ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2009, Vol. 38, No. 3

MEASURING COMPLEXITY OF DOMAIN MODELS REPRESENTED BY FEATURE DIAGRAMS

Vytautas Štuikys, Robertas Damaševičius

Software Engineering Department, Kaunas University of Technology
Studentų St. 50, LT–51368 Kaunas, Lithuania
e-mail: vytautas.stuikys@ktu.lt,damarobe@soften.ktu.lt

Abstract. Feature models represented by Feature Diagrams (FDs) prevail in the software product line approach. The product line approach and FDs are used to manage variability and complexity of software families and to ensure higher quality and productivity of product development through higher-level feature modeling and reuse. In this paper we, first, analyze the properties of feature models. Then, combining some properties of FDs with ideas of Miller's, Metcalfe's and Keating's works, we propose three FD complexity measures. The first measure gives boundaries to estimate cognitive complexity of a generic component to be derived from the feature model. The second measure describes structural complexity of the model expressed through the number of adequate sub-trees of the given model. The third measure estimates total cognitive and structural complexity of the model. To validate the introduced measures, we present a case study with three feature models of a varying complexity.

Keywords: Feature model, feature diagram, complexity measures, cognitive complexity, structural complexity.

1. Introduction

Complexity is the inherent property of systems to be designed. The need for managing the complexity issues is constantly growing because of the fact that systems *per se* are becoming more and more complex mainly due to technological advances, increasing user requirements and market pressure. Complexity management can help to increase quality and understandability of developed products, decrease the number of design errors [1] and shorten their development time. Managing complexity means, firstly, knowing how to measure it. Complexity measures allow reasoning about system structure, understanding the system behaviour, comparing and evaluating systems or foreseeing their evolution.

Researchers and practitioners struggle with the complexity problem for more than three decades. Software engineers have long seen complexity as a major factor affecting design quality and productivity. The efforts to manage complexity have resulted in the introduction and studies of such general principles as *separation of concerns*, *information hiding*, *system decomposition*, and *raising abstraction level* in a system design [2]. On the other hand, new design methodologies, which implement those principles combined with various design techniques (e.g., object-oriented design, generative programming [3]), have emerged and are further evolving. The evident example is Product Line Engineering (PLE) [4], which shifts from the design of a single system to the design of a family of related systems. The methodology widely exploits the model-driven approach, where at the focus are high-level *domain models*.

The domain model describes some fundamental properties of a domain that are to be implemented by software. These properties are: commonality, variability, specificity, and various relationships among these groups of features [2, 4, 5]. These properties are expressed through the notation, which is commonly accepted in the context of PLE – Feature Diagrams (FDs). As currently the model-driven approach prevails in the development of systems (not only in PLE, but also in a much wider context), it is not enough to deal with the complexity issues at the stand-alone program (system) level only. What is needed is to consider the complexity problem at a higher abstraction level too, i.e., to focus on the complexity of high-level domain models. We see the definition of such measures as a meta-modeling activity.

The aim of this paper is to analyze feature models in order to devise their complexity measures. As the scope of the problem is wide, we restrict ourselves to the analysis of feature models that contain a high degree of variability (actually expressing generic aspects of the model) and represent some part of a super-domain model. So far only one study [6] has

considered non-functional properties of product lines such as complexity, but we were unable to find any research on the estimation of feature model complexity. Our contribution is three complexity measures (metrics) to evaluate the complexity of generic program models represented using FDs. The measures use some properties of FDs combined with ideas from the Miller's [7], Metcalfe's [8] and Keating's [9] works.

The paper is organized as follows. Section 2 analyses the related works. Section 3 motivates the problem and provides an example. Section 4 describes backgrounds for evaluating complexity of models represented by FDs and presents measures for quantitative calculation of complexity. Section 5 presents three examples of complexity calculation. Section 6 presents the overall evaluation of the results. Finally, Section 7 formulates conclusions and states problems for further research.

2. Related works

We classify the related works into two categories: 1) works on complexity analysis and measurement of programs and their high-level models such as UML models, business process models, etc.; and 2) works on feature models and feature diagrams as a specific kind of high-level models.

Complexity measures at a program model level were at the focus of researchers for a long time. Though there are many metrics, *Cyclomatic complexity* is one of the most widely accepted *static* software metrics [10]. Proposed by McCabe in 1976 [11], it directly measures the number of linearly independent paths through a program's source code from entrance to each exit. It is intended to be independent of language and language format. Other metrics bring out other facets of complexity, including structural and computational complexity. For example, *Halstead complexity measures* [12] identify algorithmic complexity measured by counting operators and operands; *Henry and Kafura metrics* [13] indicate coupling between modules (parameters, global variables, calls); *Troy and Zweben* [1] metrics evaluate complexity of program's structure.

Since the arrival of model-driven development, complexity measures were introduced at the domain model level, too. Different measures are proposed to evaluate structural and cognitive complexity of UML Use Case diagrams [14] and Class diagrams [15-17], State-Chart diagrams [18], Entity-Relationship diagrams [19], Conceptual Schemas [20], Petri Net specifications [21]. Kim and Boldyreff [22] have defined a set of 27 metrics to measure various characteristics including complexity of UML models.

The complexity of business process models (BPM) is assessed in [23, 24]. BPM models have some similarity to FDs, because they have similar types of connectors: AND, OR, and XOR. Cardoso [25] defines Control-Flow Complexity (CFC) of business processes as the number of mental states (or possible decisions in a flow) that have to be considered when a designer develops a process. Other researchers [26-28] propose using graph complexity metrics, such as Coefficient of Network Complexity (CNC), Complexity Index (CI), Restrictiveness Estimator (RT), and the number of trees in a graph, to evaluate business processes. Mendling *et al.* [29] describe, analyze and validate experimentally 28 business process metrics (such as size, density, structuredness, coefficient of connectivity, average connector degree, control flow complexity, etc.).

Product-line engineering (PLE), which has emerged in recent years as a design paradigm aiming to ensure higher quality and productivity in the development of software systems [4, 5], widely employs feature models. Such models, which represent a set of domain-related features and their relationships, are usually described using Feature Diagrams (FDs). Though they were introduced in the context of FODA (Feature-Oriented Domain Analysis) yet in 1990 [30], the evolution of the FD notation still continues [31]. FDs are seen as higher-level models that allow describing specifications to implement generative approaches within the PLE concept (e.g., generative programming [3], aspect-oriented programming [32] or meta-programming [34, 35]). FD is a specific model that specifies not a single program, but a family of the related program instances represented at a high-level of abstraction. Though a FD and a program graph has some common properties, the complexity problem of FDs should be considered separately, first of all taking into account specific properties of FDs and, of course, the appropriate program complexity and other measures, such as proposed in [9].

3. Problem statement and a motivating example

3.1. Some preliminary remarks

As system designs evolve under pressure and demands for better quality, higher functionality and shorter time-to-market, the growth of complexity has direct impact on design methods, approaches and paradigms. Complexity is the intrinsic attribute of systems and processes through which systems are created. One way to manage design complexity is to enhance reuse in the context of PLE, where requirements may evolve. What is happening when we need to extend the scope of requirements beyond one system/component or beyond a family of related systems/components, if there is some prediction on their possible usage in a wider context? It is easy to predict intuitively: the models we need to deal with are becoming more and more complex. But to which limits we can let complexity of models grow in terms of requirements prediction, implementation difficulties and how we need to manage this complexity at a higher abstraction level? The first task is to understand

the complexity issues and to learn to measure the complexity quantitatively. The motivating example we present below provides more details for better understanding of the problem.

3.2. Motivating example of a feature model

Feature diagrams are commonly accepted models to represent domain artifacts at a higher abstraction level when a design is based on the product line methodology. Feature models are created using domain analysis methods (e.g., FODA, FORM, etc. [2]). The syntax of FDs can be easily learned from the example (see Figure 1 and sub-section 3.2) and semantics can be learned from the properties described in sub-section 3.3 (for deeper knowledge, see [31]).

As a motivating example, we have selected the model that describes features and their relationships of the *homogeneous gate* domain. The domain is well defined because it is based on the first-order logic (FOL), called also Boolean algebra. The term *gate* means that it contains not only logical (functional) features, descriptive features (e.g., behavioral, structural) and representation features (e.g., HW description languages, such as VHDL or SystemC [33]) and various relationships, but also physical features, such as technology, area, delay and energy consumption. Figure 1 presents a simplified model that defines the *representational* and *functional* aspects only. From the methodological view, the model also serves as a tutorial for understanding and learning syntax and semantics of FDs.

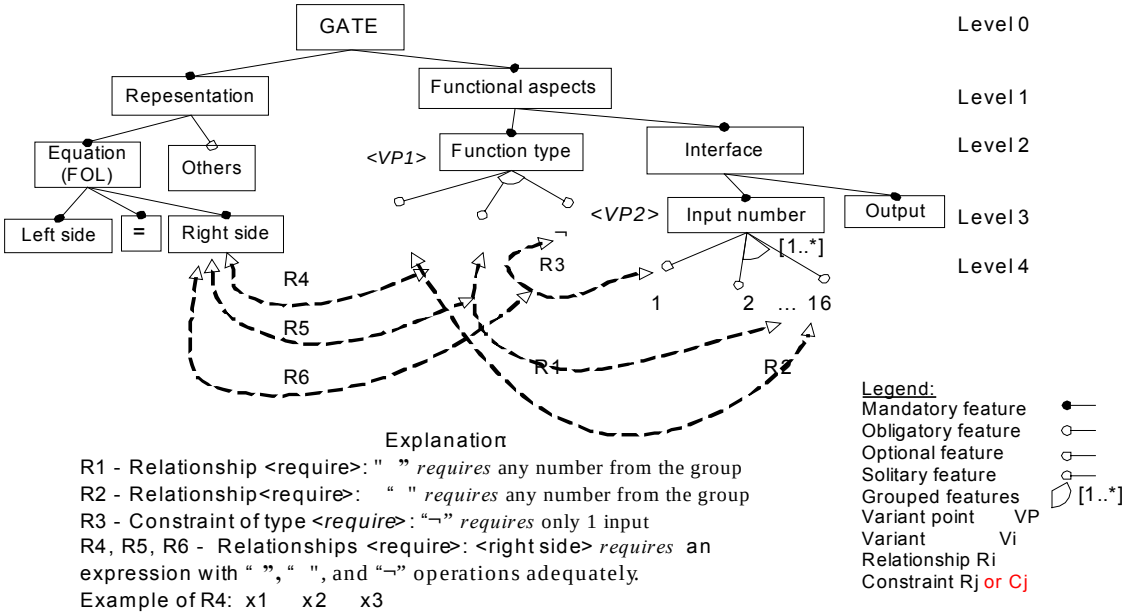


Figure 1. Homogeneous GATE feature model based on representational and functional aspects only

We define basic terms related to FDs below. Kang et al. define feature as "end-user visible characteristic of a system or a key characteristic of a concept that is relevant to some stakeholder" [4]. We define feature as an externally visible characteristic that is relevant to the given designer's context (we can allow the context to be changed [36, 37]), but here we consider that context is constant). Feature diagram is a connected graph, where boxes (nodes, leaves) represent features and edges represent various kinds of relationships among features. We accept that relationships are of the type, which are expressed through relations of the propositional logic. There are three types of features: mandatory (boxes with the black circle above), optional and alternative (both are denoted as boxes with the white circle above).

Mandatory features express commonality of the concept, whereas optional and alternative features express variability. Features may appear either as a *solitary feature* or *in groups*. If all mandatory features in the group are derivates from the same parent in the *parent-child* relationship, there is the *and*–relationship among those features (e.g., at level 1 or 2, see Figure 1). An optional feature may be included or not if its parent is included in the model. Alternative features, when they appear in groups as derivates from the same parent, may have the following relationships: *or, xor* (filled arc in Figure 2), *case* (arc in Figure 1), etc. For more advanced types of alternative features as "views on ontology", see [38].

There are also three types of *structural* relationships: *parent-child* (already has been introduced), relationships *among nodes* (e.g., *R1, R2* in Figure 1) and *constraints*. In each category there may be various *semantic* relationships, such as *<expression>* (*R2*), *<equivalence>* (*R1*), *<algebraic dependency>* [37], etc. in the category of relationships among terminals. For example, constraints among terminals may have

the following types: *<require>* (*R3*), *<implication>*, *<mutual exclusive>*, etc. (this kind don't appear in models Figure 1 and 2).

3.3. Some properties of feature models

We formulate the basic properties of feature models, which have influence on measuring complexity. Since the example (Figure 1) is too simple, we extend it with new aspects such as a *language* and *function notation* (see Figure 2). The introduction of the new aspect in one branch (at level 1, see Figure 1 and 2) may cause the need of introducing additional features, variant points, variants, and as a result, new relationships are to be taken into account. It is clear – the complexity of the model is growing too with any extension of the domain – and we can also see some restrictions or limitation of the introduced graphical notation as stated below. If the tree-like structure, i.e., feature and parent-child relationships and partial constraints work well, some difficulties arise with obtaining and representing other kinds of relationships. This happens because of: 1) FDs lack of a mechanism for representing more complicated relationships (which usually lead to domain ontology [37, 38]; 2) graphical representation of a large amount of relationships by connecting leaves diminishes readability and clarity of the graph; 3) some features may describe other domains (e.g., syntax of VHDL, see Figure 2), which are much wider than the domain at hand, and the process of decomposition should be restricted.

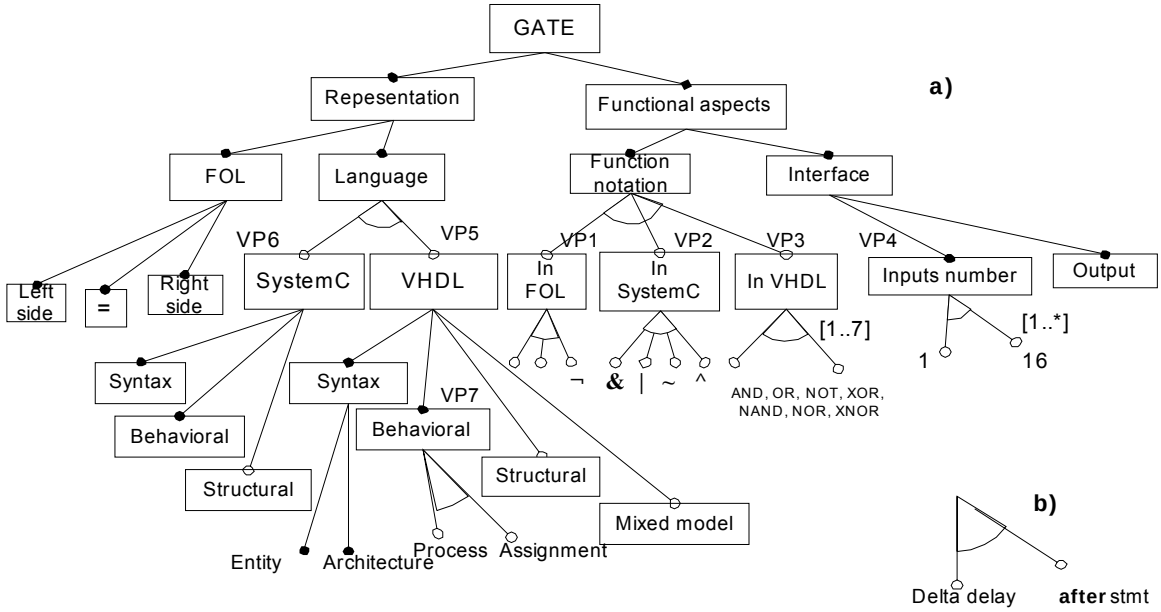


Figure 2. The extended feature model (a) of the domain GATE and possibility for further extension of features 'process', 'structural' and 'Mixed model' by 'delta delay' or using '**after** stmt' (b)

The latter has the following implication: we need to let not only the *explicit representation* (the usual intension of the model) but also the *implicit representation* of some features and relationships (e.g., aiming to simplify the model by information hiding). For example, such a sub-feature as *variable* of the feature *right side* (see Figure 1) is not defined explicitly because it is a lower-level feature and its explicit specification should be postponed till implementation.

I. The feature diagram-based *domain model* is a *connected graph* consisting of two subgraphs:

a) A tree-like subgraph in which the root represents the domain itself, the intermediate nodes represent a set of compound features (some of them may be variant points), the *leaves* represent feature variants (if derived from the same variant point) and edges represent the *parent-child relationships* among features, variant points and their variants;

b) A subgraph (it may be disconnected, see Figure 1), in which nodes are a *subset* of *leaves* derived from different parents, and *edges* between the *subset nodes* represent the *overlapping relationships* and constraints induced by the inherent properties of the domain (for simplicity reasons the subgraph is omitted in Figure 2).

II. Model can be rewritten as *n*-level structure (see Figure 1), where *n* is the longest path from the root to the node at the level *n*. If a node at level *i*-1 ($1 < i < n$) represents a variant point, then nodes at level *i* are terminals or variants.

III. Overlapping relationships may represent functional, structural and other kind of relationships among the leaves derived from different parents. Among overlapping relationships there is a specific type of relationships which represent *constraints*. Constrains may have such sub-types as "**A** *requires* **B**", "**C** and **D** *are mutually exclusive*", where **A**, **B**, **C** and **D** are feature variants.

IV. A domain model expresses such aspects as commonality, variability and specificity (if any). The

terms are described as follows. These aspects have quantitative measures. Commonality is the number of mandatory features in a feature model. Variability is a compound of the number of variant points, the number of variants and overlapping relationships including constrains (see property IX, for more details).

V. Features and relationships are represented in a FD in two ways: explicitly and/or implicitly. The explicit representation is more powerful and, as a rule, it is used for more important features (e.g., variant points that pre-define variability and relationships). The implicit representation is less powerful. It can be used to simplify the representation in the case when either it is clear from the context that a given feature/relationship should be taken into account in the evaluation. Implicit representation is also used for restricting the scope of the model.

VI. The *scope* of the feature model is a restrictive attribute that outlines the boundaries of the domain under consideration in the given circumstances. The scope of the model means all aspects which we want to include in the model. The scope depends on multiple factors: goal, context (e.g., external factors such as user requirements), domain itself and, in general, abilities of the domain analyzer to understand the domain and to grasp its properties. The scope is more relevant to the domain; while complexity is more relevant to evaluation of the domain model.

VII. Feature diagram with variant points and variants represents a family of related component instances specified at a higher abstraction level as sub-models in terms of features and relationships. A particular instance (a sub-model) in the model is represented as a sub-tree with the following properties: a) each sub-tree contains all non-leaves and only one leaf (variant) for each variant point: b) all edges that contain the above mentioned nodes. For example, a FD in Figure 1 has 31 sub-trees (15 with AND (**∧**), 15 with OR (**∨**) and 1 with NOT (¬)).

VIII. As influential factors (goal, context, etc.) evolve over time, the scope and the model representation can be extended adequately. Thus we can speak about evolving models even for the same domain objects. It is reasonable to predict that evolution adds complexity in the model. This property can be easily learned from the examples (cf. Figure 1 and 2).

IX. *Commonality* in the model is mandatory features (that are not variant points) and their parent-child relationships. For example, node <*equation*> and its edges <*left side*>, <.= > and <*right side*> , as well as <*interface*> consisting of <*inputs*> and <*output*> in Figure 1 are two different kinds of commonality. Variability in the model is variant points, variants (see VP1 and VP2 in Figure 1) and explicit relationships among variants. Constraints as a specific kind of relationships (it can be treated as domain specificity, e.g., "NOT requires only one input") diminish variability in some way.

X. Introducing new features to the model (e.g., as a result of the extension of domain scope aiming to build an evolutionary model) may result in appearance of *new variants*, the need of *clustering some* variants and even introduction of *external features and relationships* from a larger domain, which is a super-domain of the given one. This property can be understood by comparing Figure 1 and 2 and introducing new features such as area, energy, delay in Figure 2 (these features are not shown and are regarded as being defined implicitly).

4. Backgrounds for evaluating complexity of feature models

There are two different views on complexity [23]: complexity as "*difficulty to test*" (i.e., number of test cases needed to achieve full path coverage), and complexity as "*difficulty to understand a model*". The latter is also known as *cognitive complexity* of a model. Cardoso *et al.* [26] also identify different types of complexity: computational complexity, psychological (cognitive) complexity, and representational complexity. Cognitive complexity focuses on the analysis of how complicated a problem is from the perspective of the person trying to solve it. Cognitive complexity is related to short-term memory limitations, which vary depending on the individual and on what kind of information is being retained [39]. For software designers, the ability of coping with complexity of a domain model is a fundamental issue, which influences the quality of the final product. High cognitive complexity of a model leads to a higher risk of making design errors and may lead to lower than required quality of a developed product, such as decreased maintainability. We claim that the properties (such as structural complexity and size) of a feature model represented using FD have an impact on its cognitive complexity.

In this context, it is useful to have a boundary for cognitive complexity. We rely on Miller's early work [7] stating that human beings can hold 7 (+/-2) chunks of information in their short-term memory at one time. We also use the rule of Keating, which is based on the Miller's work as applied to design domain: "*The number of modules at any level of hierarchy must be 7 +/- 2*" [9]. Our empirical rule (Rule 1) for the boundary of cognitive complexity as applied to the feature model is as follows:

Rule 1. *The cognitive complexity of a FD is calculated as the number of variant points in a FD. The number of variant points in a FD must be 7 +/- 2, if a designer wants to avoid consequences of high cognitive complexity. If the number of variant points is fewer than 5, the value of the model may be diminished due to the decreasing granularity level and too much information hiding. If the number of variant points is more than 9, the user needs to decompose the model into* ***parts*** *or* ***levels*** *in order to remain within the limits of cognitive complexity.*

Rule 2. *The cognitive complexity of a FD is calculated as the maximal number of levels in a feature hierarchy or the maximal number of parts (graph leaves) in each level of a hierarchy.*

Rule 3. *The structural complexity of a feature model with variant points is evaluated by the number of sub-trees with property VII (each variant point has only one selected variant). Each sub-tree (sub-model) is derived from the initial feature diagram as a generic model for a given domain.*

Rule 3 describes the structural (representational) complexity of a feature model, i.e. the ability of a FD to represent different product instances of a product line. Rule 3 has some correlation with the *cyclomatic number*, the well known measure for evaluating the complexity of a program [10]. Each path in a program graph correlates to the adequate sub-tree in the feature diagram since the realization of the sub-tree can be seen as a program (path) with the syntax rules for correct implementation of a particular product instance.

For example, the generic domain model (see Figure 1) has 31 different sub-trees which, when realized, gives 31 logical equations of the following type:

$$y = \neg x; \quad (1)$$

$$y = x1 \wedge x2;\ y = x1 \wedge x2 \wedge x3;\ \ldots$$
$$y = x1 \wedge x2 \wedge \ldots \wedge x16; \quad (2)$$

$$y = x1 \vee x2;\ y = x1 \vee x2 \vee x3;\ \ldots$$
$$y = x1 \vee x2 \vee \ldots \vee x16; \quad (3)$$

Based on the empirical research and practical implementations [40], the cyclomatic complexity has the following boundaries: from 1 to 10, the program is simple; from 11 to 20, it is slightly complex; from 21 to 50, it is complex; and above 50 it is over-complex (untestable).

Rule 4 we describe below (see Eq. (5)) states how the cognitive complexity and the structural complexity should be combined. It is based on the Metcalfe's empiric law and Keating's adaptation of the law for the complexity evaluation of a design partitioning [9]. Metcalfe's law states that the "power" of a network is equal to the square of the nodes on it, and the "value" of the network is equal to the square of the edges on the network. The Keating's measure is

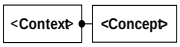
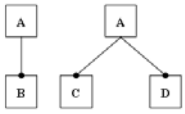
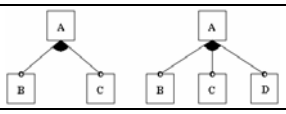
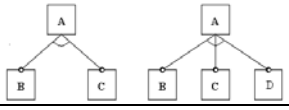
$$C = M^2 + I^2, \quad (4)$$

where C is design complexity, M is the number of modules in a design, and I is the number of interfaces among modules.

As design complexity can be presented as a graph in which nodes represent modules and edges represent interfaces (Keating's model), we can apply this complexity measure to feature models. What is different in our case is that our graph (feature diagram) has different properties: nodes and edges play different roles (see property IX in sub-section 3.3.2) and should have different cognitive weights.

We define *cognitive weight* of a feature as the degree of difficulty or relative time and effort required to comprehend it, and *total cognitive weight* of a feature model represented as a feature diagram is the sum of the cognitive weights of its graph elements. Therefore, following Shao and Wang [41], who define the weight of sequential structure as 1, the weight of branching (if-else) as 2, and the weight of case selection as 3, we also introduce some cognitive weights to Eq. (4) (see Table 1).

Table 1. Cognitive weights of FD elements

Feature Diagram element	Structure	Cognitive weight
Node (feature)		1
Mandatory feature relationship (and-relationship)		1
Optional feature relationship (or-relationship)		2
Alternative feature relationship (case-relationship)		3
Groupings of relationships (cardinality)	**[1..*]**	3
Relationships among nodes and constraint relationships (<requires>, <excludes>)		3

Note. In the paper, we consider such FDs, whose relationships can be expressed using the propositional logic only.

Rule 4. *The compound complexity measure of a feature diagram (FD) is estimated by equation (5):*

$$Cm = F^2 + (Rand^2 + 2Ror^2 + 3Rcase^2 + 3Rgr^2 + 3R^2)/9, \quad (5)$$

where *Cm* is the compound complexity measure, *F* is the number of features (variant points and variants), *Rand* is the number of mandatory relationships, *Ror* is the number of optional relationships, *Rcase* is the number of alternative relationships, *Rgr* is the number of relationship groupings, *R* is the number of relationships among nodes including constrains, and the division coefficient is the sum of cognitive weights for equalizing the role of relationships.

The equalization is based on the criticism of Metcalfe's law [8, 42] stating that the value of the network in terms of linking users (i.e., complexity) is not proportional (equal) to the square of the number of its edges but less than this value, as it is identified by the following inequality:

$$(Rand^2 + 2Ror^2 + 3Rcase^2 + 3Rgr^2 + 3R^2)/9 < (Rand + Ror + Rcase + Rgr + R)^2. \quad (6)$$

5. Example of complexity calculation

We present an example how the complexity of feature models can be calculated. Our aim is to illustrate how to use Rules 1 – 4 in practice when we try to evaluate a static (context-independent) well-defined domain feature model, and show the difficulties of the evaluation in the case when the evolutionary growth of feature diagrams is to be taken into account. We use feature models from Figure 1, Figure 2 and extended Figure 2 to estimate their complexity. The extended model (see Figure 2, a and b) contains three extra variant points added to features *Process, Assignment* and *Mixed model*, each meaning *Delay* between inputs and output and each having two variants: 1) *delta* delay and 2) *explicit delay* (described with the statement **after** <*delay constant*> <*delay unit*> in VHDL). The results are presented in Table 2.

Table 2. Estimated complexity of analyzed feature models

Feature model	Cognitive complexity (acc. to Rules 1, 2)	Structural complexity (acc. to Rule 3)	Compound complexity (acc. to Rule 4)	Estimation of compound complexity	Explanation
1 (Fig. 1)	2 (Rule 1) 5 (Rule 2)	31	990	Low	*F* = ***31***; *Rand* = ***10***; *Ror* = ***1***; *Rcase* = ***4***; *Rgr* = ***1***; *R* = ***6***
2 (Fig. 2, a)	7 (Rule 1) 6 (Rule 2)	487	3484.8	Complex	*F* = ***58***; *Rand* = ***17***; *Ror* = ***3***; *Rcase* = ***16***; *Rgr* = ***2***; *R* = ***0****
3 (Ext. Fig. 2, a and b)	10 (Rule 1) 7 (Rule 2)	760	4264.8	Over-complex	*F* = ***64***; *Rand* = ***17***; *Ror* =***3***; *Rcase* = ***20***; *Rgr* = ***2***; *R* = ***0****

***Note**. In models 2 and 3, it is assumed that *R= 0* because those relationships are not shown in FDs (although actually they exist since the models are derived from the model 1 (see Fig. 1)).

6. Summary, discussion and evaluation of the results

Quantitative evaluation of the complexity of models is a very important task due to many reasons: 1) complexity in system design is continuously growing, and as a result, there is a great need to manage complexity; 2) designs are moving towards a higher abstract level, thus the model-driven development is further strengthening its position; 3) assessment of the complexity of the developed software systems in the early stages of the software life-cycle allow to make cost-effective changes to the developed systems; 4) though software has many complexity measures (e.g., number of code lines, cyclomatic number, psychological complexity, etc.), the straightforward use of those measures is not always relevant at the model level; 5) how we can reason about the introduction of a new abstraction level objectively (in order to manage the complexity and, e.g., to avoid over-generalization in component design [43]) without having quantitative measures?

The task to deal with the complexity of models is hard because of a large variety of model types used to describe the models. We focus on a specific type of models described by Feature Diagrams (FDs), which are very useful in the context of product line engineering and the use of generative technologies for implementing product lines. Due to the number of factors that contribute to the complexity of a FD, we cannot identify a single metric that measures all aspects of a feature model's complexity. This situation is well known from the measurements of program source code complexity. A common solution is to use different measures within a metrics suite. Each individual measure can evaluate one aspect of the complexity, and together they can provide a more accurate estimation of complexity.

In this paper, we have proposed three measures for evaluating the complexity of FDs. The measures are

based on some properties of FDs, the empiric laws of Miller and Metcalfe as well as on rules of Keating [9]. The first measure evaluates the boundaries of cognitive complexity, which are expressed through the "*magic seven*" property applied to variant points in the FD. The second measure evaluates structural complexity expressed through the quantitatively identifiable number of *adequate* sub-trees in the FD. The measure correlates with the cyclomatic number that is used to evaluate program complexity. The third measure evaluates both the cognitive and structural aspects of complexity.

7. Conclusions and future work

The introduced complexity measures of feature models described using Feature Diagrams allow reasoning about the structure and behaviour of the system to be modelled at a higher abstraction level, allow comparing and evaluating system models or the complexity of their transformation into lower-level representation (e.g., into generic programs). The measures also allow to reason about the granularity level, important reuse characteristics that are difficult to express quantitatively, and generic programs (components) to be derived from the feature model. As complexity is the inherent system property with multiple aspects, it is difficult to devise a unified measure reflecting all aspects of the model. The proposed complexity measures reflect different views on complexity, and enable to evaluate the design complexity at the model level. Though the presented case study supports theoretical assumptions, more empirical research is needed in order to better evaluate the measures and to reason about their value with a larger degree of certainty.

References

[1] **D.A. Troy, S.H. Zweben.** Measuring the Quality of Structured Designs. *Journal of Systems and Software*, *Vol*. 2, 1981, 113-120.

[2] **J. Coplien, D. Hoffman, D. Weiss.** Commonality and Variability in Software Engineering. *IEEE Software*, *Vol*. 15(6), 1998, 37-45.

[3] **K. Czarnecki, U. Eisenecker.** Generative Programming: Methods, Tools and Applications. *Addison-Wesley*, 2001.

[4] **K. Kang, J. Lee, P. Donohoe.** Feature-oriented product line engineering. *IEEE Software*, *Vol*. 19(4), 2002, 58-65.

[5] **P. Clements, L. Northrop.** Software Product Lines: Practices and Patterns. *Addison-Wesley*, 2002.

[6] **N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, G. Saake.** Measuring non-functional properties in software product lines for product derivation. *Proc. of* 15*th Asia-Pacific Software Engineering Conference* (*APSEC* 2008), *Beijing, China December* 3-5, 2008,187-194.

[7] **G. Miller.** The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review*, *Vol*. 63(2), 1956, 81-97.

[8] **G. Li.** Economic sense of Metcalfe's Law. *Proc. of* 17*th Int. World Wide Web Conference* (*WWW* 2008), *April* 21-25, 2008, *Beijing, China*.

[9] **M. Keating.** Measuring Design Quality by Measuring Design Complexity. *Proc. of the* 1*st Int. Symp. on Quality of Electronic Design* (*ISQED* 2000), *San Jose, California, March* 20-22, 2000, 103-108.

[10] **Software Engineering Institute (SEI).** Cyclomatic Complexity. *In Software Technology Roadmap*, 2006. *www.sei.cmu.edu/str/descriptions/cyclomatic_body.html*.

[11] **T.J. McCabe.** A Complexity Measure. *IEEE Transactions on Software Engineering*, *Vol*. se-2, *No*. 4, 1976, 308-320.

[12] **M.H. Halstead.** Elements of Software Science. *New York: Elsevier*, 1977.

[13] **S.M. Henry, D.G. Kafura.** Software Structure Metrics Based on Information Flow. *IEEE Trans. Software Eng.* 7(5), 1981, 510-518.

[14] **M. Marchesi.** OOA metrics for the Unified Modeling Language. *Proc. of Second Euromicro Conference on Software Maintenance and Reengineering* (*CSMR*'98), *Florence, Italy,* 1998, 67.

[15] **T. Yi, F. Wu, C. Gan.** A comparison of metrics for UML class diagrams. *ACM SIGSOFT Software Engineering Notes, Vol*. 29(5), 2004, 1–6.

[16] **M. Genero-Bocco, M. Piattini, C. Calero.** A Survey of Metrics for UML Class Diagrams. *Journal of Object Technology* 4(9), 2005, 59-92.

[17] **A. Zivkovic, M. Hericko, B. Brumen, S. Beloglavec, I. Rozman.** The Impact of Details in the Class Diagram on Software Size Estimation. *INFORMATICA,* 16(2), 2005, 295-312.

[18] **M. Genero, D. Miranda, M. Piattini.** Defining Metrics for UML Statechart Diagrams in a Methodological Way. *In M.A. Jeusfeld, O. Pastor (Eds.), Proc. of Conceptual Modeling for Novel Application Domains, ER* 2003 *Workshops,* Chicago, *IL, USA, October* 13, LNCS 2814, *Springer*, 2003, 118-128.

[19] **M. Genero, L. Jiménez, M. Piattini.** Measuring the Quality of Entity Relationship Diagrams. I*n A.H.F. Laender, S.W. Liddle, V.C. Storey (Eds.), Proc. of* 19*th Int. Conf. on Conceptual Modeling, ER* 2000, *Salt Lake City, Utah, USA, October* 9-12, 2000. LNCS 1920, Springer, 2000, 513-526.

[20] **S.S. Cherfi, J. Akoka, I. Comyn-Wattiau.** Perceived vs. Measured Quality of Conceptual Schemas: An Experimental Comparison. *In J.C. Grundy, S. Hartmann, A.H. F. Laender, L.A. Maciaszek, J.F. Roddick (Eds.), Proc. of the* 26*th Int. Conf. on Conceptual Modeling, ER* 2007. *Auckland, New Zealand, November* 5-9, 2007, 185-190.

[21] **S. Morasca.** Measuring Attributes of Concurrent Software Specifications in Petri Nets. *Proc. of the* 6*th Int. Symposium on Software Metrics*, *Boca Raton, Florida, November* 4-6, 1999, 100–110.

[22] **H. Kim, C. Boldyreff.** Developing software metrics applicaple to UML models. *In* 6*th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering* (*QAOOSE* 2002), *Malaga, Spain, June* 2002.

[23] **R. Laue, V. Gruhn.** Complexity metrics for business process models. *In W. Abramowicz and H. C. Mayr (Eds.)*, *Proc. of* 9*th Int. Conf. on Business Information Systems* (*BIS* 2006), *LNI* 85, 2006, 1-12.

[24] **I. Vanderfeesten, J. Cardoso, J. Mendling, H.A. Reijers, W.M.P. van der Aalst.** Quality Metrics for Business Process Models. *In L. Fischer (ed.), BPM and Workflow Handbook* 2007, *Future Strategies, May* 2007, 179–190.

[25] **J. Cardoso.** Process control-flow complexity metric: An empirical validation. *Proc. of IEEE Int. Conf. on Services Computing (IEEE SCC* 06), *Chicago, USA, September* 18-22, 2006, 167-173.

[26] **J. Cardoso, J. Mendling, G. Neumann, H.A. Reijers.** A Discourse on Complexity of Process Models. *In J. Eder, S. Dustdar (Eds.), Proc. of Business Process Management BPM* 2006 *Workshops, Vienna, Austria, September* 4-7, 2006. *LNCS* 4103, *Springer*, 2006, 117-128.

[27] **A.M. Latva-Koivisto.** Finding a complexity measure for business process models. *Research Report, Helsinki University of Technology, Systems Analysis Laboratory*, 2001.

[28] **I. Vanderfeesten, H.A. Reijers, J. Mendling, W.M.P. van der Aalst, J. Cardoso.** On a Quest for Good Process Models: The Cross-Connectivity Metric. *Proc. of* 20*th Int. Conf. on Advanced Information Systems Engineering CAiSE* 2008, *Montpellier, France, June* 16-20, 2008. *LNCS* 5074, *Springer*, 2008, 480-494.

[29] **J. Mendling, G. Neumann, W.M.P. van der Aalst.** Understanding the Occurrence of Errors in Process Models based on Metrics. In R. Meersman, Z. Tari (Eds.): *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007,* Vilamoura, Portugal, November 25-30, 2007, LNCS 4803, 113–130. Springer, 2007.

[30] **K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson.** Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Technical Report* CMU/SEI-90-TR-21, *Software Engineering Institute, Carnegie Mellon University*, 1990.

[31] **P.-Y. Schobbens, P. Heymans, J.-Ch. Trigaux, Y. Bontemps.** Feature Diagrams: A Survey and a Formal Semantics. 14*th IEEE International Requirements Engineering Conference* (RE'06), *Minneapolis*, *Minnesota, USA*, September 2006, 139–148.

[32] **G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.-M. Loingtier, J. Irwin.** Aspect-Oriented Programming. *Proc. of* 11*th European Conference on Object-Oriented Programming* (*ECOOP* 1997), *Jyväskylä, Finland, June* 9-13, 1997. *LNCS* 1241, *Springer-Verlag*, 1997, 220-242.

[33] **W. Muller W. Rosenstiel, J. Ruf** (eds.). SystemC and Applications. *Kluwer Academic Publications*, 2003.

[34] **T. Sheard.** Accomplishments and Research Challenges in Meta-Programming. *In* 2*nd Int. Workshop on Semantics, Application, and Implementation of Program Generation* (*SAIG'*2001), *Florence, Italy. LNCS* 2196, *Springer*, 2001, 2-44.

[35] **R. Damaševičius, V. Štuikys.** Taxonomy of the fundamental concepts of metaprogramming. *Information Technology and Control, Vol*. 37(2), 2008, 124-132.

[36] **V. Štuikys, R. Damaševičius.** Development of Generative Learning Objects Using Feature Diagrams and Generative Techniques. *Informatics in Education, Vol*. 7(2), 2008, 277-288.

[37] **V. Štuikys, R. Damaševičius.** Design of Ontology-Based Generative Components Using Enriched Feature Diagrams and Meta-Programming. *Information Technology and Control,* 37(4), 2008, 301-310.

[38] **K. Czarnecki, C.H.P. Kim, K.T. Kalleberg.** Feature Models are Views on Ontologies. *Proc. of the* 10*th Int. Software Product Line Conference*, *Baltimore, USA, August* 21-24, 2006, 41-51.

[39] **W. Kintsch.** Comprehension: a paradigm for cognition. *Cambridge University Press*, 1998.

[40] **M. Frappier, S. Matwin, A. Mili.** Software Metrics for Predicting Maintainability: Software Metrics Study: Technical Memorandum 2. *Canadian Space Agency, January* 21, 1994.

[41] **J. Shao, Y. Wang.** A New Measure of Software Complexity based on Cognitive Weights. *Canadian Journal of Electrical and Computer Engineering,* 28(2), 2003, 69-74.

[42] **B. Briscoe, A. Odlyzko, B. Tilly.** Metcalfe's Law is Wrong. *IEEE Spectrum*, 26-3, *July* 2006.

[43] **J. Sametinger.** Software Engineering with Reusable Components. *Springer-Verlag, Berlin*, 1997.

Received March 2009.