A Systematic Approach for Configuration Management in Software Product Lines

K.L.S. Soujanya,*Member,IAENG* , A. Ananda Rao, *Member, IAENG*

Abstract— **Product lines achieve significant cost and effort reduction through large scale reuse of software product assets. Software Product Lines (SPL) consists of core assets and custom assets, which are shared among multiple products. Core assets, custom assets and products evolve independently. In single product the evolution of the product is in the time dimension, whereas the evolution of products in SPL is in both time and space dimension. Software Configuration Management (SCM) is a software engineering discipline that concerns the management of software evolution and change control. Available SCM systems are suitable for the single product evolution but inadequate for SPL systems. A software version management system is proposed to support product line engineering by supporting product line evolution, product derivation and change promulgation from core assets and custom assets to multiple products and vice versa. This approach supports twenty-three cases of amend promulgations.**

Keywords/phrases: **core assets, custom assets, software configuration management, software product line .**

I. INTRODUCTION

Over the years, software is developing at a fast pace, as it became inescapable and basic in our data based society so all software makers ought to expect obligation regarding its unwavering quality. Earlier “reliable software” meant error free software, but these days concerns like, adaptability and maintainability are equally vital. The need of the day is efficiency and optimization. This can be achieved by adopting software product lines. Software product line engineering is an approach that develops and maintains families of products while taking advantage of their common aspects and predicted variability. Despite the benefits of product lines many challenges remain. Product lines need to evolve and adapt continuously to stay competitive to meet the requirements of new customers, and to reflect changes in technologies [1]. However, the issue of product line evolution is hardly addressed by existing approaches and tool support is still not adequate.

Evolution in product lines is more challenging than in single systems due to the two inter-winning life-cycles of domain engineering and application engineering. In domain-engineering, reusable core assets are developed and the scope of the system is defined whereas in application engineering, the variability of the system is defined [2]. In 2004 ACM/IEEE Software Engineering Curriculum Guidelines list software evolution as one of ten key areas of software engineering education. Software is dynamic in nature. In 1970, Lehman formulated laws of software evolution, which says that a program to be used in a real world environment necessarily must change or becomes progressively less useful in the environment [3]. SCM encompasses the disciplines and techniques of initiating, evaluating and controlling change to software products during and after the development process. It emphasizes the importance of configuration control in managing software production [2]. The SPL poses a different problem to SCM in comparison to a single product software development. In a single product, the evolution of a product line is in the time dimension [18]. In SPL, products evolve independently of the components that are shared among the different products. Products and components have their own line of development. The evolution of the products are said to evolve in the space dimension while the evolution of the components are said to evolve in the time dimension [23].

Manuscript received December 2, 2014; revised January 22, 2015.
K.L.S.soujanya is Research Scholar with JNTUA, Ananthapuramu, Andhra Pradesh, India (phone: 09948143850; e-mail:souj47@gmail.com).
A.Ananda Rao is Director and professor with JNTUA, Ananthapuramu, Andhra Pradesh, India (e-mail: akepogu.rose@gmail.com).

The contribution of the paper is configuration identification of the software artifacts participating in the software product line. During the evolution the changes in the different core assets and modifications in the different custom assets are identified and stored, so that the versioned artifacts can be used as and when needed. When the changes in the assets are reflected in the products, it is termed as forward promulgation. The changes in the products are also reflected back to the assets data base. This is termed as rearward promulgation.

II. RELATED WORK

In the generic SCM model described by Clements and Northrop [4] core assets, custom assets and product instances are kept under configuration management. For each product instances under SCM, there is a corresponding product in use. Van gurp [5] proposes coupling variation modeling tools with subversion to support product derivation. He has yet to have a prototype to prove that the idea works. Kruger [6] describes an approach that uses conventional SCM tools. The core assets and product line instantiation infrastructure are kept under SCM.

Products are generated and are not kept under SCM. All changes are made in core assets and custom assets. In van Ommering [7], Kruger and van duresen[8] approaches, forward propagation is automatic. Since changes occur in the core assets, a product that uses the latest gets the new changes. Dependency among components and products is manually maintained. Molhodo SPL[9] is a prototype to solve the evolution problem at the configuration management level instead of at the source code at the programming language level. It is incomplete as it is only a research prototype.

III. PROPOSED WORK

In the existing system configuration management is applied on core assets and products. Product line software consists of domain engineering and application engineering. Domain-engineering defines the commonality and variability of assets. “Core assets – contains a set of domain specific but application independent component that can be adapted and reused in various related products”. “Custom assets – contains a set of application specific components.”

A product is a combination of core assets and custom assets. The software product line takes core and custom assets as an input and produces a product as an output. An individual product in the product line may share same core assets and different custom assets to adapt to the specific product requirement. A product can logically be considered as containing two parts - core part and custom part, which come from the core assets and custom assets respectively. Changes can propagate from the core assets project and custom assets project to products or from products to core assets project and custom assets project.

At the point when changes spread from core assets and custom assets to products is eluded as onward amend promulgation. A case of the onward amend promulgation is the change of the public asset in the product with corrective and improvement changes in related core assets and custom assets. Rearward amend promulgation is when changes transmit from the product to the core assets and custom assets. An example of rearward amend promulgation is the propagation of corrective change made in a public asset in a product in the core assets and custom assets project in order to make other products to incorporate changes in their public assets. As a matter of policy for SPLs, rearward amend promulgation should occur only when the changes being propagated is important to the product line so that other products can use.

Table 3.1 describes all possible amend promulgation that can occur by showing before and after states of hypothetical assets of core assets (IA), custom assets (DA) and product instances (P). Changes to an asset IA in the core-assets project are indicated by IA*, to an asset DA in the custom assets project are indicated by DA` and to an asset P in the product instance are indicated by P^. The merged result of the changes of the assets from the core, the custom and the product is indicated by P*`^, from the core and the product is indicated by P*^, from the custom and the product is indicated by P`^ and from the core and the custom is indicted by P*`. From the table 3.1, cases 1 to 9 shows onward amend promulgation while cases 10 to 20 show rearward amend promulgation.

The proposed system supports all the above forms of amend promulgation.

Table I: Different forms of amend promulgation

S.NO	BEFORE			AFTER		
1	IA	DA’	P	IA	DA’	P’
2	IA*	DA	P	IA*	DA	P*
3	IA*	DA’	P	IA*	DA’	P*’
4	IA*	DA	P^	IA*	DA	P*^
5	IA	DA’	P^	IA	DA’	P^’
6	IA*	DA’	P^	IA*	DA’	P^*’
7	IA*	DA’	P^	IA*	DA’	P*’
8	IA	DA’	P^	IA	DA’	P’
9	IA*	DA	P^	IA*	DA	P*
10	IA	DA		IA	DA	P
11	IA		P	IA	DA	P
12		DA	P	IA	DA	P
13	IA	DA	P^	IA	DA	P^
14	IA	DA	P^	IA^	DA^	P^

15	IA	DA’	P^	IA^	DA’^	P^
16	IA*	DA	P^	IA*^	DA^	P^
17	IA*	DA’	P^	IA*^	DA’^	P^
18	IA*	DA’	P^	IA^	DA^	P^
19	IA	DA’	P^	IA^	DA^	P^
20	IA*	DA	P^	IA^	DA^	P^
21		DA	P	IA	DA	P
22	IA		P	IA	DA	P
23			P	IA	DA	P

The following describes each of the cases in more details:

- Case 1: The product is sharing the core asset IA and custom asset DA. Changes have been made to DA in the custom assets project. In this case the changes made in the custom-assets project are brought to the shared asset in the product. An example of this case is a correction made to an asset in the custom asset projects which is useful to the product sharing the asset. Thus the changes are pushed to the product.
- Case 2: The product is sharing the core asset IA and custom asset DA. Changes have been made to IA in the core assets project. In this case the changes made in the core assets project are brought to the shared asset in the product. An example of this case is a correction made to an asset in the core asset projects which is useful to the product sharing the asset. Thus the changes are pushed to the product.
- Case 3: The product is sharing the core asset IA and custom asset DA. Changes have been made to IA and DA in the core and custom assets project. In this case the changes made in the core assets and custom assets project are brought to the shared asset in the product. An example of this case is when a correction is made to an asset in the core and custom asset projects which is useful to the product sharing the asset and thus the changes are pushed to the product.
- Case 4: Product is sharing the asset IA from the core asset project and DA from the custom asset project. Changes have been made to the shared asset P in the product and the asset IA in core asset project. In this case the changes from the asset IA in the core project is merged with the shared asset P with the product specific changes.. This case would represent a products independent evolution while bringing correction changes from the core project.
- Case 5: Product is sharing the asset IA from the core asset project and DA from the custom asset project. Changes have been made to the shared asset P in the product and the asset DA in custom asset project, in this case the changes from the asset DA in the custom project is merged with the shared asset P with the product specific changes.At this stage, shared asset P of the product has both set of changes. This case would represent a products independent evolution while bringing correction changes from the core project.
- Case 6: Product is sharing the asset IA from the core asset project and DA from the custom asset project. Changes have been made to the shared asset P in the product, the asset IA in core asset project and the asset DA in the custom asset project, in this case the changes from the asset IA in the core project and DA in the custom project are merged with the shared asset P with the product specific changes, now P of the product has all the set of changes. This case would represent a products independent evolution while bringing correction changes from the core and custom project.
- Case 7: Changes have been made to the assets in the custom assets project, core assets project and product project. The developer wants to replace the modified asset in the product with modified assets in the core and custom assets projects. After the developer performs this action, the asset in the product will be identical to the one in the core and custom assets project. In this case, the developer may find the product specific changes which may not be useful and could be subsequently replaced with the changes made in the core and custom project.
- Case 8: Changes have been made to the assets in the custom assets project and product project. The developer wants to replace the modified asset in the product with modified assets in the custom assets projects. After the developer performs this action, the asset in the product will be identical to the one in the custom assets project. In this case the developer may find the product specific changes which may not be useful and could be subsequently replaced with the changes made in the custom project.
- Case 9: Changes have been made to the assets in the core assets project and product project. The developer wants to replace the modified asset in the product with modified assets in the core assets projects. After the developer performs this action, the asset in the product will be identical to the one in the core assets project. In this case the developer may find the product specific changes might not be useful and could be replacing with the changes made in the core project.
- Case 10: At this stage, an asset from the core assets and custom assets project, which had not been shared with the product, is now shared with the product. The asset may be needed to the product.

- Case 11: At this stage, an asset from the core assets project that had not been shared with the product is now shared with the product. The asset may be needed to the product.
- Case 12: At this stage, an asset from the custom assets project that had not been shared with the product is shared with the product. The asset may be needed to the product.
- Case 13: The Product is sharing the asset IA from the core asset project and DA from the custom asset project. Changes have been made to the shared asset P in the product. In this case, the changes in the product asset P is changed with product specific changes. This case might represent a product's independent evolution.
- Case 14: The Product is sharing the asset IA from the core asset project and DA from the custom asset project. Changes have been made to the shared asset P in the product. In this case, the changes in product asset P are changed with product specific changes and these changes are reflected on core assets and custom assets projects.

Cases fifteen to twenty three are similar to the cases one to nine but changes are propagated in the opposite direction. There is a semantic difference among cases seven eight nine and cases twenty one, twenty two, twenty three.

IV. IMPLEMENTING CHANGE PROMULGATION

The proposed method allows product specific changes to shared components without interfering with the changes made to the referred component in the core and custom projects. To support product specific changes to shared core assets and custom assets in order to avoid interference between the product's changes and the changes to the core asset's and custom asset's project's, the core asset project and the custom asset project creates a product specific branch to support the changes. When a product developer checks in their product project with changes to a shared core asset or custom asset, the core assets and custom assets projects created an automatic branch to support it. The subsequent checkin of changes to this shared asset for that particular product creates more versions of the product specific support branch created earlier.

The following is the algorithm for the proposed approach.

Procedure1. Configuration management:

```
1.  create assets();
2.  create product();
3.  if product is changed to next version then
4.  read next version elements or assets
5.  d=diff(historical artifacts, next version elements or artifacts)
6.  if d is true then
7.  Update the artifacts with corresponding core and custom artifacts in IA and DA list respectively.
8.  go to step 5 until artifacts are completed.
9.  end if;
10. end if;
11. go to step 2 until product = n
```

Procedure 2. Creating the core and custom assets:

```
1. create assets()
2. {
3. read asset
4. if asset is basic component then
5. store in IA list
6. else
7. store in DA list
8. end if;
9. }
```

Procedure 3. Creating the product with different core and custom assets

```
1.  create product()
2.  {
3.  read the requirements of customer
4.  if requirements match with the historical asset then
5.  get IA list
6.  select required core assets from IA list
7.  get DA list
8.  select required custom assets from DA list
9.  else
10. create assets();
11. end if;
12. }
```

The following is a run-through evolution of a simple product line example that demonstrates our approach, which supports product line evolution and change promulgation. Fig 4.1 depicts the main development case of product PR project, the core assets (IA) project, the custom asset's (DA) project and the product PQ project. The core asset's project consists of A,B and C. The custom project consists of I, J and K.. Product PR is using A and B asset's from the core asset's project and J from custom asset's project. The product PQ is using B and C from the core asset's project and I and K from the custom asset's project. In the Fig the version trees of four projects consists of trees with one version. In the Fig product PR is at version PR1.0, the core assets project is at IA1.1.1, custom assets project DA1.1.1 and PQ is at version PQ1.0.

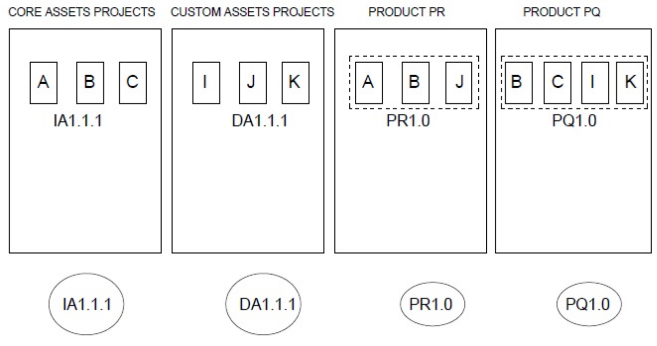

Fig 1: The version trees with First version

Fig 1: Product PR1.0 is using A and B from core assets project and J from custom assets project. Product PQ1.0 is using B and C from core assets project and I and K from custom assets project.

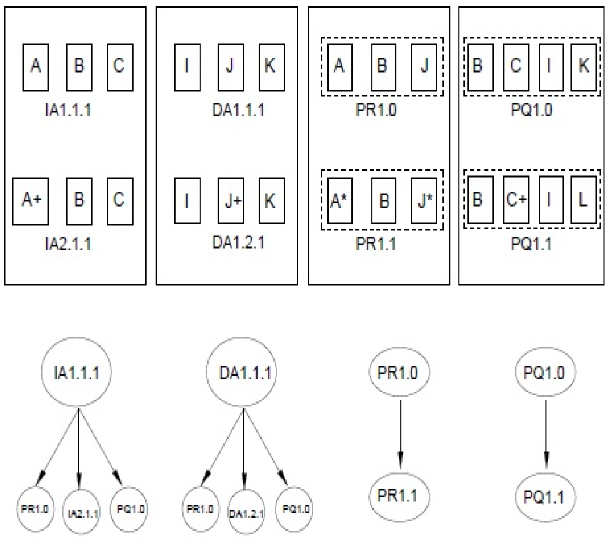

Fig 2: version trees with second version

In the Fig 2 amendments are made to A of core assets, J of custom assets projects, A and J of product R and C of product Q. The changes made in both products R and Q are to shared assets. To support product specific changes to shared assets, this approach automatically creates branches. These branches are created when the changes checked in. PR1.1 is created to support the changes to A and J in product R and PQ2.0 is for product Q's changes to C. The main idea is that all the changes to the shared components are stored in corresponding core assets, custom assets and special branches are created to support product specific changes.

Fig 2 Core assets project has changes to A resulting in IA2.1.1. Custom assets project has changes to J resulting in DA1.2.1. Product R has product specific changes to core asset A and custom asset J, resulting in version PR1.1. Product Q introduces at product specific component L and makes product specific changes to C resulting in PQ2.0.

The Fig 3 shows how amend promulgation is performed. Product R updates A with changes made in the core assets project (onward amend promulgation) this results in version PR2.0 our approach performs a merge of changes made in A in product R of version PR1.1 with A of core assets project of version IA2.1.1. Resulting in IA2.1.2. The changes made to C in product Q is pushed to the core assets project (rearward amend promulgation) in addition the product specific component J is pushed to the custom assets project, so that other products can use it. Our approach performs a merge of changes made in C in product Q of version PQ2.0 with C of the core assets project of version IA 2.1.2. When L is propagated or pushed to the custom assets project, L is also removed from product Q and a shared component is created in its place that refers to L in the custom assets project.

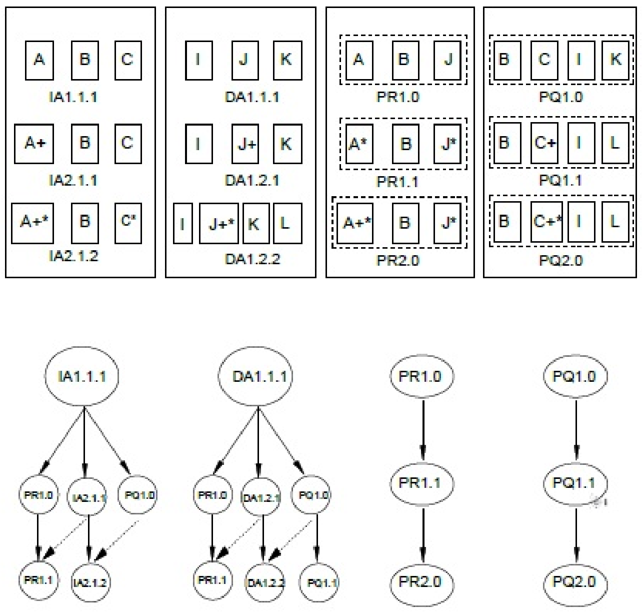

Fig 3: version trees with third version

Fig 3: Changes to C in product Q is pushed to core assets project (rearward amend promulgation), changes to J in product R is pushed to custom assets project (rearward amen promulgation). In addition product specific component J is moved to the custom assets project so that other products can use (rearward amend promulgation). As a result product Q shares J. Product R updates A with changes made from core assets project (onward amend Promulgation).

V. RESULTS AND CONTRIBUTION

To evaluate this approach AAR (Automated Academic Regulations) product line is used. To derive a product user can choose artifacts from core and custom assets. Once the user is with a product, the user can add product specific content and modified shared content. All the cases of change promulgation are evaluated and the model was able to perform all the twenty-three cases of change promulgations described above. In using this approach it is clearly visible as to which version of which asset is present in a particular product.

VII. CONCLUSION AND FUTURE WORK

In configuration management of product line engineering work has been done on multiple evolving baselines of the assets rather than on a large number of individual product baselines. The proposed approach consists of a version model for a product line consisting of a core assets project, custom assets project and multiple product projects, where core assets and custom assets are shared among the products with the use of shared components. Using the shared component data structure and branching of core assets project and custom assets project it is able to support the independent development of core assets, custom assets and products and change promulgation between them. This approach supports twenty-three cases of amend promulgations. Older SCM systems such as CVS [11] do not support code sharing. More recent SCM systems such as subversion [12], GIT [13] and Bazaar [14], support sharing of repositories which are closer to the idea of sharing components. However, these do not address the SPL evolution problem.

As a part of future work, a frame-work for managing interdependencies among various parts of the system in product line evolution is proposed. It is also proposed to give certain recommendations to be followed in the rearward amend promulgation.

ACKNOWLEDGEMENTS

We are very much thankful for reviewers at ROSE lab, JNTUA, Anantapuramu, and CMRCET for providing facilities to carry out this research work.

REFERENCES

[1] Pressman, R.1992.Software engineering: A practitioner's Approach, 3rd edition, Mc-GrawHill.
[2] K.pohi, G.Bockle, F.Van der Liinden, Software Product Line Engineering Foundations, Principles and Techniques, Springer,2005.
[3] Frame work for Software Product Line Practice, http://www.sei.cmu.edu/reports/87cm004.pdf
[4] T.Mens, S. Demeyer, software evolution, DOI 10. 1007/978-3-540-7644 Springer 2008.
[5] L.N. Paul Clements and L.M. Northorop, software product lines practices and patterns, Addison- Wesley professional, 3rev ed., 2001.
[6] J. Van gurp and C. Prehofer, Version management tools as a basis for integrating product derivation and software product families, in proceedings of the workshop on variability management- working with variability mechanisms at SPLC, No. 152.06/E,pp. 48 – 58, October 2006.
[7] C.W Krueger, Variation management for software production lines, in SPLC 2: proceedings of the second international conference on software product lines, (London, UK), PP.37-48,Springer- verlag, 2002.
[8] R.C. van Ommering, Configuration management is component based product populations, in SCM, pp.16-23, 2001.
[9] van Deursen, M.de Jonge, and T. Kuipers, Feature-based product line instantiation using source-level packages, 2002.
[10] Thao. Managing Evolution of Software Product Line, in Proceedings of the 34th International Conference on Software Engineering (ICSE 2012), IEEE computer Society Press, 2012
[11] A configuration Management Model for Software Product Line, Linguo Yu and Srini Ramaswamy
[12] T. Morse, CVS, Linux Journal, vol.1996, no 21es, p.3, 1996.
[13] Subversion.tigris.org, http://subversion.tigris.org/
[14] Git-fast version control system, http://git-scm.com/
[15] Bazaar versioning system, http://bazaar.conical .com/
[16] Cvs-concurrent versions system
[17] Mercurial SCM, http://mercurial.selenc.com.
[18] T.Mens, A state-of-the-art survey on software merging, Software Engineering, IEEE Transactions on, vol.28,pp. 449-462,May 2002.
[19] XML security standard, http://www.w3.org/standards/xml/security/
[20] Google docs, http://www.google.com/google-d-/documents/
[21] P.Clements and L.M..Northrop, Software Product lines: Practices and Patterns. Addison-wesley,2002.
[22] Software Architecture in Practice, second edition LenBass, Paul Clements, Rick Kazman, pearson-2010.
[23] D.Batory, D.Benavides, and a. Ruiz-Cortes, Automated analysis of feature models: challenges ahead, Commun.ACM,vol. 49, pp. 45-47,December,2006.

K.L.S Soujanya received B.E degree from Osmania University, Hyderabad, Telangana , India and M.Tech degree in CSE from JNTU College of Engineering, Anantapuramu, Andhra Pradesh, India. She is persuing Ph.D at JNTUA, Anatapuramu, Andhra Pradesh, India. Attended various conferences at IIIT Hyderabad, IIT Chennai, Infosys Mysore and workshops at JNTUA, JNTUH. Her research areas include software engineering, cloud computing and data mining.

Dr. Ananda Rao Akepogu received B.Tech degree in Computer Science & Engineering from University of Hyderabad, Andhra Pradesh, India and M.Tech degree in A.I & Robotics from University of Hyderabad, Andhra Pradesh, India. He received PhDdegree from Indian Institute of Technology Madras, Chennai, India. He is Professor of Computer Science & Engineering Department and currently working as Director industrial relations and placements of JNTUA College of Engineering, Anantapur, Jawaharlal Nehru Technological University, Andhra Pradesh, India. Dr. Rao published more than 100 publications in various National and International Journals/Conferences. He received **Best Research Paper award** for the paper titled "An Approach to Test Case Design for Cost Effective Software Testing" in an International Conference on Software Engineering held at Hong Kong, 18-20 March 2009. He also received **Best Educationist Award,Bharat Vidya Shiromani Award**, **Rashtriya Vidya Gaurav Gold Medal Award, Best Computer Teacher Award** and **Best Teacher Award** from the Andhra Pradesh chief minister for the year 2014. His main research interest includes software engineering and data mining