INTERNATIONAL JOURNAL OF PURE AND APPLIED RESEARCH IN ENGINEERING AND TECHNOLOGY

A PATH FOR HORIZING YOUR INNOVATIVE WORK

AN INTRODUCTION TO SOFTWARE PRODUCT LINE: A VIABLE PARADIGM TO SOFTWARE DEVELOPMENT

NARENDER SINGH

Assistant Professor, Department of Computer Science & Applications, Chhaju Ram Memorial Jat College, Hisar - 125001, Haryana, India.

Accepted Date: 06/08/2017; Published Date: 01/09/2017

Abstract: - The objective of a software product line is to reduce the overall engineering effort required to produce a collection of similar systems by capitalizing on the commonality among the systems and by formally managing the variation among the systems. Hence, Software Product Line is an effective way to implement software production for mass customization.

Keywords: Software product line, software reuse, object-oriented programming.

Corresponding Author: MR. NARENDER SINGH

PAPER-QR CODE

Access Online On:

www.ijpret.com

How to Cite This Article:

Narender Singh, IJPRET, 2017; Volume 6 (1): 12-20

INTRODUCTION

In today's industrial and commercial environment all businesses are getting tougher competition and the customers' expectations are continuously increasing due to advancements in technology at an unpredictable higher growth rate. Rapid and constant changes that are very common to the present business environments affect not only business itself, but also the production scenarios. Software is perhaps the most fragile piece of business entity in this digitized business environment, where make or break occurs so rapidly that organizations are not usually left behind with sufficient time to respond quickly enough for survival. Software organizations are continuously adopting innovation and improvement plans in major areas of business operations like technology, administration, process, and product development. Their major concern is effective utilization of software assets, thus reducing considerably, development time and cost of software products to capture market segments.

Programmers have always reused sections of code, templates, functions, and procedures. Software reuse as a recognized area of study in software engineering, however, dates only from 1968 when Douglas Mcllroy of Bell Laboratories proposed basing the software industry on reusable components.

Jacobson [1] defines software reuse as *"Systematic software reuse is the purposeful creation, management, support, and reuse of assets"*. An asset could be any software artifact such as requirements specification, design, document, and implementation code. Perhaps the most well known reusable asset is code. Code reuse is the idea that a partial or complete computer program written at one time can be, should be, or is being used in another program written at a later time. The reuse of programming code is a common technique, which attempts to save time and energy by reducing redundant work.

The main purpose with software reuse is to improve software quality and productivity, and thereby maximize a software development organizations profit [2]. The software engineering community has had long-standing high hopes that software reuse would be the answer to the *software crisis* [3]. A number of software reuse approaches have been presented over the years. One example of such an approach is the object oriented programming paradigm (OOP). OOP supports software reuse by techniques known as polymorphism, encapsulation and inheritance [4]. These techniques help the developer in producing highly modular and to some extent reusable code. Much research has also been done on reuse libraries [5] [2]. The basic idea of such traditional software reuse approaches is that organizations create repositories where the outputs of practically all development efforts are stored. These repositories would

typically contain components, modules and algorithms that developers are then urged to use. Unfortunately, it usually takes longer to find the desired functionality and adapting it to current needs than it would to build it anew [6]. The typical programmer solution to this problem is to ignore the legacy and build most of the software from scratch. Traditional techniques, which support so-called small-grained reuse, have therefore proved ineffective [7] when trying to address the software crisis in practice [6].

Another and more effective approach to software reuse is known as the "clone and own" approach [8]. When a new product project is initiated using this approach, the development team tries to find another product within the organization that resembles the current product as much as possible. The organization then copies (clones) all project artifacts, and modify and add whatever needed to launch the new product. This approach can yield considerable savings compared to developing all products from scratch. One drawback with the clone-and-own approach is however inefficient maintenance. When "cloning" an existing product to create a new product, its maintenance trajectory is split into two separate paths. This could lead to considerable additional maintenance costs for the common parts of the products over their lifespan.

Software product lines are about strategic reuse; this means that software product lines are as much about business practices as they are about technical practices [9]. Adopting a software product line approach requires a shift in mind for an organization. An organization must move from developing single products to developing product families. During analysis several related products are envisioned together and a design that can capture the requirements of the whole family must be developed. This means that everything is developed with reuse (within the family) in mind. This in turn implies that the effort needed for customization of the reusable assets to fit a new system is largely reduced compared to traditional reuse approaches. Another benefit of software product line development compared to traditional reuse is maintenance. In software product line development, products are built on a common platform and all products using the platform can share maintenance costs of the platform.

The field of software product lines is new enough to offer different definitions for similar concepts. The SEI has derived a definition from the hard goods industry that brings together the key intent of these sometimes-competing definitions. We define a product line as *"A group of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission"* [10].

In this paper, an attempt is made to highlight the concept of Software Product Line along with its objectives, process model, and benefits.

1. The Concept of Software Product Line

The origins of the software product line can be traced back to [11] program families. Parnas actually defers to Dijkstra's paper on structured programming (Dijkstra, 1970) as originating program families; however Parnas more fully investigates the concept. Program families are *"...sets of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members"*. Development of these families is done using either stepwise refinement or modular programming to postpone decisions. These decision points would be used to develop different versions of programs. [12] took the next step toward the modern software product line by introducing the concept of "domain analysis". Domain analysis is an "activity of identifying the objects and operations of a class of similar systems in a particular problem domain". Neighbours' domain analysis approach, the Draco method, relies on high-level problem domain-specific languages that can be manipulated into executable form. Essentially the domain specific language is used as the medium to capture the results of domain analysis.

2. Objectives of Software Product Lines

The most commonly stated objectives of a software product lines still relate to the goal of increasing reuse to decrease time and effort. Some examples of this viewpoint include the following:

- Product-line development separates the software development process into two separate life cycles: domain engineering, which aims to create reusable assets, and application engineering, which fields systems using those assets [13].
- Product line engineering aims to make large-scale reuse a commercial reality - to develop common or similar functionality explicitly reuse in multiple systems and thus to distribute cost and effort [14].
- The objective of a software product line is to reduce the overall engineering effort required to produce a collection of similar systems by capitalizing on the commonality among the systems and by formally managing the variation among the systems [15].
- The intention of the product line is large-grained reuse of components and straightforward inexpensive assembly of systems from components [16].

- In marketing however, product lines have different objectives related to improving branding, exploiting bundling, and in general, improving the perceived value of each product in the line. With software product lines, the critical nature of interoperability cause them to exhibit network effects, whereby the value of the overall line becomes highly dependent on the number of users of products in the line [17].

- A solely reuse-centred view of software product lines risks attempting to shape the expectations of the customer to better match the product line rather than shaping the product line to better match the customer. This is actually referred to in several cases, though many times not as a risk but rather a deliberate choice:

- Customers now see a benefit (lower cost, greater reliability) to fitting their requirements into the context of the product line [18].

- Reacting flexibly to customers' needs gains fewer benefits than actively steering these needs in a direction where they would be supported by their own product line [14].

- Of course the other consequence is that the customers must adjust their expectations to meet the reality of the product line offerings [16].

- This approach may lead to placing the firm in a position vulnerable to attack from a competitor who more accurately matches customer expectations, especially if cost and reliability are not the primary determining factors in whether they will choose the product [19].

- Instead of continually *reinventing the wheel*, or incorporating parts of old systems in an ad hoc manner, organizations following a product line approach can consolidate their key software assets within a high-quality, reusable software core, and concentrate their resources on adopting this core to meet the changing needs of the customer [20].

To address appropriate marketing objectives and general interoperability issues of software product lines, it is important to examine product characterization.

3. Essential Activities of Product Line

At its essence, fielding a product line involves core asset development or acquisition and product development or acquisition using the core assets [10]. Formally, product line practice is defined, as "The systematic use of software assets to modify, assemble, instantiate, or generate the multiple products those constitutes a product line."

Multiple verbs appear in this definition because there are a variety of ways in which the core assets can actually be used to create products. The production plan would elaborate on which technique is to be used for a given product line.

The activities of core asset and product development/acquisition can occur in either order, or (most commonly) in concert with each other. Core asset development/acquisition has been traditionally referred to as domain engineering. Product development/acquisition from core assets is often called application engineering. The entire process is staffed, orchestrated, tracked, and coordinated by management. Figure 1 (Essential Activities of Product Lines) illustrates this triad of essential activities. The iteration symbol at the centre represents the decision processes that coordinate the activities.

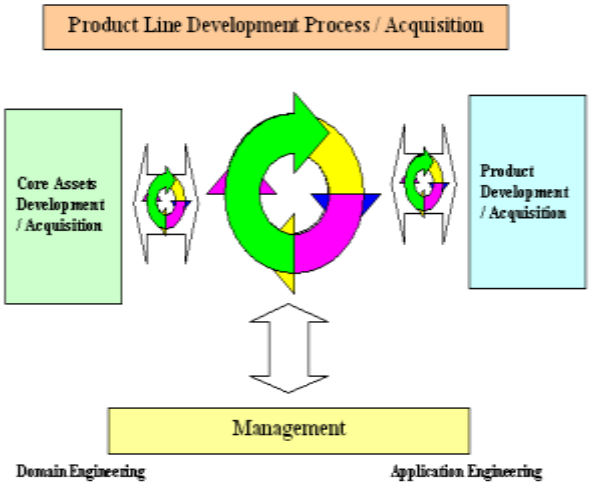

Figure 1: Essential Activities of Product Lines

The bi-directional arrows indicate not only that core assets are used to develop products, but also that revisions to core assets or even new core assets might, and most often do, evolve out of product development. The diagram does not specify which part of the diagram is entered first. In some contexts, already-existing products are mined for generic assets that are then migrated into a product line. At other times, the core assets may be developed or procured first in order to produce a set of products that is merely envisioned (i.e., planned) and does not yet exist.

There is a strong feedback loop between the *core assets* and *products*. Core assets are refreshed as new products are developed. The potential value of the core assets is realized through the number of products that are developed from them. As a result, the core assets are

typically made sufficiently generic by considering potential new products on the horizon. Finally, both the core asset and the product development/acquisition are themselves iterative, as illustrated in Figure 1.

Software product lines are not a panacea, but can have colossal impact if properly used. Before embarking on a product line approach, it is important to understand the business goals and to develop a business case for choosing product line practices. It is also very important to carefully scope the product line. A product line that potentially includes too many systems may need to support an unwieldy amount of variation.

In this section, we have discussed the high-level product line activities in terms of both development and acquisition of core assets and products. More detail can be found in A Framework for Software Product Line Practice [10].

4. Benefits of A Product Line Approach

A number of organizations have already gained order-of-magnitude improvements in efficiency, productivity, and quality through a product line approach. Often even more important than cost savings is the fact that product line practice enables an organization to more rapidly field products to satisfy operational needs. For the defence, this factor is key, allowing for the rapid deployment of new technologies and capabilities to support the war fighter. As Robert Harrison, Naval Systems Warfare Centre, succinctly stated, "*the right answer delivered late is the wrong answer*" [21].

Specific quantified benefits of software product line practice have been reported in workshops and case studies conducted by the Software Engineering Institute [21]. For example, the Swedish naval defence contractor, CelsiusTech, reported a reversal in the hardware-to-software cost ratio, from 35:65 to 60:20 that now favours the software as a result of their software product line approach for defence ship systems. Hewlett Packard has collected substantial metrics showing two to seven times cycle time improvements with product line practices. Motorola has shown a four times cycle time improvement with 80% reuse on their Flexworks pager product line Cummins Engine realized a decreased time for system build and integration from about one year to as little as three days in one case. Among other organizations that have shown efforts yielding equally dramatic product line results are: Thompson-CSF in air traffic control systems, Alltel in commercial bank systems, Ericsson, Nokia, Lucent, and AT&T in telecommunication systems, Buzzeo in college registration systems, Boeing in air flight software, and the National Reconnaissance Office in ground-based command and control systems for satellites.

5. CONCLUSION

Software product lines are rapidly emerging as a viable and important software development paradigm allowing companies to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and other business drivers. Software product line engineering can also enable rapid market entry and flexible response, and provide a capability for mass customization.

REFERENCES

1. Jacobson Ivar, Griss Martin, Jonsson Patrik. Software Reuse: Architecture Process and Organization for Business Success. Addison-Wesley Pub Co. May 22, 1997.
2. Frakes W, Kang K.: Software Reuse Research: Status and Future, IEEE Transactions on Software Engineering, vol. 31, no. 7, (2005), 529-536.
3. Gibbs W.: Software's Chronic Crisis, Scientific American 271, 3 (1994), 72-81.
4. Ezran M., Morisio M., Tully, C: Practical Software Reuse, Springer, (2002).
5. Mili A., Mili R., Mittermeir R.: A Survey of Reuse Libraries, Annals of Software Engineering, vol. 5, (1998), 349-414.
6. Bosch J.: Design & Use of Software Architectures, Addison-Wesley (2000).
7. Cusumano M.: The Software Factory: A Historical Interpretation, IEEE Software, vol. 6, no. 2 (1989), 23-30.
8. Clements P., Northrop L.: Software Product Lines, Practices and Patterns, Addison-Wesley (2001).
9. Northrop L.: SEI's Software Product Line Tenets, IEEE Software (2002), 32-40.
10. Clements, Paul & Northrop, Linda: A Framework for Structured Software Systems Ltd, (1999).
11. Parnas, D. L.: On the Design and Development of Program Families", lEEE Transactions on Software Engineering, (1976), 2(1), 1-9.
12. Neighbours, J.: Software Construction Using Components, PhD thesis, Department of Information and Computer Science University of California, lrvine, (1980).
13. Macala, R. R., L. D. Stuckey Jr., D. C. Gross: Managing Domain-Specific Product-Line Development, IEEE Sojivare, (1996), 13(3), 57-67.
14. Knauber, P., D. Muthig, K. Schmid, and T. Widen: Applying Product Line Concepts in Small and Mediumsized Companies, IEEE Soffivare, (2000), 17(5), 88-95.
15. Krueger, C. W.: Software Product Line Reuse in Practice, Proceedings of the Sd IEEE Symposium of Application-Specific Systems and Software Engineering Technology (ASSET '00), (2000).

16. Wood, W. G.: Government Product Lines", Software Product Lines: Experience and Research Directions, Kluwer Academic Publishers, (2000), 183-1 92.

17. Liebowitz, S. and S. Margolis: Network Externalities (Effects), The New Palgrave's Dictionary of Economics and the Law, MacMillan (1998).

18. Clements, P. C.: Successful Product Line Engineering Requires More Than Reuse, *Proceedings of the Eighth Workshop on Institutionalizing Software Reuse,* (1997).

19. Christensen, C. M.: Patterns in the Evolution of Product Competition", *Innovation and the General Manager*, Irwin McGraw-Hill (1999), 121-140.

20. Colin Atkinson, Joachim Bayer and Dirk Muthig. Component Based Product Line Development – The KobrA Approach. Fraunhofer Institute for Experimental Software Engineering, Germany. 1999.

21. Bergey, John, et al. DoD Product Line Practice Workshop Report (CMU/SEI-98-TR-007, ADA346252). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, (1998).