Tested Approach for Variability Management Enhancing in Software Product Line

Andrii Kolesnyk[1] and Olga Slabospitskaya[1]

[1] Institute of Software Systems of NAS, Akedemika Glushkova st., 40, Kiev, Ukraine
{kolesnyk, slabospitskaya.olga}@gmail.com

Abstract. The paper declares a novel approach for the process enhancing of managing Variability – the ability of a software system or artefact in Software Product Line (PL) to be extended, changed, customized or configured for use in a specific context – with the proper quality characteristics to mitigate its current limitations. New Variability Model and Management Functions to process its element are proposed as this process Core. The model consistently represents variabilities both in PL structure and artefacts across all PL development stages and stakeholders' viewpoints along with the dedicated assessment submodel. The functions compose separate actions as to variability into the single cycle like Doeming Plan-Do-Check-Act one where decisions should be rational. Presented successful Case Study purposes at the Core proposed testing along with the dedicated Configurator implemented in instrumental and technological complex just developed in the Institute of Software Systems of NAS.

Keywords: Software Product Line, Variability Model, Variability Management, Reusable Asset, Configuration.

Key Terms: Model-based software system development: Method, Model, Methodology, Process, Software System.

1 Introduction

Effective and efficient large, complex and multi-purposed software systems composition from more simple reusable assets was one of the challenges being addressed in the research project named "Theoretical Fundament of Generative Programming and Means of its Support"(2007-2011, № 0107U002205) [1, 2] just accomplished in the Institute of Software Systems of NAS of Ukraine. Over the project Software Product Line (PL) Engineering [3] has proven to be the promising paradigm to produce a diversity of high-quality similar-but-different products with limited time and efforts.

But it was at once well-recognized that the key success factor in PL Engineering is a proper management of both the two kinds of variability disambiguated by Metzger et al. [4]. The first, specific PL variability, describes properties and qualities that should vary between the systems of the PL and that should not. In return, the second

one is single Software variability – i.e., the ability of a software system or artefact to be extended, changed, customized or configured for use in a specific context.

However, just now researchers' efforts concentrate foremost on variability modeling [4, 5] and implementing [3], while challengeable problems of its planning and evolving less attention are paid. One of perspective frameworks to consistently cope with them is COVAMOF [6] determining whether, when and how software variability in PL should evolve with special meta-model and method for its assessment.

The paper pursues the same goal but for both the above kinds of variability. It proposes dedicated Variability model and Management Functions based on its estimates as the Core of an empirical approach for PL Variability Management process defining that is enhanced with appropriate quality characteristics. The Core is tested with the dedicated Configurator elaborated within the research project above.

2 Variability Issues in PL

Variability items to be managed. Let fix, to use hereafter, the definitions of basic Variability items that have to be manage over PL development and therefore need to be explicitly modeled, following up the origins [3, 4, 6].

The first such item is a variation point. It is an abstraction that identifies location in software system or artifact at which a choice can be made between values or variants. As Deelstra et al. [6] note, it is not by-product of design and implementation of variability, but answers the question, what does vary, being therefore identified as central element in managing variability. Each variation point is associated with a value, zero or more variants. Variation points are categorized to five basic types such as: optional (zero or one variant out of 1,…, m associated variants), alternative (1 out of 1,…, m), optional variant (0,…, n out of 1,…, m), variant (1,…, n out of 1,…, m), and value (a value that can be chosen within a predefined range).

A variant is thus the second Variability item answering the question, how does vary the variation point it is associated with [4].

The third one is dependency [3, 6]. It specifies a function of how the choices at the variation points in the PL influence a system property value, e.g. quality attribute, as well as the valid range for this value. The last one, namely constraint, is a predicate that defines possible interrelations between various variation points and variants.

Variability Levels in PL development. Thorough study of PL Development process [2, 3] experiences five possible types (t) of variation points and variants corresponding their Lyfe Cycle over PL Development:

- Features, i.e. abstract concepts reflecting commonalities and variabilities of software products in PL relevant for some Stakeholder that might represent a technical function, a function group or a non-functional characteristics ($t = 1$)
- Requirements as to Software Products ($t = 2$)
- Architecture components ($t = 3$)
- Database tables ($t = 4$)
- Software artifacts ($t = 5$).

Target characteristics for Variability Management enhancing. Based on the experience and ideas formed during the above Institute of Software Systems of NAS research project (№ 0107U002205) [1, 2], four Berg et al. [5] essential quality characteristics are chosen to adopt as a target for the Variability Management process to be defined. These are consistency, scalability, traceability and visibility.

Consistency means that Variability should be handled the same way at all above levels of abstraction and across all PL development phases to reduce the ambiguities that might occur when using different methods for managing variability at different abstraction levels. Scalability prescribes that the methods used should be easily applicable for both the single component and a large complex system. In turn, traceability requires that Variability items at different levels of abstraction and across development phases should be explicitly linked both upwards and downwards to simplify PL evolution and maintenance. Lastly, visibility presupposes understandable representations of all Variability items in appropriate and intellectual user interface.

3 An Approach Proposed to enhance Variability Management

An approach proposed is simply to define a Core of Enhanced (i.e. possessing the above target characteristics) Variability Management process, then continuously test it under various stressing conditions and refine accordingly to the lessons learnt.

The Core is formally a tetrad of a sets open for expanding

$$C=\langle AS; FN; ENV; DM\rangle;\ ENV=\langle VM; RP; VP; CP\rangle, \tag{1}$$

where AS denotes a priori assumptions as to PL development organizing; FN is the set of Generic Functions for PL Variability Management Process; ENV is the environment for FN operation including dedicated Variability Model (VM) described hereafter, PL core assets Repository (RP) and profiles both for PL variability with VM (VP) and for core assets reuse over PL development (CP); DM is the set of Demands the Functions should meet to enable the target quality characteristics be really attained.

Initially AS in (1) assumes PL development to be the series of unified production rounds interchanging with the rounds of PL environment actualization.

In the FN set, four target Variability Management functions are elicited as generic through comparative study of popular Variability Management process templates [2-4, 6] within the perspective of Doeming's PDCA Management Cycle [7]. These are informed and consistent Variability Planning, Implementing in PL artifacts, all-aspect Controlling and Evolving up to both retrospective and current elements of VP and CP. They serve due rationales for appropriate managerial decisions over FN processing. All necessary technological prerequisites as well as initial VM and RP are created with the fifth function of Variability Management Initiation.

The DM set from (1) composes consistency, scalability, traceability and visibility demands being inspired the title target characteristics and also the additional demand of rationality for all decisions being made under the functions FN processing. To

meet this demand is the main purpose of *VM* and both the above variability kinds assessment method with it that enables *VP* and, particularly, *CP* .

4 Consistent and Traceable Variability Model

Let's particularize the above target characteristics and demands *DM* from (1) to fix inspired demands the dedicated *VM* should meet to pursue its purpose in (1):

- uniform, consistent and traceable representation for all the variety of variability items and their interrelations over all the stages both for PL Domain and Application Engineering processes [2-6, 8] as well as for all functional segments from PL scope and also for all its stakeholders' viewpoints
- traceable notations usage for PL artefacts modelling appropriate to their types
- explicit identification of commonalities and variabilities across all PL development artefacts, stages and stakeholders' viewpoints
- sound, informed and consistent PL variability profile assessment.

Relevant Variability Model is defined [8] to be a hybrid structured triple

$$VM = \langle SV; AV; EV \rangle, \tag{2}$$

where: submodels SV and AV represent variability in PL structure and its artifact; EV is an integrated submodel for informed and consistent variability assessment.

The first submodel SV in (2) gives the formal representations of all the features from PL scope, both commonalities and variabilities, artifacts to implement them and their links on the base of feature modeling approaches [2-4, 6]. It is a structured tuple

$$SV = \langle G_1; \langle \langle G_t, TR_t \rangle, t = 2,3,4,5 \rangle; CN; DP \rangle, \tag{3}$$

where: $G_t = \langle F_t, LF_t \rangle$ is the graph where unique identifiers of *t*-typed PL artefacts (i. e. features, requirements, architecture modules, database tables, software components and tests) are nodes sets (F_t) linked through obligatory and variant binding (LF_t); TR_t is bilateral traceability links between the nodes of G_{t-1} and G_t graphs; CN and DP are the predicates on $\otimes_{t=1,\ldots,5} F_t$ representing PL constraints and dependencies.

In turn, the second submodel AV in (2) provides unified formal representations for all PL software products currently located in repository, being developed or might be developed eventually within current PL scope together with their development products (requirements etc.). To explicit reflecting the elements of SV (2) model in PL software products, any t-typed artefact is formally seen as cross-cutting "fragment" of SV . It is bounded with continuous upwards – downwards traceability links TR_t from (3), which interrelates this artifact with the features it should implement and the final software product. The model AV is a structured tuple

$$AV(id_t) = \langle g_1; \langle \langle g_u, tr_u \rangle, u = 2,\ldots,t \rangle; \langle \langle p_u, tr_u \rangle, u = t+1,\ldots,5 \rangle; cn; dp; s \rangle, \tag{4}$$

where: id_t is the modeled artefact's unique identifier; g_u and p_u are subgrafs of G_u from (3) representing the artefacts that are implemented with id_t and, respectively, implement it over PL development; $tr_u \in TR_u$ are subsets of traceability links between the nodes of g_{u-1} and g_u; cn and dp are the limitations of CN and DP on Cartesian product of g_u; and p_u nodes sets and s is the artefact's current state (e.g. "core asset" etc.).

Note that each of the five "horizontal" planes, implicitly defined with formulas (3), (4), reflects the viewpoints both at PL and artefact variability by specific PL Stakeholders group being represented over PL development with the proper-typed artifacts – from the customers' features at the first level (G_1, g_1) downwards to the programmers' and testers software and tests at the fifth one (G_5, g_5).

The third submodel of SV model expands Metzger's Orthogonal Variability Model [4] with a novel dimension of sound quantitative variability estimates $vp \in VP$ from (1). They quantify the level of PL variability adequacy within the perspectives of both PL customers' business needs in its products' functions and PL developers' requirements as to them. For the estimates' plausibility and consistency to increase, universal preferences model such as Bayesian Net, Analytical Hierarchy and Value Tree should be configured up to the assessment situation [9]. It is also a triple

$$EV = \langle \langle VL, VR \rangle; VP; VAR \rangle; VP = \langle VpR, VpA, VpE, VpB \rangle; VAR = \langle VR, VA, VE, VB \rangle, \quad (5)$$

where: VL and VR are subsubmodels both for integrated variability adequacy level in a whole and, respectively, for its sublevels corresponding the artifacts' types; VpR, VpA, VpE, VpB are the sets of variation points in SV (3) model; VR, VA, VE, VB are the sets of variants for these variation points.

5 A Case Study to Test the Variability Management Core

Here the probe implementation of PL artefact variability model AV (4) is considered for simple domain of quadratic equations solving. While classical feature diagram [4, 6] clearly demonstrates the variety of variability items at the feature level, it's quite difficult to implement it in specific application in PL repository. Instead MS Visual Studio Windows Workflow Foundation (WWF) enables more information about AV with appropriate diagram (see Fig. 1).

Let's explain how such variability might be visualized and on-line managed with that WWF diagram. Note that the letter *A* at the Fig. 1 connected with the "plus" pictograms denotes the variation points that might be filled with the reusable assets as their variants while the letter *B* denotes possible variants. In the case at hand the "Discriminant" component contains simple code to find discriminant ($D = b * b - 4 * a * c$), implemented by the developer responsible for producing PL core assets. Depending on its operating outcome, there are three scenarios (use case):

$D>0; D=0; D<0$ and three corresponding assets for them: “TwoRoots”, “ExactlyOneRoot”, “NoRoots“.

When using WWF, Visual Studio environment don’t prohibit the developer from binding any reusable assets and variation points. In other words, we need a dedicated software tool that should support both the *VM* proposed (2)-(5) forming and actualizing and application configurations changing based on *VM* and reusable assets.

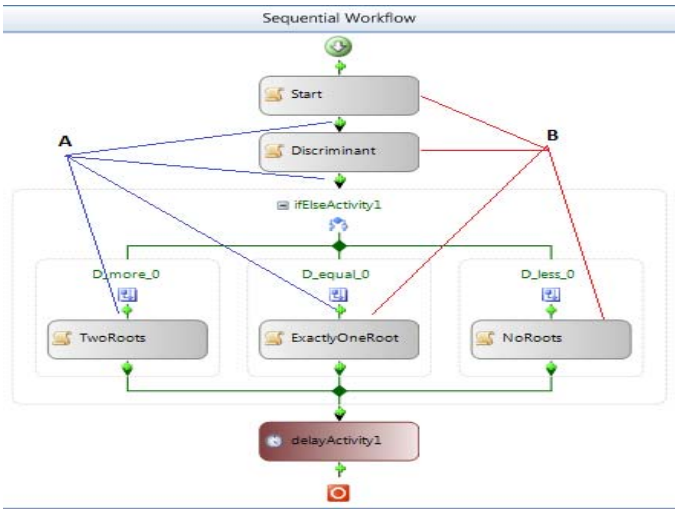

Fig. 1. Sample artefact variability model and reusable components are represented.

Trial prototype of such a tool, named Configurator, has been implemented within instrumental and technological complex just developed in the Institute of Software Systems of NAS [2, 10]. It purposes at configuring a diversity of similar-but-different applications from reusable software components and also at expanding and modifying applications with variation points and variants [1, 2, 8] based on WWF [2, 10, 11]. An interim result of configuration process with the Configurator is XML file shown beneath. It is an instruction for executable file compiling to create the target application.

```
  <SequentialWorkflowActivity>
  <CodeActivity x:Name="Start"
  ExecuteCode="codeActivity1_ExecuteCode" />
  <CodeActivity x:Name="Discriminant"
  ExecuteCode="codeActivity1_ExecuteCode_1" />
  <IfElseActivity x:Name="ifElseActivity1">
            <IfElseBranchActivity x:Name="D_more_0">
              <IfElseBranchActivity.Condition>
                 <CodeCondition Condition="WorkMeth1" />
               </IfElseBranchActivity.Condition>
               <CodeActivity x:Name="TwoRoots"
  ExecuteCode="codeActivity1_ExecuteCode_2" />
              </IfElseBranchActivity>
              <IfElseBranchActivity x:Name="D_equal_0">
```

```
                <IfElseBranchActivity.Condition>
                  <CodeCondition Condition="WorkMeth2" />
                </IfElseBranchActivity.Condition>
                <CodeActivity x:Name="ExactlyOneRoot"
ExecuteCode="codeActivity2_ExecuteCode" />
              </IfElseBranchActivity>
              <IfElseBranchActivity x:Name="D_less_0">
                <IfElseBranchActivity.Condition>
                  <CodeCondition Condition="WorkMeth3" />
                </IfElseBranchActivity.Condition>
                <CodeActivity x:Name="NoRoots"
ExecuteCode="codeActivity3_ExecuteCode" />
              </IfElseBranchActivity>
            </IfElseActivity>
            <DelayActivity TimeoutDuration="00:00:05"
x:Name="delayActivity1" />
</SequentialWorkflowActivity>
```

The configuration process producing this XML file is initiated through the special chart processing with WWF tool in Visual Studio environment (see Fig. 2).

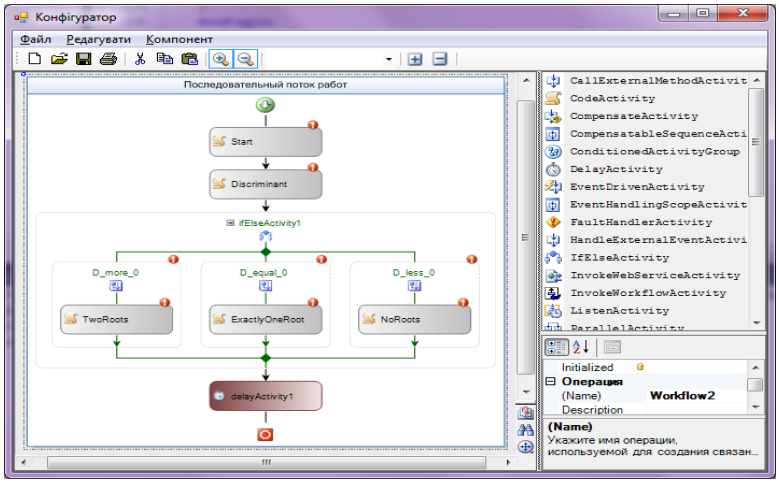

Fig. 2. Processing the reusable components by the Configurator is depicted.

Note that the application created is variable i.e. enables square equation solving under various conditions prescribed.

6 Conclusions

A novel Variability Model and generic Functions are elaborated for its enhanced (i.e. informed, consistent, scalable, traceable and capable to visualize the variability) support whole over PL development. The Model uniformly and consistently

represents all Variability items across all the relevant stakeholders' viewpoints and over all abstraction levels both in PL structure and artifacts. It also includes dedicated submodel for informed and consistent variability assessment. In turn, the Functions – Variability Planning, Implementing in PL artifacts, all-aspect Controlling and Evolving up to assessment results are serviced with Initiation one to initially create the above Model.

An approach is declared to construct Variability Management process being enhanced in the above sense. It prescribes to couple the Model and the Functions as a priori process Core, then continuously test and refine it up to the lessons learnt.

To attempt such a testing, trial Workflow-based Configurator is implemented within the instrumental and technological complex just developed in the Institute of Software Systems of NAS to effectively produce complex systems from the assets. Based on successful Case Study of sample product variant deriving, the authors now update their approach to support all the Functions and fulfil an industrial Case Study.

References

1. Lavrischeva, E.: Generative Programming of Software Products and Their Families [in Ukrainian]. In: Problems in Programming, vol. 1, pp. 3--16. Kiev (2009)
2. Lavrischeva, E., Koval, G., Babenko, L., Slabospitska, O., Ignatenko, P.: New Theoretical Foundations of Production Methods of Software Systems in Generative Programming Context [in Ukrainian]. Electronic monograph, in: UK-2011, vol. 67. Kiev (2011)
3. V. d. Linden, F., Schmid, K., Rommes, E.: Software product lines in action: the best industrial practice in product line engineering. Springer, Heidelberg (2007)
4. Metzger, A., Heymans, P., Pohl, K., Schobbens, P.-Y., Saval, G.: Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis. In: 15th IEEE International Requirements Engineering Conference, pp. 243--253. IEEE Press, New York (2007)
5. Berg, K., Bishop, J., Muthig, D.: Tracing Software Product Line Variability – From Problem to Solution Space. In: Proc. of SAICSIT'2005, pp. 111--120. (2005)
6. Deelstra, S., Sinnema, M., Bosch, J.: Variability assessment in software product families. In: Information and Software Technology, vol. 51, pp. 195--218. Elsevier (2009)
7. Walton, M.: The Deming Management method. Dodd, Mead. New York (1986)
8. Lavrischeva, K., Slabospickaya O., Kolesnik, A, Koval, G.: The Theoretical View for Software Family Variability Management [in Ukrainian]. In: Bulletin of University of Kiev. Series: Physics & Mathematics, vol. 1, pp. 151--158. Kiev (2011)
9. Lavrischeva, E., Slabospitcka, O.: An Approach for Expert Assessment in Software Engineering. In: Cybernetics and Systems Analysis, vol. 45, no. 4, pp. 638--654. SpringerLink (2009)
10. Lavrischeva, E.: Instrumental and Technological Complex for Developing and Learning Aspects of Software System Development [in Ukrainian]. In: Bulletin of NAS of Ukraine, vol. 3, pp. 17--27. Kiev (2012)
11. Kolesnik, A.: Approaches to Configure Reusable Assets [in Ukrainian]. In: Problems in Programming, vol. 4, pp. 63--71. Kiev (2011)