Emergence of Software Product Line

Divya Chadha
Maharishi Markandeshwar University
Mullana-Ambala, India.

Narender Singh
Maharishi Markandeshwar University
Mullana-Ambala, India.

ABSTRACT
Software Product line is emerging as an important paradigm and has provided competitive and various other benefits to organizations. This can help to overcome problems caused by resource shortages. The approach promotes asset re-use throughout the software life cycle, and facilitates product customization.

In this paper, we describe the emergence of software product line and its increasing scope. The first part of paper describes introduction to software engineering, software crisis and its principle. The other parts describe software reusability, types of reusable software assets, software product line, its processes, and its real-life applications. Related work and conclusion are discussed at last sections of the paper.

General Terms
Software Reusability, Software Product Lines.

Keywords
Software Engineering, Software Engineering Principles, Software Crisis, Software Reuse.

1. INTRODUCTION
Over the past number of years Software is evolving at a rapid pace. Information System is also becoming more complex and is being emerged as an important engineering discipline. In this scenario it is very important to implement high level concepts and techniques to develop this complex information system and this process is not straight forward but a bit tedious. Development of this complex system and large size projects always involve some unpredictable errors and violation of constraints and results into *software crisis* [1] or software runways. The main causes behind this are exceeding budget, late delivery of software, poor quality software, user requirements when not completely supported by the software, and when maintenance is difficult and in some cases where software is unreliable.

With evolution of software, constraints related to various technologies are also increasing. Wasserman [2] summarized the seven key changes that have impact on software engineering practice. It involves changing economics of computing-lower hardware & greater development cost, emergence of powerful desktop computing, extensive local and wide area network etc. He argued that any of these developments would have had a significant impact on the software development process. To overcome these constraints, the concept of software engineering introduced and it is evolving rapidly.

The conferences [3] sponsored by NATO in 1968 gave popular impetus to the term "*software engineering*". Since that time, the need for a more disciplined and integrated approach to software development has been increasingly recognized. Software Engineering is a profession that is dedicated to design, implement and modify software to improve its quality, reduce cost and make it maintainable and faster to build. It is a systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software. The IEEE Computer Society's Software Engineering Body of Knowledge defines software engineering as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software [1]. It aims at establishing sound principles to obtain economical software that is reliable and works efficiently on real machines. Sommerville [4] gave a new definition of software engineering as being concerned with theories, methods and tools for developing software.
Software engineering like other engineering disciplines is not constrained by materials governed by physical laws. A McDermid [5] point out that software engineering is to become a true discipline .It too must have an appropriate foundation in science and mathematics.
Evolution of software engineering has laid to emergence of new techniques and phases in software development process. The results of various phases were being analyzed by observing their results and software engineering has to evolve its principles as a result of it. The following are the software engineering principles that develop a foundation for efficient software process model.

Modularity [6] is expressed in terms of independent functions. A program is divided into various small modules or subprograms that are separately compiled. There is connection between these structures and modularity is based on constraints applicable on modules.

Abstraction [7] is another principle that only presents some necessary features or a brief description about the process.

Conformability [7] is a principle, which ensures that information needed to verify correctness has been explicitly stated.

Source code browsing [8] describes contribution of various factors to readability and maintainability of an application. Readability is increased by making a clear intention while writing code that is program by intention. The main idea is to make code understandable by user e.g. choosing identifier name that can be easily readable and recognized by programmers as well as other users and also matches that application.

Software metrics [9] is a principle that measures various aspects of software. The major advantage is that it provides an ease classify, compare and analyze the task. It enhances features of interest in software quantitatively.

Reverse engineering [10] category is during maintenance phase when engineers have to solve problems having poorly documented software system. This is helpful to represent system at higher level of abstraction.

Reuse [2] includes making use of various components again to improve usability of already available features and reduce cost.

2. SOFTWARE REUSABILITY

Software reuse is a major area that has significantly improved software productivity and quality. This is an important utility for software engineering. Software reuse [11] is the utilization of already developed components. These previously built components like code, architecture, design, programs can be used to implement or update software systems. A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time [12]. It is process of creating software components from predefined software systems. In this process software assets can be used as components that can be used again and again. Software can be developed for Reuse and with Reuse. It is very important to explicitly reuse components while satisfy all other properties. It is very important to develop an analytical method based on measures of software size in terms of complexity and functionality [13]. Reusability rising trends are shown in Fig 1. This shows that reusability usage is day by day increasing because of increased productivity and cost effectiveness. New techniques are implemented for software reuse with various software components which considers two main principles commonality and variability.

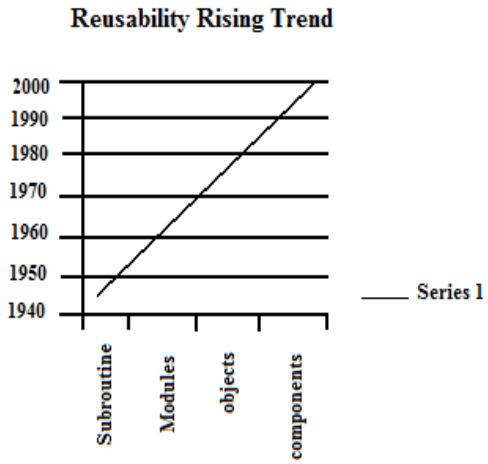


Fig 1. Reusability Rising Trend

3. TYPES OF REUSABLE SOFTWARE ASSETS

Reusability of software depends on how efficiently the software assets or components used. There are various components that can be reused like code or design. It is important to locate all these components [14] and efficiently stored them in a repository and their existence must be realized.

3.1 Software code

It can be reused. It can be used as a module again in another software development and add features of software asset reusability to software development.

3.2 Requirements

Requirements can also be used as assets. A template software requirement component can be implemented in various aspects for customer having same or different category of requirements by making few deviations in present model.

3.3 Architecture or design document

This is a representation of software process, or model it can be considered as architecture of whole development cycle or a particular module. An architecture or design is also considered to be an important software asset that can be reused.

3.4 Test plans

These are very necessary for checking working of software. It saves a lot of time if same test plan is used for one or more software development lifecycle.

3.5 Design decisions & Templates for any asset

These are the most expensive factor of design planning phase of software development. Its reuse is most cost effective for developers of software. Software Reuse results in reduced cost and investment. The reused components when collected must be properly located [15]. There must be a classification scheme for components and effective repositories for getting efficient core asset.

4. TYPES OF SOFTWARE REUSE

The Software reuse is classified on the basis of some measures. These parameters are described in [16].

4.1 Reuse over fungible systems

Software reuse has portability feature which means that new software can be created from already existing software and can be moved from one environment to another. It is adaptable to various platforms. Commonality is the important feature of software reuse that shows usage of a common component again and again for different versions.

4.2 Reuse over different domains

This is an important concept in which domain knowledge can be used to get new system form a past of previous one. Domain is first analyzed and then redesigning of domain is done according to usage and requirement in new production. It involves process of identifying domains, finding commonalities and then reusing them.

4.3 Reuse over time

Reuse over time has given concept of maintenance, evolution and versions. Maintenance of software is the need after some time of its delivery. Its maintenance is required each time. Evolution of software is its emergence with new features. Versions of software are released with new features. Example is *Software Product line* that reuses software with commonality and variability.

5. SOFTWARE PRODUCT LINE

Software Product Lines (SPL) [17] refers to engineering techniques where a new system is created by using common means of production. In this system, software assets can be reused to construct new software. For example to create a product line of similar products some common components or parts are assembled or configured to design various products. The study of software product lines includes collection of similar type of software system that has some commonality. The architecture and components are central to the set of core assets. The individual products in the software product line are built from these core assets according to a pre-defined production plan, sometimes called a reuse guide [18]. It arises repository of common software components that create software

artifacts when reuse is predicted [19].This can be achieved by using a commonality function or by using common assets again and managing it by implementing along with variability. SPL techniques can be used in various phases of software development life cycle. For example in software testing phase software product line can be used for using similar test plan components in various software development processes. Optimization of SPL and deviation of these techniques are also considered for efficient implementation.

6. SOFTWARE PRODUCT LINE PROCESS

Software product lines can be described in terms of four simple concepts that form the process of software product line, as illustrated below:

6.1 Core Assets

These assets can be composed in various ways to create products that can be configured. These software assets are the main ingredient for software product line because of their nature to be reused again. These assets are requirements, source code components, test cases, architecture, and documentation. The main feature of these assets is commonality of architecture for various product lines. To accommodate variation among the products, variation points are included and deviations are affixed among paths.

6.2 Product Decisions

This phase describes analysis of those variable features that can be reused. Based on particular software asset a product decision suitable for that particular product is taken with constraints and few deviations in asset or that reusable component product decision actually defines a product line. A decision model is then built

6.3 Production process

This is a phase in which product decision is used to select software asset that is used and its configuration is also determined. This consists of composing and configuring products from the software asset inputs. This is a means of composing products out of assets.

6.4 Process Outcome

It is the result of Software process and gives the range of product that can be produced by applying software product line. Software Product Output can best tell the results of implementing this methodology in software development lifecycle. The output of software product line that is produced by applying common software assets and various decisions can tell effectiveness of this technique.

Fig 2 shows that the key to the SPL approach is to identify the commonalities and variability of the product members. Commonality refers to core assets and Variability refers to decisions that can vary depending on a particular problem, requirement and component development. Fig 2 taken from [21] shows the usage of Core Assets with product development. These are amalgamated under the control of management to give unique output.

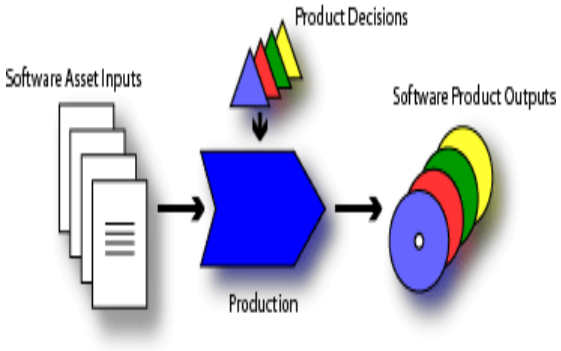
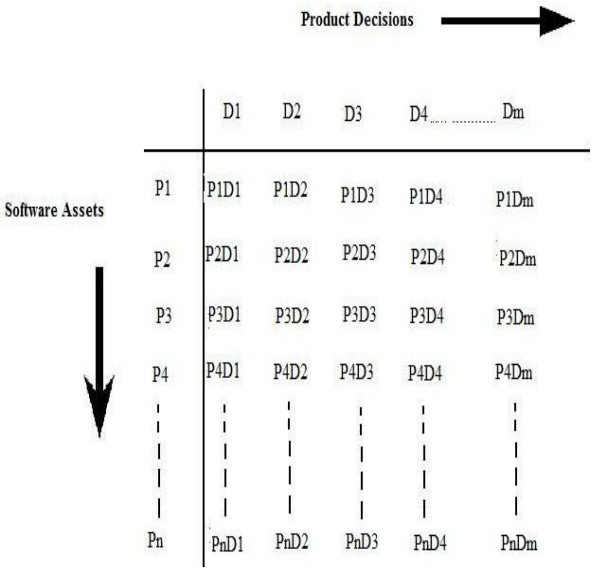


Fig 2. Basic Software Product Line Concepts

Software Product Line is a methodology that provides a variety of software products by accumulation of product decision and software asset. A matrix of software product line is presented below in Fig 3. It shows software assets at X-axis and product decision at Y-axis. And output is shown inside the matrix that is outcome of these two features. The results in every case vary by applying combination of one asset with a particular product decision.



Pi where i =1 to n is n number of Software Assets.
Dj where j=1 to m is m number of Product Decisions
PiDj => is combination of ith software asset and jth product decision.

Fig 3. Shows matrix of software assets and product decision.

7. REAL LIFE APPLICATIONS

There are various applications of this technology. Some applications are as following technology.

- Big Lever Software [20] researched and developed new software product line technology and was referred as Self Configuring Software Product line technology. They provided different point of views and levels of software reuse that effectively enhanced productivity of software product line.
- Advancement in Big Lever Software [21] was described with three new methods that provided very significant software product lines practices, software mass customization with configurations, minimally invasive transitions to software product line practice, and bounded combinatory.

- Philips Medical System, MRI Software [22] adopted Software Product line methods because of its variability and commonality feature. Philips provides several product lines in various technologies X-ray, ultrasound, magnetic resonance and computer tomography. It is very difficult process to test MRI scanner software and it heavily depend on selected hardware.
- Service-oriented applications [23] required implementation of Software Product Line. SPL practices can be leveraged to support the development of these applications and to promote the reusability of assets throughout the iterative and incremental development of software product families.
- Control Channel Toolkit (CCT) [24] built under the direction of the United States National Reconnaissance Office (NRO) is a software asset base for a software product line of ground-based spacecraft command and control systems came into existence. This product line base system had a set of reusable software components and tools to help integrate them into complete systems.
- Capability Maturity Model [25] proposed by Lawrence G. Jones Albert L. Soule is based on concepts and essential elements of Software Product Line. Various software engineering process groups have given their acceptance to this model. It focuses on continuous process improvement, process standardization and basic project management and covers various process areas like organizational process performance, verification, validation, technical solution, product integration, project monitoring and control.
- Metal Processing Line (MPL) [26] Implemented Software Product line for its machine software. In a MPL, each machine is controlled by its own software; there is a Programmable Logic Control (PLC). PLC is a program that controls and synchronizes all the machines. Supervision Control and Data Acquisition (SCDA) is another program for monitoring and visualizing the line. Program like PLC were made to attend customer requirement each time because of reusability concept MSI adopted SPLE to its MPL software and to its sub-domains also.
- A new version of Supplier Trading exchange (STX) [27] and its predecessor successfully implemented at Komatsu were developed with support of already available functionality of ERP product to provide a solution next to ERP products. So, an integrated procurement product was planned including direct materials purchasing, indirect materials purchasing, and e-procurement, e-invoicing to be integrated via one Supplier Trading exchange (STX). Some of the functionality was already available in existing products (e.g. direct materials purchasing in the ERP-product), while other functionality needed to be created.

8. RELATED WORK

Software Product line is a developing area in which researchers have proposed many techniques, approaches and methods. This section reviews the related work done by various researchers in this field. J. Bayer et al. [28] developed the Pulse (Product Line Software Engineering) methodology for the purpose of enabling the conception and deployment of software product lines within a large variety of enterprise contexts. The main consideration in this system was customizability and product centric feature with domain engineering. John D. McGregor [29] presented a different view of product line by implementing test related activities that can be used to form test process for product line organizations. The report described the test cases used to detect defects for software product line processes. L. M. Northrop [30] from Software Engineering Institute reviewed SPL activities and practice area. Some basic concepts are formulated in the paper. SPL approaches and some success stories have also been described in this paper. M. Matinlassi [31] performed comparative study for Software Product Line design architectures. An evaluation framework is introduced in this paper.This evaluation framework was introduced for comparison between five design methods *COPA, FAST, FORM, KobrA and QADA.* Yu Chen et al. [32] proposed to develop a simulator for cost-benefit analysis. The aim is to predict cost of selected software line product process. Process definition is an important stage to initiate the actual working. It is very necessary to determine cost, time and resource estimation. Simulator can be used to estimate these parameters for various product line processes. D. Fischbein et al. [33] proposed Modal Transition system and its semantics for behavior conformance for software product line architectures. They provided shortcomings of traditional behavior modeling formalisms such as Labelled Transition Systems and difference between existing and proposed approach. G. K. Hanssen and T. E. Fægri [34] presented a different perspective of Software Product Line by performing case study for company that has integrated practices from software product line engineering (SPLE) and agile software development (ASD). The paper described overview SPLE and ASD and how they are related. J. Zhang et al. [35] described role of aspects in Software Product Line from 3 phases. The effects of implementation of aspect-oriented software development are presented in this paper. The encapsulation and compatibility of variable features into architectural components is also presented. L. Chen et al. [36] presented the concept of variability management from different aspects. Virtual management is used to estimate the degree to which SPL is successful. They presented chronological background of different approaches and reviewed key issues regarding evolution of these approaches in VM research. F Mafi et al. [23] described how reusability of assets can be attained by developing service oriented applications by implementing software product line practice.

9. CONCLUSION

Software product line has become a very important area to be implemented in software development lifecycle because of its advantages. The most important is competitive benefit that is the main aim of every project. Other benefits are reduction in time to create and deploy a new product, number of defects per product, maintenance and cost per product. SPL increases total numbers of products that can be deployed and managed, improve competitive product value, profit margins, product quality, scalability of market and ability to hit market. Hence after analyzing the literature of this research topic it has been realized that at present and in coming time software product line is going to be emerged as a very effective methodology to reduce cost and increase efficiency of software systems and can efficiently use already stored software assets in repository. Application of SPL is felt necessary with rapid evolution of technology or software. It will be very necessary to make more and more use of already present assets and resources in near future to compete in an environment when there will be heavy scarcity of resources and limited cost constraint**.**

10. REFERENCES

[1] Pressman, R. 1992. Software Engineering: A Practitioner '.s Approach, 3'd edition, McGraw Hill.

[2] Wasserman, A. I. 1996.Toward a Discipline of Software Engineering, IEEE Software.

[3] Naur, P. and Randel, B. 1969. Software Engineering: Report of the Working Conference on Software

Engineering, Garmisch, Germany, October 1968. NATO Science Committee.

[4] Sommerville, I. 1996. Sofinre Engineering, Addison Wesley.

[5] McDennid, J. 1991. Software Engineer's Reference Book, Butterworth Heinemann.

[6] Parnas, D. L 1972. A Technique for Software Module Specification with Examples. Communications of the ACM (CACM), 15(5):330–336.

[7] Ross, D. T., Goodenough, J.B. and Irvine, C.A. 1975. Software Engineering: Process, Principles and Goals.

[8] Astels, D. 2003. Test-Driven Development – A Practical Guide, Prentice Hall.

[9] Conte, S. D., Dunsmore, H. E. and Shen, V. Y. 1986. Software Engineering Metrics, Benjamin Cummings, Software Metrics: A Rigorous Approach, Chapman and Hall, Fenton, N, E.

[10] Chikofsky, E. J. and Cross, J. H. 1990. Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software.

[11] Jones, T. C. 1984. Reusability in Programming: A Survey of the State of the Art.. IEEE Transactions on Software Engineering, vol. SE-10, no. 5.

[12] .Jalender, B., Govardhan, A. and Premchand, P. 2010. A Pragmatic Approach To Software Reuse. Journal of Theoretical and Applied Information Technology (JATIT), Vol 14 No 2, pp.87-96.

[13] Harsh, O. K. and Sajeev, A. S. M. 2006. Component based-Explicit Software Reuse, Engineering Letters, 13:1, EL_13_1_4, Advance online publication.

[14] Rao, C. V. G. and Niranjan, P. 2010. A Mock up tool for Software Component reuse repository. International Journal of Software Engineering and Applications, Vol 1, No.-2.

[15] Rao, C. V. G. and Niranjan, P. 2008.An Integrated Classification Scheme for Efficient Retrieval of Components. Journal of Computer Science 4 (10): 821-825, 2008 ISSN 1549-3636, Science Publications.

[16] http://alpha.fdu.edu/~levine/reuse_course/lesson1.html.

[17] www.softwareproductline.com

[18] Software Engineering Institute, The Product Line Practice (PLP) Initiative, Carnegie Mellon University, www.sei.cmu.edu/activities/plp/plp_init.html.

[19] Krueger, C. 1992. Software Reuse. ACM Computing Surveys. 24, 2 (June), 131-183.

[20] Krueger, C. W. 2000. Big Lever Software, Software Product Line Reuse in Practice. IEEE.

[21] Krueger, C. W. 2006. New Methods in Software Product Line Development.10th International Software Product Line Conference, IEEE.

[22] Jaring, M., Krikhaar, R. L. and Bosch, J. 2008.Modelling Variability and Testability Interaction in Software Product Line Engineering, Seventh International Conference on Composition-Based Software Systems, IEEE .

[23] Mafi, F., Mafi, S. and Mohsenzadeh, M. 2010. Service Composition in Service Oriented Product Line. International Journal on Computer Science and Engineering, Vol. 02, No.

09, 2859-2864.

[24] Clements, P., Cohen, S., Donohoe, P. and Northrope, L. 2001 Control Channel Toolkit: A Software Product Line Case Study. Technical report CMU/SEI-2001-TR-030.

[25] Jones, L. G. and Soule, A. L. 2002. Software Process Improvement and Product Line Practice: CMMI and the Framework for Software Product Line Practice.

[26] Sellier, D., Benguria, G. and Urchegui, G. 2007. Introducing Software Product Line Engineering for Metal Processing Lines in a Small to Medium Enterprise, IEEE.

[27] Versendaal, J. M. and Brinkkemper, S. 2003.Benefits and Success Factors of Buyer-Owned Electronic Trading Exchanges: Procurement at Komatsu America Corporation. Journal of Information Technology Cases and Applications, vol. 5, no. 4, 2003, pp. 39-52.

[28] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T. and DeBaud, J. 1999. PuLSE: A Methodology to Develop Software Product Lines. Best Paper Symposium on Software Reusability '99 (SSR'99), Los Angeles.

[29] McGregor, J. D. 2001. Testing a Software Product Line..

[30] Northrop, L. M.. 2002. Software Engineering Institute, SEI's Software Product Line Tenets, IEEE SOFTWARE J u l y / August 2002

[31] Matinlassi,, M. 2004. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA, IEEE.

[32] Chen, Y., Gannod, G. C. and Collofello, J. S. 2005.A Software Product Line Process Simulator.

[33] Fischbein, D., Uchitel, S. and Braberman, V. 2006. A foundation for behavioural conformance in software product line architectures. ACM New York, NY, USA.

[34] Hanssen, G. K. and Faegri, T. E. 2007. Process fusion: An industrial case study on agile software product line engineering.

[35] Zhang, J., Cai X. And Liu, G. 2008. The Role of Aspects in Software Product Lines.

[36] Chen, L., Babar M. A. and Ali, N. 2009. Variability management in software product lines: a systematic review, ACM USA.