Designing variability models for software, operating systems and their families[1]

[1,2]*E.M. Lavrischeva <lavr@ispras.ru>*
[1]*V.S. Mutilin <mutilin@ispras.ru>*
[1]*A.G. Ryzhov <ryzhov@ispras.ru>*
[1]*Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, A. Solzhenitsyn st., Moscow, 109004, Russia
[2]*Moscow Institute of Physics and Technology,*
9, Institutskiy per., Dolgoprudny, Moscow region, 141701, Russia

Abstract. The complexity of existing Legacy systems and the difficulty of amending it led to the development of the new concept of variability of systems specified by a model of the characteristics of FM (Feature Model). In the paper, we discuss the approaches to formal definition of FM and creating on its basis variants of program systems (PS), operating systems (OS) and families of program systems (FPS) for PS and OS. We give methods of manufacturing of PS in the Product Family/Product Lines, the conveyor of K.Czarnecki for assembling of artifacts in the space of problems and solutions, logical-mathematical modeling of PS from the functional and interface objects by Object-Components Method (OCM), extraction of the functional elements from OS kernel to FM for the generation of new variants of the OS. We discuss approaches for formalization of variability of legacy and new PS and their FPS. The new concept of management of variability systems with help OCM is defined. The approach to verify models of the FM, PS, FPS and OS and to configuration of functional and interface objects for obtaining the variants of the resulting product are proposed. We elaborate the characteristics for the testing process of variants of the PS, OS and FPS.

Keywords: variability model; software systems; family of systems; configuration; variant; functional, interface element; requirement; management

DOI:10.15514/ISPRAS-2017-29(5)-6

For citation: Lavrischeva E.M., Mutilin V.S., Ryzhov A.G. Designing variability models for software, operating systems and their families. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 93-110. DOI: 10.15514/ISPRAS-2017-29(5)-6

[1]The work is supported by RFFI grant N16-01-00352

1. Introduction

In the recent years, new modeling methods of the program systems (PS) and families (FPS) appeared in software engineering. The methods are aiming to ensure the variability of software systems, both legacy and newly produced ones. One of the first Feature Models (FM) called Product Line/Product Family was developed at the Software Engineering Institute (sei.cmu.edu) for manufacturing software products and their families basing on the assets by customers requests. Product line is group of products or services sharing a common managed set of features that satisfy specific needs of a selected market or mission. K.Czarnecki proposed a concept of generation of PS and FPS based on FM from reuses and artifacts. Object-Component Method (OCM) enables modeling of functional elements with support for variability [1-15].

In the paper, we introduce new models with functional and interface elements and FM from these elements for generation of variants of PS and their families.

2. The Basic Foundation of the Variability of Systems and Families

The FM for software products was first proposed by K.Pohl [1, 2] as a basis for creating variants of software and OS [3-9]:

1) requirements for software are specified by means of the languages – FODA, RSEB, Forfamel, RequiLine, CBFM , Use Case precedents UML etc.;
2) tools – ConIPF, CONSUL/Pure::Variants, GEARS are used for integrating the variability of artifacts with special languages, like Koala, xADL, OVM, VSL etc.;
3) OS mechanisms and functions (e.g. Unix, Linux, etc.), which can be generated in LEADS, OCM[16-27] with the languages (VSL, ConIPF, CBFM, Koalish, Pure::Variant, COVAMOF and others) establishing relationships between characteristics of FM and the variants of PS.

This paper sets out the basic principles of simulation of variability of PS and FPS in existing approaches of K.Pohl, K.Czarnecki, etc. and proposes the Object Component Method of presenting FM on four-level design using a functional and interface objects. We also define configuration management process in accordance with the Deming cycle [22] to obtain variants of the PS and FPS.

2.1. Variability of Products and Systems

K.Pohl introduced the concept of variability in FM out of existing artifacts, reuses, etc. Variability is a property of a product (system) for expanding, changing, adaptation, or configuration for use in a particular context and to ensure its subsequent evolution [1, 2]. The FM model includes common functional and non-functional characteristics of items that can be used by members of the family of FPS for creating different variants of the PS configured at variation points [1-29].The variation point is a place in the Legacy-system, which are used for production variant of the big systems.

FM defines the process of creating a product from existing software elements, which are called Ready to Use Component (RUC) [11], which includes – reuses, assets, applications, etc. The FM in Software Product Line Engineering (SPLE) is based on two processes: engineering of domain and application engineering.

The main aspects of variability of products and systems are:

- model characteristics of the FM with variation points for functional elements;
- variability of the system architecture with variation points;
- managing variability of RUC.

2.2. Variability in the Space of Problems and Solutions

K.Czarnecki [3, 6] provides a modeling of the architecture of the PS and FPS in the problem space and problem solution similar to SPLE approach. The basis of the approach is the characteristics of RUC that appear in FM implementing requirements to PS or FPS. Between characteristics (n) and requirements (m) there may be $n \times m$ relations. Each PS is defined by selecting a group of characteristics.

FM consists of the functions that are available to the user of the system and can be in the spaces of problems and solutions, and describes the domain model by means of DSL (Domain Specific Language) with a means for increasing the level of abstraction of FPS.

2.3. Variability of the Functional and Interface Elements in OCM

The Object-component method OCM proposes a four-level design of object model (OM) of PS and FPS [21]. After design of the OM, we obtain the graph G, which has the form:

$G = (G_{t1}, G_{t2}, G_{t3}, G_{t4})$, where

G_{t1} – objects at the synthesis level (t = 1);

G_{t2} – FM at the characteristics level (t = 2);

G_{t3} – functions at the structural level (t = 3);

G_{t4} – interfaces relationships at the behavioral level (t = 4).

Fig. 1 shows the elements of processing on the levels of design and structure of OCM objects, their structure, characteristic functions $F_o = (f_{o1}, \dots, f_{on})$ and interface elements of them $I_o = (i_{o1}, \dots, i_{om})$ [4, 13, 21].

These features of functional and interface object of OM are located in the PS and FPS.

A functional object f_o specifies a formal description of application functions PS, which ensure the solution of problems of a particular domain. This object is given by a triple: the name, data types and their values.

Interface object i_o specifies a formal description of the operations and data of functional objects. The i_o object is a mediator of interacting functional objects and $I_o(f_o)$ equals to $In(f_o)$ or $Out(f_o)$ or $Inout(f_o)$, where

$In(f_o)$ is a set of input interfaces for transferring data from the f_o to the other objects;

$Out(f_o)$ is a set of output interfaces for transferring output data back to the object f_o;

$Inout(f_o)$ is an intermediate interface that converts data from/to f_o.

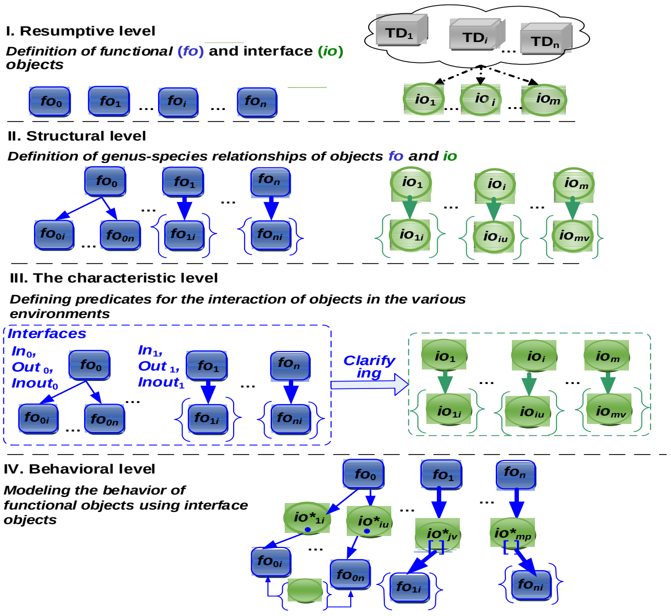
I. Resumptive level
Definition of functional (fo) and interface (io) objects
TD1
TDi
TDn
fo0
fo1
foi
fon
io1
io i
iom
II. Structural level
Definition of genus-species relationships of objects fo and io
fo0
fo1
fon
fo0i
fo0n
fo1i
foni
io1
ioi
iom
io1i
ioiu
iomv
III. The characteristic level
Defining predicates for the interaction of objects in the various environments
Interfaces
In0, Out 0, Inout0
In1, Out 1, Inout1
fo0
fo1
fon
fo0i
fo0n
fo1i
foni
Clarify ing
io1
ioi
iom
io1i
ioiu
iomv
IV. Behavioral level
Modeling the behavior of functional objects using interface objects
fo0
fo1
fon
io*1i
io*iu
io*jv
io*mp
fo0i
fo0n
fo1i
foni

Fig. 1. OM of graph G with functional and interface objects

Axiom. For each functional object, the FPS has at least one characteristic (internal or external) that defines semantics and a unique identification it in the set of F_o and interfaces I_o.

Features allow to establish the truth of the matching types with $Con_i = (P_{i1}, \ldots, P_{ik})$ where P_{ik} is a condition predicate on F_{oi}.

Four-level mathematical design FPS of functional and interface objects is defined by the graph: $G = (F_o, I_o, R)$, where F_o– the set of functional objects, I_o – the set of interface objects, R – the set of relations between these objects [13].

Graph G includes a front-end objects I_o (Fig. 2), which call the other object and pass appropriate data with the required type and size.

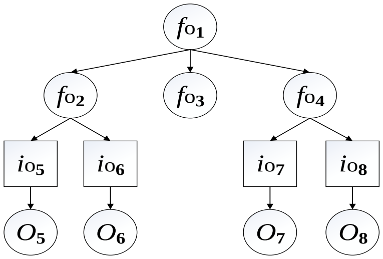

Fig. 2. The graph G on the set of functional and interface objects

Nodes of G represent functional elements – f_{o1}, f_{o2}, f_{o3}, f_{o4}, f_{o5}, f_{o6}, f_{o7}, f_{o8} and interface elements – i_{o5}, i_{o6}, i_{o7}, i_{o8}, and edges correspond to relations *R* between all types of objects.

Elements $f_{o1} – f_{o8}$ are described in any programming language, and front-end objects $i_{o5} – i_{o8}$ are described in IDL. The parameters of the external characteristics of the interface objects are passed between objects through interfaces, and are marked as *In* (input), *Out* (output) and *Inout* (input and output).

The relationship between the functional objects f_{ok}, f_{ol} is provided by interface objects, i.e. f_{ok} is $In(f_{ok})$ or $Out(f_{ok})$; f_{ol} is $In(f_{ol})$ or $Out(f_{ol})$:

$$f_{ok} \cdot f_{ol} = (Out(f_{ol}), In(f_{ok}));$$

Theorem. The functional interaction between two functional objects is correct, if the first object fully matches functions and data that are required by another object: $In(f_{ok}) \subseteq Out(f_{ol})$.

With graph G it is possible to construct individual programs P_0 - P_3 using mathematical operation $\cup$ and corresponding *link* operation:

1) $P_0 = (P_1 \cup P_2 \cup P_3)$;
2) $P_1 = f_{O2} \cup f_{O5}$, *link* $P_1 = In\ i_{O5} (f_{O2} \cup f_{O5})$;
3) $P_2 = f_{O2} \cup f_{O6}$, *link* $P_2 = In\ i_{O6} (f_{O2} \cup f_{O8})$;
4) $P_3 = f_{O4} \cup f_{O7}$, *link* $P_3 = In\ i_{O7} (f_{O4} \cup f_{O7})$;

Below we consider the design of models of systems and their variability models.

2.4. The definition of models of the PS, OS and their variability

This section discusses models of PS, FPS, OS and their variability.

2.4.1 The models of PS, FPS and their variability

Model of PS – $M_{ps} = (C_L, M_f, M_s, M_i, M_d)$, where

C_L – are languages $L = L_1, L_2, ..., L_N$,

$M_f = (f_{o1}, f_{o2}, ..., f_{or})$ – functional objects;

$M_s = (Ms_{in}, Ms_{out}, Ms_{inout}\}$ is the set of services – input Ms_{in}, output Ms_{out} and server Ms_{inout};

M_i – is a set of interfaces in IDL;

M_d – data of the *PS* [9, 13-22].

Model of variability PS – $M_{var} = (SV, AV)$ [14, 15], where

SV – submodel of variable architecture *PS*;

AV – submodel of variability of artifacts FPS or RUC.

M_{var} enables variability of products and reduces development costs with the help of RUC.

The submodel *AV* determines the structure of the *PS* from RUC, which are stored in the repository. This submodel displays the characteristics of FPS, as well as aspects of the relationship (through the interfaces) between different levels of the OM. Variation points are handled by the configurator and replaced by some other RUC (correct ones or new).

The submodel $SV = ((G_t, tr_t), Con, Dep)$, where

$G_t = (F_t, LF_t)$ – is a graph of artifacts on level *t*;

tr_t – connection between artifacts on level *t*;

Con, *Dep* – the predicates of the sets of artifacts that define the constraints and dependencies among the functional elements and their indicators of quality.

The concept of a family of programs introduced Dijkstra (1970), which is based on "family" which can be derived from different versions of programs, and can be adjusted and replaced according to requirements [11-24]. Family of program systems – FPS is a set of systems with a common set of concepts, specific data, and functional and interface characteristics that are inherent to every member of the family.

Model of FPS family has the form:

$M_{FPS} = \{M_{OM}, M_{FM}, M_{var}, MC\}$, where

M_{OM} – OM;

M_{FM} – FM;

M_{var} – variability model of FPS;

MC – model of configuration assemblies;

Model of variability of the FPS has the form [18]:

$SV_{FPS} = ((CF; (DR, TC); (CM, FR, TS, TA); (ER, TF)); Con, Dep)$, where

CF – characteristics of the system,

DR – detailed characteristics related to requirements of PS;

TC – relations between the requirements of PS and consumer properties;

CM – the set of formally described software elements of the set of functions *FR*;

TS – the set of formally described tests;

TA – interfaces between elements of FPS;

ER and *TF* – database for processing elements of *CM*;

Con – are domain constraints;

Dep – are dependencies between artifacts of FPS.

To assess the variability of the FPS an orthogonal variability model (OVM) is created [15, 16]. It coordinates the composition and interrelation of the family elements and artifacts of the assembly processes of the PS and FPS. The evaluation model is included in the integrated model of variability of the OVM. It is used to assess the level of variability, taking into account the requirements for artifacts.

The model OVM has the form:

OVM = (EVM, VP), where

EVM = (VL, VR),

VL – model for estimating the level of variability, taking into account the requirements to the components of the architecture, artifacts and data;

VR – model for estimating the level of variability in the FPS, taking into account the requirements, the architecture, artifacts, and data.

VP – the sets of variation points in the FPS structure that specify individual characteristics of the PS, including the constraints *Con* and *Dep* dependencies;

The OVM model defines two types of assessments of the FPS variability – level and relevance to the consumer needs. The assessment of the level of variability and the degree to which it meets the needs is carried out using the value tree and the parameters *VL* and *VR*, which take into account the costs and the frequency of producing new variants for the customer.

2.4.2. Model of operating system kernel and its variability

Model of OS kernel is a collection of individual program fragments implementing the functions of the OS, e.g. Linux [3, 23, 25, 26].

Model of OS kernel is a set of artifacts and interfaces between them:

$M_{os} = (S_k, M_f, M_o, M_i)$, where

S_k – a set of fragments of OS code;

M_f – a set of features;

M_o – a set of dependencies between features,

M_i – a set of interface features (subset of M_f).

The variability model of the OS is identical to the PS model [25].
The OS defines a set of functions and their features. To generate some variant of OS we define the required set of functions and specify the set of interface features.

3. *Managing variability of systems*

Variability of systems depends on requirements, FM, architecture, documentation, tests, etc. In general, the variability can be implemented in both PS and FPS. In the case of PS variability includes documentation, functions and elements of any type. In FPS, the variability includes the sets of individual products.
The variability of the FPS is managed with:

- variation points;
- versions of the artifacts;
- predicate constraints for variation points.

The variability is managed by method of E.Deming, determined by the functions F1 - F4 (Fig.3) of the development of FPS [22-27]:

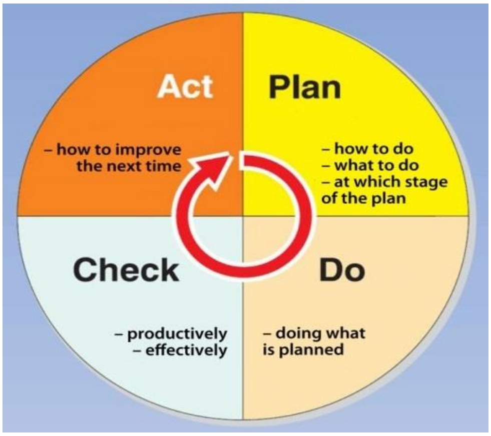

Fig. 3. Functions in the Deming cycle

F1 – operations and actions for preparation of artifacts (Act) [23];

F2 – planning of system construction from the artifacts (Plan at the levels of domain engineering and application engineering);

F3 – testing and verification of changes (Check);

F4 – update system FPS (Do).

Managing variability of FPS in accordance with the requirements *R* is performed by:

1) justification of the solution *F1* (requirement *R1*);

2) agreement on the implementation approach (requirement *R2*);

3) validating the correctness implementation (requirement *R3*);

4) tracking relationships between system characteristics at all development stages (requirement *R4*).

4. Verification of the model variability

The object of verification is a model of the characteristics of the FM and requirements for the development of a new system. Properties of objects, subject to verification of FM are described by means of Linear Temporal Logic (LTL) or a Computational Tree Logic (CTL). Main approaches to formal verification of object are based on deductive verification and model checking [22-29].

Model checking is only applicable to models with a finite number of states and consists in checking that the model conforms to its formal specifications. The specifications are described using the language of temporal logic and assertions. If there is a mismatch between the model and specifications then the counterexample is produced.
The model checking involves execution of the following actions:

1) Build a model of the functional and interface objects, which must have a small number of states.

2) The specification of the requirements in terms of temporal logic.

3) Verification of the model.

5. Assembling artifacts and RUC in FPS

The assembly of artifacts includes steps F1–F4 using the RUC, and storing them in the repository according to requirements and predicates for RUC [13-15]. The configuration management of the PS in the FPS includes:

- identification of configuration items and data;
- managing the process of changes of artifacts and products;
- changing the models of variability under the new requirements;
- assessment of the variability of the PS and the FPS.

For managing artifacts and their variability in the PS and FPS, so-called model configuration environment is created, which includes:

– building process of RUC and artifacts of the system;

– schema description of artifacts and database of requirements;

– architecture, a set of basic RUC and PS in repository.

– Configurator, combining the artifacts in PS and FPS

Managing of configuration environment includes collecting data for such standard operations like reporting and audit.
Reporting configuration – collecting and reporting all necessary information about the state of the development process of the PS.
Audit configuration – guarantee that the PS contains the functionality planned in accordance with the specifications including requirements, architecture and user documentation.
In the development of PS the term "assembly" refers to the process of source code transformation from artifacts or RUC, which can be done on a computer and converted into code to run. One of the steps of Configurator is compilation of the source code into intermediate code or into the machine code. Then the linking process replaces the addresses of functions by real addresses used in the program at run time.
Configuration build is based on (Fig.4):

- the engineering model for the development of elements, components, assemblies, reuses, assets, services, product, FPS, and development management to plan and coordinate this activity [25];
- the process model under development "to ensure re-use" (for reuse) and develop "the use of RUC" (with reuse); the model for controlling variability in the process of configuring the product from RUC.

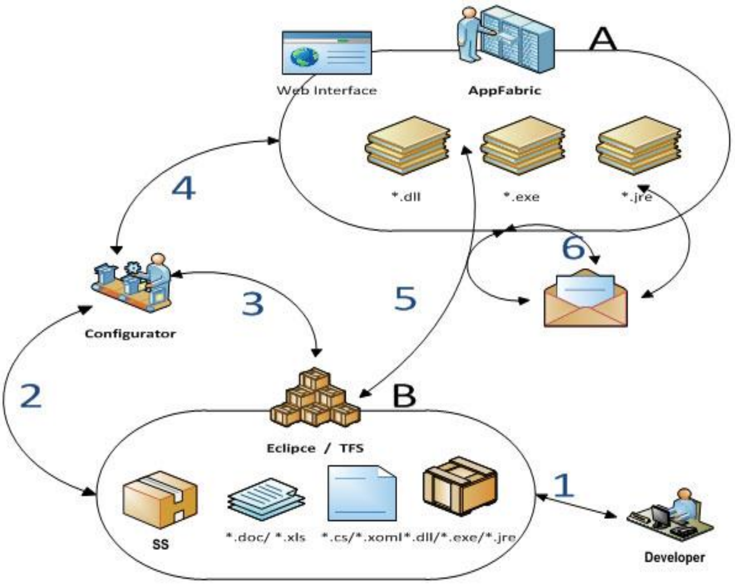

Fig. 4. General model of configurator in .Net

Organization of the development of the PS and FPS is based on the following axioms.

Axiom 1. The technology defines a cyclic sequence of software development processes and updating FPS.

Axiom 2. Each terminal characterization of OM is implemented by one and only one RUC.

6. Testing of FPS products

The structure of FPS includes RUC and test products (plans, test suite, test data, etc.) [25-28]. Test code is generated for testing individual PS and form a set of tests for the FPS. Testing method of FPS is based on requirements-based testing. It specifies the actions to manage testing on the basis of "requirements" with the help of tests to verify functional and interface objects. It determines the control degree of test objects and interfaces, and also the evaluation criterion for the quality of the FPS for the variants of the PS family of FPS. The scheme of testing of the FPS is given in Table 1.

Table 1. Basic schema of testing PS families

Types of testing test	**Testing objects**
Testing of software system architecture	All products, individual products, and functional elements
A set of tests of the family, adaptation as a set of FM to a specific product	All products are individual products of the family
Testing requirements (ScrenTED)	Products of the FPS self-test
Self-testing	Functional objects and interface
The application of metadata	Functional objects and interface
"Test print components" "testable beans" (improving productivity)	Functional objects and interface
Orientation to the service security and reliability (e.g. for Web applications)	Separate objects and interfaces
Automatic test generation for specifications in Boolean form	All products of the family
FCTA (Fault Contribution Tree Analysis)	Family products

Thus, this technique of testing of FPS [28] consists of three steps:
1) Testing artifacts, applications, PS, RUC and reducing the defects in the FPS.
2) Testing of FPS by means of tests.
3) Checking the degree of testing functional and interface objects of the FPS.
The test used offline and is common to test individual elements of the FPS. In the last step the degree is defined by the quality of testing metric *KT*:
KT = 1, if tested operations are independent from each another;
KT = 0, if tested operations depend on the execution path and interoperability.
KT belongs to the segment [0; 1] and is calculated according to the following formula:

$$KT = \frac{1}{n}\sum_{i=1}^{n} KTi$$

The final value KT of the FPS specifies the level of examination: *KT* = 1 if all objects are controlled and *KT* = 0 if not all objects are controlled and 0<*KT* < 1 means that the objects of FPS partially controlled.

The metrics of the control interface CI_i is calculated according to the formula:

$$CI = 1/n\sum_{i=1}^{n} CI_i$$

Where CI_i – the degree of correctness of data type conversion in the *i*-th interface object.
If $CI_i = 1$, the interface is fully controlled, $CI_i = 0$, the interface is not completely controlled;
$CI_i = (0, 1)$ – the interface is partially controlled.
The metric value of *CI* means: 1 – control of CI_i interface is complete; 0 - otherwise.
The test used offline and is common to test elements of the FPS.

testing for compliance with specified requirements to individual family members PS and the entire family FPS checks the degree of product testing. If this degree is high, then the manufactured product is delivered to the customer.

7. Conclusion

The basic fundamental concepts of modeling variability of PS and FPS are given in the original methods of K.Pohl according to FM in Product Family, K.Czarnecki in the space of problems and solutions with ready resources (reuses, assets, artifacts, RUC, etc.) and logic-algebraic approach OCM for modeling the FPS from functional and interface elements. We developed the theory of model definition of FPS from the ready resources for software for FM. The proposed variability model of FPS is based on the specified requirements of the FPS for solving optimization problems of planning of development processes and for evaluation of variability model. Methods for verification, testing, and executing of variants of PS and FPS were proposed.

References

[1]. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.
[2]. Bachmann F., Clements P. Variability in software product lines. CMU/SEI Technical Report CMU/SEI-2005-TR-012, 2005.
[3]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC’10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[4]. Lavrischeva E.M., Grischenko V.N. Methods and tools for object-component programming // Cybernetics and System Analyses, 2003, №1, pp.39–55.

[5]. Kang K., Cohen S., Hess J., Novak W., Peterson S. Feature-oriented domain analysis (FODA) feasibility study. CMU/SEI Technical Report CMU/SEI-90-TR-21, 1990.

[6]. Berger T., She S., Lotufo R., Wąsowski A., Czarnecki K. A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering, 39(12):1611-1640, 2013. DOI: 10.1109/TSE.2013.34.

[7]. Zippel R. et al. Kconfig language. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt.

[8]. Wang H., Li Y., Sun J., Zhang H., Pan J. A semantic web approach to feature modeling and verification. Proc. of Workshop on Semantic Web Enabled Software Engineering (SWESE'05), p. 44, 2005.

[9]. Lavrischeva E.M., Slabospitskaya O.O., Koval G.I., Kolesnik A.L. Theoretical Aspects of Variability Management in Product Lines Families. Vesnik KNU, series on maths and physics (1):151-158, 2011 (in Ukrainian).

[10]. Berger T. Variability mining with LEADT. DOI TSE 2014.

[11]. Lavrischeva, E.: Formal Fundamentals of Component Interoperability in Programming. In: Cybernetics and Systems Analysis, vol. 46, no. 4, pp. 639–652. Springer, Heidelberg (2010) http://link.springer.com/article/10.1007/s10559-010-9240-z

[12]. Lavrischeva E.M., Grischenko V.N. Assembling programming. K.: Basic foundation industry Software Products. - K.: Nauk.dumka, 2009.-372 p.

[13]. Ekaterina Lavrischeva, Andrey Stenyashin, AndriyKolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software Engineering and Applications, 2014, 7, Published Online August 2014 in SciReshttp://www.scirp.org/journal/jsea

[14]. Kolesnyk A., Slabospitskaya O. Tested Approach for Variability Management Enhancing in Software Product Line. – In: ICT in Education, Research and Industrial Applications: Integration and Knowledge, Proc. 8 –th Int. Conf. ICTERI 2012, CEUR –WS.org/Vol –848, ISSN 1613 –0073, urn:nbn:de:0074 –848-8. – P. 125 –133.

[15]. Kolesnyk A.L. Model and methods development families of variants of systems. – Autoref. Disser, KNU, 2013. –22p. (ukr.)

[16]. Cohen M.B., Gibbons P.B., Mugridge W.B., Colbourn C.J. Constructing test suites for interaction testing. Proc. of 25-th Intl. Conf. on Software Engineering, pp. 38-48. IEEE, 2003. DOI: 10.1109/ICSE.2003.1201186.

[17]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC'10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[18]. C.Käster, A. Dreiling and K. Ostermann's ,Variability Mining with LEADT/- work is supported by ERC grant #203099

[19]. Grindal M., Offutt A.J., Andler S.F. Combination testing strategies: a survey. Software Testing, Verification, and Reliability, 15(3):167-199, 2005. DOI: 10.1002/stvr.319.

[20]. Lavrischeva E.M. Slabospitskya O.A. Approach to development object-component model family systems software products. Problems of Programming, 2013, №3, pp.14–26 (ukr.)

[21]. Lavrischeva E.M. Theory of object-components modeling of the programs systems. Preprint ISP RAS № 29, 2016, www.ispras.ru/preprints/docs/prep_29_2016.pdf.

[22]. Deming E. New economics for manufactures, governments and education, 1993.

[23]. Gruler A., Leucker M., Scheidemann K. Modeling and model checking software product lines. Proc. of IFIP Intl. Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS), pp. 113-131. Springer, 2008. DOI: 10.1007/978-3-540-68863-1_8.
[24]. Ekaterina M. Lavrischeva. Assemblling Paradigms of Programming in Software Engineering, 2016, 9, pp.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/ jsea.96021
[25]. Kuliamin V.V., Lavrischeva E.M., Mutilin V.S., Petrenko A.K. Verification and analysis of variable operating Systems.Trudy ISP RAN/Proc. ISP RAS, 23:359-370, 2012 (in Russian) Vol 28, Iss.3, pp.189-209. DOI: 10.15514/ISPRAS-2016-28(3)-12
[26]. Lavrischeva E.M., Petrenko A.K. Software Product Lines Modeling. Trudy ISP RAN/Proc. ISP RAS, 2016, vol 28. Iss. 6, pp. 180 -190. DOI: 10.15514/ISPRAS-2016-28(6)-4
[27]. C.Kästner, A. Dreiling, K. Ostermann, Variability Mining with LEADT, In Proc. Int’l Conf. Generative Programming and Component Engineering (GPCE), pp. 157–166. 2009.
[28]. Korotun T.M. Methods and tools testing families system in resource-limited settings (ukr.), 2005, Autoref. Dis. IC NANU, 22 pages.
[29]. Lavrischeva E.M. Software Engineering. Paradigms, Technology, CASE-tools –M: Urait, 2016. – 280 p.

Проектирование моделей вариабельности для программных, операционных систем и их семейств

[1] [2]Е.М. Лаврищева <lavr@ispras.ru>
[1]В.С. Мутилин <mutilin@ispras.ru>
[1]А.Г. Рыжов <ryzhov@ispras.ru>
[1]Институт системного программирования им. В.П. Иванникова РАН,
г. Москва, 109004, ул. А. Солженицына, 25
[2]Московский физико-технический институт,
Московская обл., г. Долгопрудный, 141701, ул. Институтская, 9

Аннотация. Сложность существующих систем и их сопровождения привела к созданию новой концепции вариабельности систем, определяемой с помощью модели характеристик (МХ). В статье мы рассматриваем подходы к формальному определению МХ и созданию на их основе вариантов программных систем (ПС), операционных систем (ОС) и их семейств. Мы рассмотрим методы создания ПС в линейке продуктов (ProductFamily/ProductLines), конвейере К.Чарнецки для сборки артефактов в пространстве проблем и решений, логико-математическое моделирование ПС из функциональных и интерфейсных элементов в объектно-компонентном методе (ОКМ), выделение функциональных элементов в ОС в МХ для генерации новых вариантов этой системы. Обсуждаются подходы формализации вариабельности существующих, новых ПС и их семейств. Определена новая концепция управления вариабельностью с помощью ОКМ. Предложены подходы к верификации МХ для ПС, ОС, их семейств и конфигурирования функциональных и интерфейсных объектов для получения новых вариантов системы. Изучены характеристики процесса тестирования

ПС, ОС и их семейств.

Ключевые слова: модель вариабельности; программная система; семейство систем; конфигурация; вариант; функциональный, интерфейсный элемент; требование; управление.

DOI: 10.15514/ISPRAS-2017-29(5)-6

For citation: Лаврищева Е.М., Мутилин В.С., Рыжов А.Г.. Проектирование моделей вариабельности для программных, операционных систем и их семейств. *Труды ИСП РАН*, том 29, вып. 5, 2017 г., стр. 93-110 (на английском языке). DOI: 10.15514/ISPRAS-2017-29(5)-6

Литература

[1]. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.

[2]. Bachmann F., Clements P. Variability in software product lines. CMU/SEI Technical Report CMU/SEI-2005-TR-012, 2005.

[3]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC’10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[4]. Лаврищева Е.М., Грищенко В.Н. Методы и средства объектно-компонентного программирования. Кибернетика и системный анализ. 2003.-№1, с. 39-55.

[5]. Kang K., Cohen S., Hess J., Novak W., Peterson S. Feature-oriented domain analysis (FODA) feasibility study. CMU/SEI Technical Report CMU/SEI-90-TR-21, 1990.

[6]. Berger T., She S., Lotufo R., Wąsowski A., Czarnecki K. A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering, 39(12):1611-1640, 2013. DOI: 10.1109/TSE.2013.34.

[7]. Zippel R. et al. Kconfig language. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt.

[8]. Wang H., Li Y., Sun J., Zhang H., Pan J. A semantic web approach to feature modeling and verification. Proc. of Workshop on Semantic Web Enabled Software Engineering (SWESE’05), p. 44, 2005.

[9]. Лаврищева Е.М., Слабоспицькая О.А., Коваль Г.И., Колесник А.А. Теоретические аспекты управления вариабельностью в семействах ПС. Весник КНУ, серия физ.–мат. наук, 2011, № 1, стр. 151-158.

[10]. Berger T. Variability mining with LEADT. DOI TSE 2014.

[11]. Lavrischeva, E.: Formal Fundamentals of Component Interoperability in Programming. In: Cybernetics and Systems Analysis, vol. 46, no. 4, pp. 639–652. Springer, Heidelberg (2010) http://link.springer.com/article/10.1007/s10559-010-9240-z

[12]. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. К.: Наук. Думка, 2009, 371 с.

[13]. Ekaterina Lavrischeva, Andrey Stenyashin, Andriy Kolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software

Engineering and Applications, 2014, 7, Published Online August 2014 in SciReshttp://www.scirp.org/journal/jsea

[14]. Kolesnyk A., Slabospitskaya O. Tested Approach for Variability Management Enhancing in Software Product Line. – In: ICT in Education, Research and Industrial Applications: Integration and Knowledge, Proc. 8 –th Int. Conf. ICTERI 2012, CEUR –WS.org/Vol –848, ISSN 1613 –0073, urn:nbn:de:0074 –848-8. – P. 125 –133.

[15]. Колесник А.Л. Модели и методы разработки семейств вариантных программных систем.-Автореф.- КНУ.- 2013. 22 с.

[16]. Cohen M.B., Gibbons P.B., Mugridge W.B., Colbourn C.J. Constructing test suites for interaction testing. Proc. of 25-th Intl. Conf. on Software Engineering, pp. 38-48. IEEE, 2003. DOI: 10.1109/ICSE.2003.1201186.

[17]. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC'10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[18]. C.Käster, A. Dreiling and K. Ostermann's ‚Variability Mining with LEADT/- work is supported by ERC grant #203099

[19]. Grindal M., Offutt A.J., Andler S.F. Combination testing strategies: a survey. Software Testing, Verification, and Reliability, 15(3):167-199, 2005. DOI: 10.1002/stvr.319.

[20]. Лаврищева Е.М., Слабоспицкая О.А. Подход к построению объектно-компонентной модели семейства программных продуктов. Проблемы программирования, 2013, №3, стр. 14–26 (укр.).

[21]. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН № 29, 2016. http://www.ispras.ru/preprints/docs/prep_29_2016.pdf.

[22]. Deming E. New economics for manufactures, governments and education, 1993.

[23]. Gruler A., Leucker M., Scheidemann K. Modeling and model checking software product lines. Proc. of IFIP Intl. Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS), pp. 113-131. Springer, 2008. DOI: 10.1007/978-3-540-68863-1_8.

[24]. Ekaterina M. Lavrischeva. Assemblling Paradigms of Programming in Software Engineering, 2016, 9, pp.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/ jsea.96021

[25]. Кулямин В.В. Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды ИСП РАН. Том 28. вып. 3, стр. 189-209. DOI: 10.15514/ISPRAS-2016-28(3)-12

[26]. Лаврищева Е.М. Петренко А.К. Моделирование семейств программных систем. Труды ИСП РАН, 2016, том 28. вып. 6, стр. 180 -190. DOI: 10.15514/ISPRAS-2016-28(6)-4

[27]. C. Kästner, A. Dreiling, K. Ostermann, Variability Mining with LEADT, In Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE), pp. 157–166. 2009.

[28]. Korotun T.M. Methods and tools testing families system in resource-limited settings (ukr.), 2005, Autoref. Dis. ICNANU, 22 pages.

[29]. Лаврищева Е.М. Программная инженерия. Парадигмы, Технологии, CASE-средства программирования.2 изд. М: Юрайт, 2016, 280 с.