 'problem domains' than systems created with traditional methods. Think of objects as standardized parts in a manufacturing environment. Objects encapsulate their data and functionality to model real-world entities. Once fully tested, many software systems can incorporate an object. Object-oriented systems leverage the costs of developing objects through reuse and are easier to adapt and maintain. Object-oriented systems created from pre-tested objects take less time to develop. OOSD integrates and intertwines stages of software development more than traditional methodologies. For example, design can begin before analysis is complete. Also, user feedback can drive design changes even while programmers begin implementation. Finally, OOSD promotes reusability of prewritten code, although, in practice, reusability is more of a dream than a reality.

The adoption of lean practices into the object-oriented realm ultimately comes about with the standardization of programming methods. With lean manufacturing, less is best, but with information systems departments, more is better (Piszczalski 2000). With the introduction of lean manufacturing techniques into OOSD, programmers will have to produce less code as well as become much more proficient in the field of reusability. "Object-orientation teaches the principle of 'design for change': if we structure the code into modules we can minimize the impact of future changes, i.e., side effects on the structure as a whole" (Valerio; Cardino; Di Leo 2001, pg. 100). Implementing lean techniques in software development improves processes, lowers development costs, shortens development times, enhances reusability, increases maintainability, and improves documentation.

- **Inputs, processing, and outputs**

Manufacturing, at its most basic, is a combination of inputs, processing, and outputs. Raw materials, engineering specifications, and customer requirements are some inputs. Raw materials are transformed based on specifications and designs into real products that satisfy customer requirements. As a result, the outputs are the final products that meet or exceed customer expectations and satisfy engineering specifications.

In OOSD, the inputs are the system analysts' specifications, customer requirements, and programmers' coded objects. Customers are often present during processing, where processing solicits customer reactions to application prototypes. Objects interact to create the final software system. During the entire development stage, system analysts' specifications are updated to correspond to changing clients' requirements. Finally, the outputs of the software development process are similar to those of manufacturing. They include the final application matching customer specifications as well as the library of objects and documentation that will be used later for training and process standardization. Figure 2 summarizes the correlation of inputs, processing, and outputs for both manufacturing and OOSD.

Implementation of Lean Techniques to OOSD

This section applies the major lean techniques that so effectively increase productivity and efficiency in manufacturing to OOSD. Lean software may be the solution to minimizing defects, reducing numerous occurrences of waste, and maximizing efficiency and overall improvement of software quality.

- **Preliminary Steps**

Determine reasons to go lean. First, before implementation, a company must answer the

Figure 2. Process Inputs, Throughputs, and Outputs

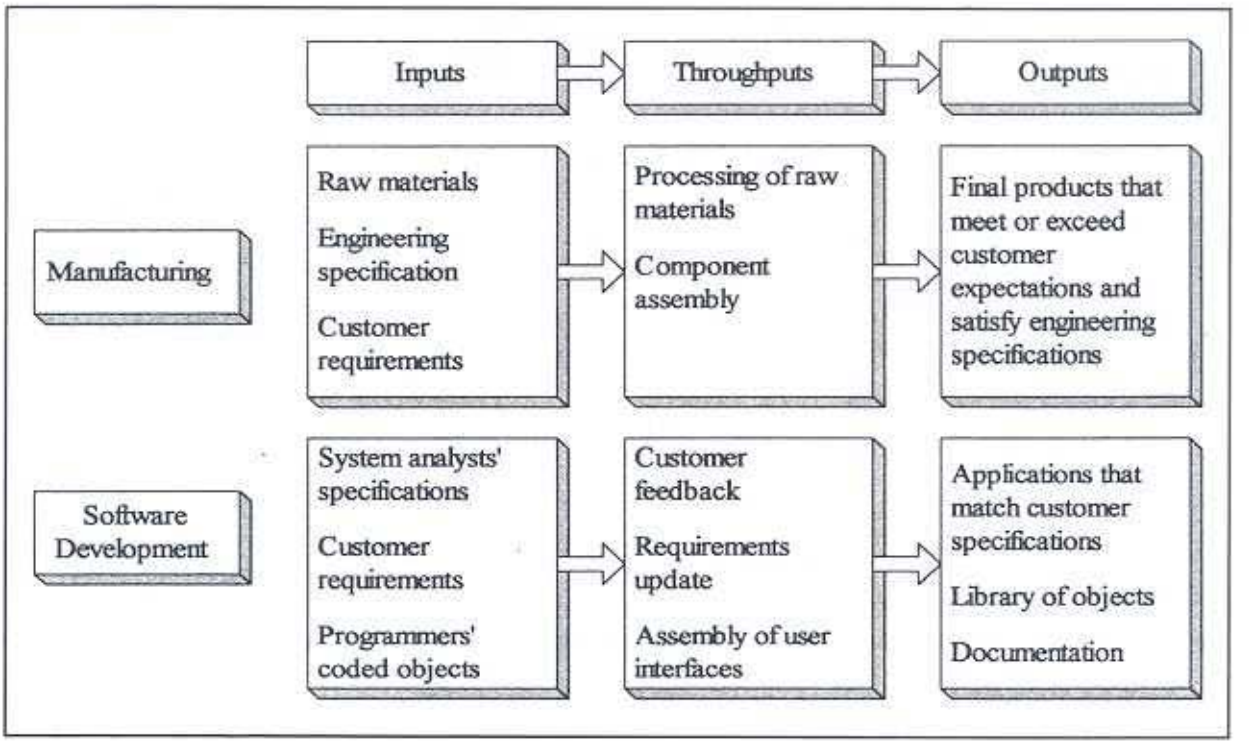


Table 1. Impact of Functional and Relationship Quality on Clients

		Functional Quality: Low	Functional Quality: High
Relationship Quality	High	Client sees opportunity to improve	Client provides repeat business and referrals
	Low	Client terminates project	Client provides no repeat business after project completed

question "Why pursue lean?" A lean transformation is not a quick fix. It is expensive, requires hard work, and creative thinking about how to become lean and persistence. However, companies have little choice but to create quality software or else face the possibility that poorly designed systems will result in lawsuits from customers, government agencies, or shareholders. Two general types of quality are key: functional and relationship.

Functional quality is more than the absence of syntactical and semantic bugs. Satisfaction is driven by the presence of features that fulfill needs. Quality depends on how well the system fulfills users' needs (i.e., functional quality) and the quality of the relationship with the service provider (i.e., relationship quality). In addition to functional quality, both internal and external clients "attach as much if not more importance to the relationship they have with

service providers than they do to the functional quality of the deliverables." (Russell and Chatterjee, 2003, p. 86) See Table 1 for the impact of functional and relationship quality on clients (adapted from Russell and Chatterjee 2003). Going lean allows companies to devote more resources to functional and relationship quality.

In addition to determining which quadrant with respect to functional and relationship quality a company is in, a company needs to determine which of the following four scenarios of applying lean best applies to their current situation:

A. Many benefits with little or no negative repercussions
B. Some benefits as well as minor negative repercussions
C. Few benefits and major negative repercussions
D. Many benefits but major negative repercussions

Organizations that fit into A and B are good candidates for applying lean techniques. Those in B probably have underestimated the benefits. Start-ups and companies with limited operating funds falling under scenario C have little spare capital for education and training and lack the time to research and develop appropriate implementation plans or resources needed to create a library of reusable objects. Large companies as well may be unwilling to invest the necessary capital into a lean transformation. First, it may not be cost effective to change a functioning, although inefficient, system. Second, employees may not be receptive and committed to the transformation process, sentencing it to failure. Companies in scenario D should work to reduce the negative repercussions, moving the company toward scenario A and ultimately enabling it to pursue a lean transformation. Once the decision has been made to pursue lean, developers must be educated, trained, and empowered.

Educate and train developers. Educating developers, project leaders, and users in change management, dealing with resistance and problem employees, communication and feedback skills, goal-setting and leadership, and management principles for a collaborative environment are all desirable first steps. Specific lean conversion training for developers should include value stream mapping and analysis — how to create current, ideal, and future value stream maps. Developers should be educated concerning what constitutes inventory in a software development environment. For example, every file, script, Web page, object, and design document represents capital tied up in the form of labor (i.e., inventory). Just as in manufacturing, inventory levels should be lowered over time. Developers need to define metrics such as the number of usable objects transferred out of a project as a percentage of the total number created or milestone due-date performance (i.e., percentage completed on or before due date). Measuring the trend in inventory reduction should be emphasized rather than setting specific absolute goals. "Five S" training should be mandatory for everyone. The five Ss are (Tomas 2002):

- Sort — classify tools, programs, and procedures into necessary and unnecessary, eliminating the former.
- Straighten — designate a place for everything and ensure everything is in its place.
- Shine — clean workplace, maintain its appearance, and use preventive measures to keep it clean.
- Standardize — make *sort*, *straighten*, and *shine* standard practice.
- Sustain — make a habit of properly maintaining and following standard practices

Training in a variety of areas allows the assembly of multi-functional teams capable of performing assorted software development tasks. When developers understand the diverse nature of business, it is easier for them to interact and communicate with users, thus increasing productivity. Education in lean principles encourages developers to recognize problems and potential waste independently without management's tedious and constant involvement.

Empower developers. Management must include all employees from the beginning in the lean transformation process to minimize resistance and prevent developers from feeling lean is being forced on them. Circulate information about upcoming changes regularly throughout the entire organization. Clearly state and post company statistics, policies, and agendas. Institute an easy and effective suggestion system to gather employee feedback and focus on quality improvement. Train workers to make decisions and then give them enough authority to use their individual or team problem-solving skills to establish and execute effective solutions. For example, value stream mapping will do little for

your business without action. If developers know what should be done to solve new problems autonomously, then project leaders can concentrate on investigating new techniques and strategies to increase productivity and efficiency.

Encourage cooperation among users, system analysts, and programmers. Empowering employees not only increases job satisfaction, improves the working environment, and increases interdepartmental cooperation, but also encourages teamwork, independent decision-making, and efficient responsibility delegation with shared accountability. Workers will not depend on management to make all the decisions. Allocating support without removing an individual's responsibility is not only achievable but also generates greater feelings of individual accomplishment and contribution. Cross-training in a variety of different business areas ensures that the majority of developers have the necessary knowledge and hands-on experience to cooperate with users. Choosing the proper person for a specific task is imperative. While the task is delegated to a particular individual, it is everyone's job to provide continuous performance feedback so that the person can track his or her progress. Key performance measurements should be reported weekly. Avoid too many metrics.

To assure continuous flow in software development, users, programmers, and system analysts must collaborate throughout the entire process. Effective communication and feedback about customer requirements, application prototypes, design specification, and implementation details is crucial. Cooperation does not stop at effective communication. It should include collaborative idea generation and decision-making relevant to software development as well as continuous resource support. Finally, allocate adequate time to discuss software development processes and potential improvement strategies.

Apply persistence and commitment. Lean is a high-risk, high-return endeavor. Even when implementing lean techniques successfully, the benefits of reduced errors, lower costs, and shortened development or return on investments will take months and sometimes years to materialize. A company may need to spend millions first on training developers to reuse code, instituting a system that rewards reuse, and setting up a library to house and track software artifacts. Successfully implementing lean techniques requires the dedication of the entire organization. As a result, a company pursuing lean must have an implementation road map with clearly defined milestones and metrics to measure performance. Otherwise, lean will not be used to its full potential. Transform all aspects of the software development life cycle. Areas that originally might seem inconsequential can later be crucial. Inconsistently implementing lean will create more waste, rework, and recovery. Furthermore, it might discourage continued implementation of lean and prolong the payback period by requiring a higher financial resource allocation.

Customize the transformation process. Corporations wishing to implement lean cannot use another company's transformation process without adapting it to their culture, infrastructure, and environment. Customization involves adjusting lean principles and implementation strategies to fit a particular organization's vision, mission, and processes. Commitment, customization, action, and accountability are key to success.

- **Recommendations**

Three general recommendations should enable the transition to lean to be valuable, efficient, and straightforward. As mentioned, lean principles include eliminating waste, standardizing work, producing zero defects, and continually improving. The following sub-sections address these lean principles.

Eliminate waste. A company must examine its entire system, documenting it in full. Problem areas, especially wasteful processes, are identified and removed. One area where manufacturing and software development are very similar is with respect to muda. Therefore, if lean techniques help eliminate waste in manufacturing, they have the potential to substantially decrease or eliminate waste in software development.

Value stream mapping helps identify waste. Carefully investigate every process to determine if it adds value. Categorize activities as value-added, nonvalue-added, and nonvalue-added but required. Five S programs provide the organizational discipline for waste reduction and correction (Tomas 2002). The following strategies are suggested:

Table 2. Waste Correlations in Manufacturing and Software Development

Area of waste	Manufacturing	Software Development
Unnecessary complexity	Product features, some never used by the customer, or documentation that makes it harder to understand and use.	Application objects and methods not needed to perform the specified task or the elements and added features that will never be used by the customer.
Labor used in production	At times production is subdivided into overly specific sections that require too many people performing similar or same tasks.	Development teams have an unnecessary large number of members carrying out redundant functions and reworking each other's code and ideas.
Space taken up by materials	Inventory might exceed the needed amount, taking valuable space in a factory. Even though the products or parts are not being sold or used, the company still pays for the space.	The application, especially with unnecessary features and inefficient coding, is unreasonably large. Therefore, a customer might not be able to utilize the application due to hard drive space restrictions.
Defects	Product defects might exceed the accepted norm, often 0-1%.	Excessive number of program defects due to the lack of testing and debugging, or the lack of acceptable defect rates.
Raw materials	Unused inventory of materials on hand because of incorrect order numbers or supplier delivery schedule. Just-in-time delivery not implemented.	Unreasonable customer requirements with unattainable details as well as the size and complexity of objects.
Inventory on hand as a result of overproduction	Pull production might not be instituted in the company and, therefore, too many parts are being produced without being used by the assembly department. At times, one component piles up while the team waits for another part.	The programmers are writing code and features that the customer might not be willing to pay for due to poor communication with system analysts. Therefore, you have too much code or objects that will not be implemented.
Travel distance of a product	Components might be too far away from each other so that the travel time is wasted.	The virtual travel of the code can be excessive. Also, departments or teams might be located too far from each other whether different floors or buildings. This does not allow a clear communication flow and leads to misunderstanding.
Time	In manufacturing, this involves both production and delivery time.	In object-oriented software development, this deals with the time it takes for the development of individual objects and design specifications, as well as the time to deliver the final system.
Skills	Each factory needs unskilled and skilled labor to perform various tasks, however, at times the balance is not achieved properly and, therefore, some skills are wasted.	Overqualified employees might be performing nonvalue-adding tasks. Under-qualified personnel can be in inappropriate positions.

- Organize the digital workspace to ensure developers are not distracted by clutter and can quickly locate software artifacts.
- Eliminate cubicles to allow easier information flow and communication between developers. Team members should be located close to each other. This decreases the waste associated with information flow and product or employee travel.
- Compile a predefined list of components and capabilities to allow customers to select features, similar to buying a car. A standard package should be available and add-ons chosen separately. This not only clarifies user requirements but also eliminates unnecessary and unused features in an application.

Waste elimination smoothes process flow before it improves individual operations (Tomas 2002). Once waste is eliminated, increase process flow velocity and standardize work practices.

Focus on standardization, not creativity. The goal of making programs readable and easily maintainable is laudable. However, trying to get every programmer on the same page is nearly impossible. The training costs to do so may outweigh the benefits with no guarantee of success. Some believe focusing on standardization rather than creativity is an argument against the introduction of lean to software development. Automation and reuse may stifle the creativity of programmers and quite possibly impede their ability to solve problems. It also constrains a programmer's flexibility. However, standardization results reduces process variation and defects, increasing the quality of systems and reusability of objects.

Standardization means the identical sequence, material, tools, education and training, orientation, and steps to complete a process. It suggests that a company should inaugurate a set of standards, rules, and procedures that developers follow consistently. These would include general design principles like object encapsulation, information hiding, hierarchical structure utilization, polymorphism, and, finally, inheritance. These are just some of the object-oriented design principles. Standardization also applies to waste reduction during all development stages and to statistical quality control procedures where the same measurements are evaluated for all applications, and to the processes themselves. An organization should encourage standardized coding procedures, provide methods to analyze requirements, and implement design principles.

A company's software development standards should be consistent with industry standards. Lean techniques cannot be implemented effectively if the company's software processes are always changing. Development process stability is important. Newly instituted standards must be used on a regular basis for lean execution to be productive. Standard practices guide and effectively constrain each programmer's object design and implementation, as well as the overall software development process. This increases reusability of objects, code blocks, methods, and functions.

Produce zero defects and continually improve. Standardized testing tools, suites, scripts, reference and test data, reference implementation, and metrics that have undergone a rigorous certification process can go a long way toward improving software quality (NIST 2002). Standardized testing also provides a consistent way to determine when to stop testing (NIST 2002). Unfortunately, not all objects are unit-tested upon creation. Some programmers defer testing until integration or systems testing. The rationale is supposedly to save debugging time during the development stage. However, research shows it takes much longer to debug and test if unit testing is inadequate or nonexistent, since it is about five times harder to detect defects and errors, especially if they are the result of incorrectly translating user requirements. Therefore, if testing is properly conducted, the resulting system will have substantially fewer errors and deficiencies.

Lean is a daily improvement process, and continuous improvement is the most important implementation principle. Even if transformation from traditional to lean software development is successful, an organization cannot stop searching for better quality and process efficiency. Continuous improvement encourages ceaseless system, software, and hardware updates to keep up with changing customer requirements. Exceeding customer expectations and raising the quality bar for competitors is essential for survival. Developers should think lean, produce lean, and recognize waste. Training developers in new technologies, processes, and principles will help them continuously improve.

Conclusion

Without a doubt, unless certain transformations are undertaken and new approaches developed, the problems plaguing software development will persist. Reforms and standards must be instituted. One solution is to incorporate lean techniques into OOSD. Before manufacturing transformed to lean, it had similar problems. Lean techniques significantly decreased the number of errors and encouraged a more effective and productive manufacturing system with just-in-time inventory, less waste, and better resource management.

There are numerous parallels between manufacturing and software development. Implementing lean techniques drastically reduces the

O'Conner, R. (2002). Plug the Expatriate Knowledge Drain. *HR Magazine, 47*(10), 101–107.

Paik, Y., Segaud, B., and Malinowski, C. (2002). How to improve repatriation management: Are motivations and expectations congruent between the company and expatriates. *International Journal of Manpower, 23*(7), 635–648.

Poe, A. (2000). Welcome back. *HR Magazine, 45*(3), 94–105.

Solomon, C. M. (2001). Global HR: Repatriation planning. *Workforce,* 22–23.

Suutari, V., and Brewster, C. (2001). Expatriate management practices and perceived relevance: Evidence from Finnish expatriates. *Personnel Review, 30*(5/6), 554—577.

Taylor, S. J, and Bogdan, R. C. (1984). *Introduction to qualitative research: The search for meanings.* New York: John Wiley & Sons.

number of defects waste, decreases product development time, increases quality, and helps standardize software development within a firm. However, to utilize lean principles to their full potential, companies must undergo a difficult and time-consuming transformation process. Some lean initiatives are destined to fail. The success of implementing lean in software development depends on a company's internal processes, social structure, physical environment, and a reexamination of its mission with lean principles in mind. Upon successful implementation, object-oriented software development will uncover its true potential of producing reusable objects as well as high-quality, lower-cost applications that exceed customer expectations.

Dr. Cook, professor, speaker, author, and consultant, has made over 60 conference presentations and published numerous journal articles. His areas of expertise include electronic commerce, information systems, and production / operations management. Victoria Semouchtchak is completing her Bachelor of Science in Management Information Systems and working as an intern in VeriSign's Engineering Department.

REFERENCES

Corr, P. (2002). Software quality assurance, using *Standish Group Statistics* MIEEE (www.standishgroup.com). Belfast: Queens University. Retrieved February 22, 2004 from http://www.cs.qub.ac.uk/~P.Corr/ProfPrac/Software_QA.ppt#2

Hayes, M. (2002, May 20). Quality first. *InformationWeek, 889,* 38–54.

NIST. (2002, June 28). Software Errors Cost U.S. Economy $59.5 Billion Annually: NIST Assesses Technical Needs of Industry to Improve Software-Testing. Retrieved August 15, 2003, from http://www.nist.gov public_ affairs/releases/n02-10.htm.

Piszczalski, M. (2000, August). Lean vs. information systems. *Automotive Design & Production, 112*(8), 26–27.

Russell, B., Chatterjee, S. (2003, August). Relationship quality: The undervalued dimension of software quality. *Communications of the ACM, 46*(8), 85–89.

Tomas, S. (2002, July/August). A good fit. *APICS — The Performance Advantage,* 34-39.

Valerio, A., Cardino, G., and Di Leo, V. (2001). Improving software development practices through components. *27th Euromicro Conference Proceedings,* 97–103.

Vitharana, P. (2003, August). Risks and challenges of component-based software development. *Communications of the ACM,* 46(8), 67–72.