DOI: http://dx.doi.org/10.26483/ijarcs.v11i5.6644

Volume 11, No. 5, September-October 2020

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

SURVEY REPORT

Available Online at www.ijarcs.info

ECONOMIC IMPACT OF SOFTWARE PRODUCT LINE ENGINEERING METHOD– A SURVEY

Adekola Olubukola Daniel
Computer Science Department
Babcock University, Ilisan Remo, Ogun State,
Nigeria

Omotosho Olawale J.
Computer Science Department
Babcock University, Ilisan Remo, Ogun State
Nigeria

Olaniyan Oluwabunmi Omobolanle
Computer Science Department
Redeemer University, Ede, Osun State
Nigeria

Abstract: Software Engineering has to do with the art of design, development and maintenance of software products that adequately meet user's need. The key market requirements this field tries to meet are basically time to deliver, product cost and quality. With these goals in mind, software engineering researches had experienced rigorous changes in time and in space especially in the area of "software re-use". Software Product Line Engineering (SPLE) leverages on building reusable components to achieve massive re-use. It is about designing systems for, and with reuse. In traditional software engineering, requirements and software architectures are engineered based on individual product alone but a product line approach requires the software expert to do same for a family of related products. Therefore, common assets are built for these related products while variable assets are also discovered which will lead to production of each specific product. This process, as it were, does not come cheap at first. There are surrounding economic, social and other consequences.
This work proposes to survey the economic impact of adopting software product line engineering methods in software production. This will help software developers make sound business case as well as appropriate judgments in terms of decision making.

Keywords: Common assets, Market requirement, Software Engineering, Software Product Line Engineering, variable assets

I. INTRODUCTION

Product Line Engineering (PLE) was born to increase economy of scale. Originally, this concept used to be a dependable instrument to the manufacturing and engineering industries such as automobiles. But today, it has also become a virile tool for researchers and industrialists to work hand in hand in order to increase software quality (as regards time to deliver and product itself and to reduce software development costs. Hence, as this concept burrows into the software industries we are ushered into what is described as Software Product Line Engineering (SPLE).

Traditional software engineering is geared towards building an individual product alone while Systems and software product line engineering is the engineering of a portfolio of related products using a shared set of engineering assets and an efficient means of production. The implication here is that assets are engineered to be shared across your product line. It is a move away from traditional product-centric development patterns.

Two important aspects in product line include identifying product ***commonality*** and deriving product ***variability.*** That is, Systems are described by features they have in common (commonality or the core assets) and those that separate them (variable assets or variability).

In simple language, designers employing this technique have reuse as first thing in mind [1]. Product line proactively combines development for re-use and development with re-use. Therefore, a software product line model describes a set of products in the same domain (family of products) instead of a single software system.

In traditional software engineering, software architecture is evaluated with respect to the requirements of the individual product alone. Whereas a product line approach requires the software expert to consider requirements for a family of related / similar systems and the relationship between those requirements [2].

With the conventional software engineering approaches, there are definitely some measures of reuse among the products but it's usually not systematic. Going along with the usual programming methods that leverage on reuse of subroutines, modules, objects and component based systems where these artifacts are copied from one another for reuse, things do not still scale very well. Software product line dwells primarily on planned reuse. This is familiarly called opportunistic reusability during software development. It seems to become a novel pace ahead in the field of components reusability. Software product lines characterize an original and emergent concept in software engineering. This approach is based on a development process including both developments for reuse and with reuse [3]. Indeed, a new product is not actually executed but more integrated in a Product Line PL, adding new necessary components,

reusing common assets, and using preplanned variation mechanisms as inheritance.

On the economic impact of SPLE, commonly, large and stabilized companies are the ones that opt for SPLE methods because of the expenses involved. This is because they are the ones who can easily support important initial investment and wait for a long term return on investment. Whereas, small firms are not usually enduring or patient enough to go the miles required and pay the initial dues.

When a company develops multiple products in the same domain, it benefits from organizing its software development activity as a product line. A product-line provides a platform (also known as a core asset base) shared by a set of related products that are developed by an organization. The shared platform identifies points of commonality and variation. Products are created on top of the platform by reusing its core assets, while reducing the effort that goes towards developing assets that are unique to the product. The motivation for a product line is reducing the cost of developing new products while increasing their quality and reducing the time to market. Product line approach helps to manage product diversity and reuse more systematically. In other words, products built using a product line approach will share a common base, which allows a company to manage customer-specific variations more systematically.

It is often observed that somewhere between 50% and 90% of development effort is spent on creating software that does not differentiate a company from its competitors. Only the remainder differentiates a company from its competitors [4]. The three major variables used to model the economics of product lines are distinctly time (taken to develop a product or a system), quality (of the system), and cost (of production and product).

II. HIGHLIGHT OF MAJOR SOFTWARE PRODUCT LINES DEVELOPMENT PROCESSES

The general process of product lines majorly hinges on reusability of requirements, architecture and components. The development processes is subdivided into two main phases - Domain Engineering and Application Engineering.

Domain Engineering: The primary target is identification and determination of the common features and the variability of a product line, the derivation of reference architecture and the realization of generic components and the associating quality assurance. Core assets are engineered through domain analysis, domain design and domain implementation processes.

Application Engineering: Focus on realization of the customized products, using the core assets and the specific variabilities peculiar to individual systems. A differentiation of product is established by systematic binding of variation points with the predefined variants. This phase is composed of three processes; application requirements, application design and application coding.

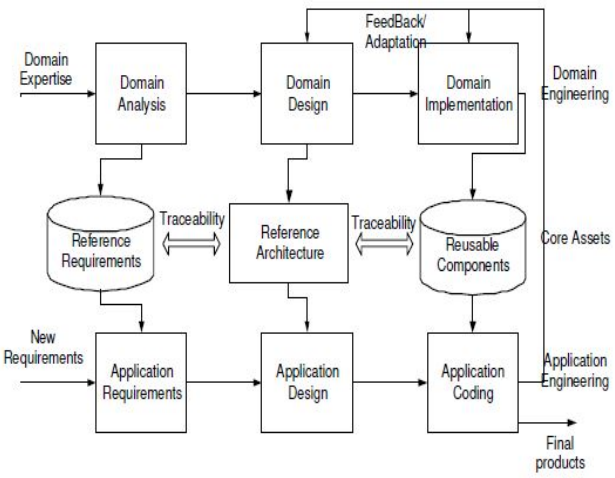

Figure.1: Software product lines process [5]

Domain analysis produces a set *of reference requirements* reusable in defining an *application requirements* and integrating *new requirements*. The *domain design* defines a *reference architecture* used to develop applications. And the *domain implementation* generates *reusable components* used in optimizing *applications coding* time. A *back and forth traceability* is established between the reference requirements, the reference architecture and the reusable components to promote product line changes and updates management. *A feedback /adaptation process* is used at the *application engineering* level to revise the domain design and the domain implementation [3].

III. MOTIVATIONS FOR PRODUCT LINE ENGINEERING

The following depicts the key motivations for developing software through the application of product line engineering approach.

1. **Reduction in Development Costs:** Embarking on all-inclusive engineering practices as product line must have an underlying viable economic justification. Reduction of development cost is a vital motive for bringing in product line engineering. Artifacts derived as core assets are massively are reused in several different kinds of systems which dramatically mean substantial reduction in cost of producing many individual systems.
2. **Products Quality Improvement:** As core assets are built from consideration of requirements from related products, they become standardized artifacts which are eventually used in the production of many products. Since it is not a matter of an individual system development, the artifacts are subjected to extensive quality assurance which necessitate a significantly higher chance of detecting faults and correcting them and consequently increasing the quality of all products.
3. **Boost in Time to Market/Deliver:** Time to market for traditional approach of single-product development is assumed to be roughly constant - which is basically the time to develop the product in question. Employing product line engineering initially has a longer time to market because multiple related products are put on the line while common artifacts are built. After this while, the time to market is considerably and drastically

reduced as many engineered or manufactured artifacts are now readily reusable for each new product.

4. **Reduction in Maintenance Effort:** This is synonymous to what is obtainable in object-orientation methodology where features like inheritance ease code modification and extensibility such that a change or correction made at ***super class*** level naturally flows down to the ***sub classes.*** Whenever an artifact is modified for the purpose of error correction or quality improvement, the changes can be propagated to all products in which the artifact is used. This, of course, goes a long way in reducing maintenance effort. Hence, the same techniques that lead to massive reuse also lead to extensibility and maintainability.
5. **Improvement in Cost Estimation:** A software house can focus its marketing efforts on those products it could easily roll out within the product line. More so, estimating costs for products realized within the product line is relatively straightforward and does not include much challenge. Over and above, the SPLE approach provides a lofty basis for cost estimation.

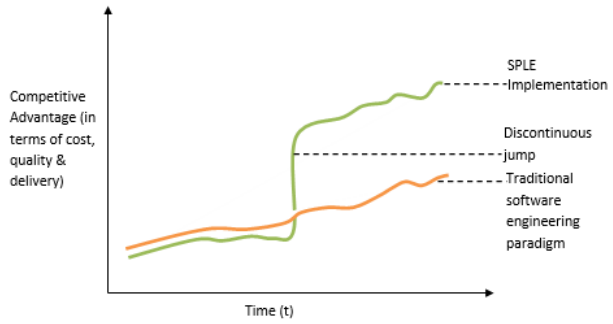

Figure 2: Sketch showing the competitive advantage of SPLE- starting off tends to be demanding and costly at the initial stage but on the long run there is a discontinuous jump as shown in the graph.

IV. CLASSIFICATION OF SOFTWARE PRODUCT LINE BENEFITS

Software product line paradigm comes with a variety of benefits classified into three reasonable types: Organizational benefits, software engineering benefits and business benefits.

1. **Organizational benefits:** benefits like a better domain's comprehension, facility to train people, high quality product and customer's trust.
2. **Software engineering benefits:** advantages such as all-round reusability counting from requirements and their components, better analysis of requirements, control of software quality, standardized coding and design patterns, removal of redundant implementations and complete reusable documentations.
3. **Business benefits:** a huge save in production, maintenance and test costs. Also, budget and time planning improvements are quite promising.

V. REVIEW OF SOME RELATED WORKS

Economics of Software Product Development Collectives [6]

This work retraced the evolution of software product development, how it is organized and then proposed imminent into the economic drive for combined or communal work between the developers, which involves relevant companies joining a software product development group. He identified three factors affecting the economics of collectives (level of contribution, number of members, and diversity of use), and then built up a model that links those factors to three economic outcomes (time, quality, and cost). The problem identified here is that this work did not consider the costs it will take to make changes to the organization for it to be more effective to create and sustain a software product line.

REARM: A Reuse-Based Economic Model for Software Reference Architectures [7]

REARM presented a practical economic model to perform cost-benefit analysis on the adoption of software reference architectures as a key asset for optimizing architectural decision-making.

This is also considered an economic model to translate expected data (i.e., metrics) into monetary terms which is used as a basis for analyzing the economic value of a reference architecture of which any organization that practices this will recognize a return on their investment within few years. The gap here is that the work did not address the effects of maintenance and evolution over time.

New Methods in Software Product Line practice: Examining the benefits of next-generation SPL Methods [5].

This research work had its major objective rested on varying tools and techniques for software development from focusing on developing individual products but rather a product family. The methodology adopted entails software mass customization (to eliminate labor-intensive application engineering), minimally invasive transitions (to eliminate the adoption barrier) and bounded combinatorics (to extends the scalability of product line portfolios. The sole benefit of the work so far include improvement in software re-use, increased time to market and above all standardized quality.

Incremental Return on Incremental Investment: Engenio's Transition to Software Product Line Practice [8]

This research work dwelt on making transition to software product line practice in order to keep pace with growing business demand for its products. Methodology employed include staging an incremental transition, then reinvesting the returns partially or fully to fuel the next incremental steps in the transition. The relevance of this is to avoid the typical upfront adoption barrier.

VI. COST FUNCTIONS ON SOFTWARE PRODUCT LINE ENGINEERING

In the past work done by Clements, P., et al (2005), cost and benefit functions were introduced to describe the constituent aspects of the overall economic impacts. This is to relate how much a particular software product line paradigm will cost an organization and to showcase the relative what benefits compared to building these products singly. The four basic cost functions introduced to compute estimates for economic effects include:

1. **Cost Of Organization, Corg( ):** function that shows how much it cost an organization to adopt the product line approach for its products. This cost can include reorganization, process improvement, training, and other essential organizational repairs [9].
2. **Cost Of Core Asset Base, Ccab( ):** function that shows how much it costs to build up a core asset base appropriate to satisfy a particular scope. The core asset base also includes non-asset base such as plans, schedules, budgets, the scope definition, and various kinds of documentation. It will also consider the costs of performing a commonality/variability analysis, defining the product line's scope, designing and then evaluating generic software architecture; and developing the software so designed [9].
3. **Cost Of Unique Parts, Cunique( ):** function that shows how much it costs to build up the unique parts of both software and hardware of a product that are not based on assets in the core asset base [9].
4. **Cost Of Reusing Core Assets, Creuse( ):** function that explains how much it costs to build a product reusing core assets from a core asset base [10].

Cost of Building Products Using Product Line

The following equation represents the cost of building a single product line containing **n** products.

Cost of building a product line = Corg( ) + Ccab( ) + Σ (Cunique (product$_1$) + Creuse (product$_1$)) [9].

Cost of Building Products In A Stand-Alone (Cprod)

Cprod is the cost of building a product in a stand-alone fashion. The cost of building **n** products independently is show in the following equation:

Cost of building *n* products = $\sum_{i=1}^{n}$ Cprod(product$_i$) [9].

The economic benefits for building *n* products implementing product line compared to building them independently could be expressed as:

Cost of Building **n** Products in a Stand-alone – Cost of building **n** product as a product line + Benefits achieved from product line approach [9].

Some tangible set of benefit functions include Research and Development investment, Productivity and quality, Long-run Time to market gain, Employees career development opportunities, etc.

VII. SUMMARY

Software product lines (SPL) methodology is a paradigm shift from the traditional software engineering method of producing software on the basis of a single product per time to manufacturing of families of products. The underlying motive is to achieve a better way of evolving software products that could give an organization competitive edge in the areas of meeting market requirements, time, cost and quality. SPLE leverages of maximizing reusability by building core assets as software artifacts which is the back-bone of the so-called industrial / massive reuse. Various products evolve by applying the variant components on the core artifacts. Adoption of this methodology does not come cheap at first. Issues like staff training and restructuring might have to be dealt with rigorously as well. In fact, employees who are rigid to change might be lost along the line. In all, the comparative advantages of adopting this approach are well-worth it on the overall. There tend to be a discontinuous jump in the benefits offered when the overall economic impact is evaluated. SPLE provision is far beyond the common practices in traditional software engineering where developers practice code reuse majorly by copying functions, modules or taking reusability advantages in Object-oriented design(implementation of inheritance, polymorphism etc.). It is about of reuse of artifacts that were generated purposely built for reusability as the original goal of the SPLE paradigm.

VIII. CONCLUSION

In the technology driven world we now live, the size and complexity of software systems together with critical time-to-market needs demand new software engineering approaches to software development. Notable among these approaches is the use of Software Product Line Engineering (SPLE) which is becoming widely studied and adopted in research and practice. The motivations behind SPLE is to systematically reuse knowledge and software elements when developing concrete software for new systems and thereby harvest potential savings through reduced cycle times, cost, risk and increased quality to help with the evolution of a set of systems and to achieve product standardization.

Although the adoption of an SPLE might have plenty of benefits for an organization, it also implies several challenges; among them is the need for an initial cost investment. SPLE goes a long way in attempting to eliminate copying of defects as the case is with the traditional software engineering approach (most times when a component is copied any existing defect is copied along). Moreover, in the case of error correction, the communication and coordination that has to occur for a portfolio of **n** products is proportional to **n** raised to the power of 2 ($\mathbf{n^2}$) in the traditional approach. Tracing to fix errors become more problematic. Consequently, this implies that delivering 5 products is like engineering 25 and so on. But with SPLE, engineers work on assets shared by the product family and things get done in good time achieving standardized quality product and eventual cost gain in terms of product development and product cost.

IX. REFERENCES

[1] Pohl, Klaus, Böckle Günter, & Frank van der Linden. (2005). Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Berlin, Heidelberg, New York

[2] Adekola O.D, & Awodele O. (2014), Enhancing Software Development through Software Product Line: Developing Product Family rather than Individual Products. International Journal of Advanced Studies in Computer Science and Engineering. IJASCSE, Vol. 3, No.1; ISSN : 2278 7917

[3] Patrick Heymans & Jean-Christophe Trigaux. (2003). Software Product Lines:State of the art, FUNDP - Equipe LIEL, Institut d'Informatique Rue Grandgagnage, 21 B - 5000 NAMUR (Belgique)

[4] Ali, M., Babar, M., Schmid, K. (2009). A comparative survey of economic models for software product lines. In: Software Engineering and Advanced Applications. pp. 275-278.

[5] Charles W. K. (2006). New Methods in Software Product Line practice. Examining the benefits of next-generation SPL Methods. Association of Computing Machinery, vol. 49, no. 12, 37 – 40

[6] Michael Weiss (2011), Economics of Software Product Development Collectives. Technology Innovation Management Review. October 2011: 13-18.

[7] Silverio M.F., Claudia A., Xavier F., Helena M.M., : REARM: A Reuse-Based Economic Model for Software Reference Architectures : GESSI Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain

[8] William A. H., Charles W. K., & Joseph G. M. (2006). Incremental Return on Incremental Investment: Engenio's Transition to Software Product Line Practice. Association of Computing Machinery, 2,13

[9] Clements, P., McGregor, J., Cohen, S. (2005). The structured intuitive model for product line economics (SIMPLE). Tech. rep., DTIC Document.

[10] Clements, P., and Northrop, L. (2003). A Framework for Software Product Line Practice - Version 4.1 [online]. Carnegie Mellon, Software Engineering Institute URL: http://www.sei.cmu.edu/plp/framework.html, Pittsburgh, USA.