International Journal of u- and e- Service, Science and Technology
Vol.8, No. 11 (2015), pp.241-252
http://dx.doi.org/10.14257/ijunesst.2015.8.11.24

Towards Expression and Evaluation of Design Deviation

Yucong Duan[1], Honghao Gao[2], Xiaobing Sun[3] and Nianjun Zhou[4]

[1]*College of Information Science and Technology, Hainan University, Haikou, China*
[2]*Computing Center, Shanghai University, Shanghai, China*
[3]*College of Information Engineering, Yangzhou University, Yangzhou, China*
[4]*IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA*
[1]*duanyucong@hotmail.com*, [2]*gaohonghao@shu.edu.cn*, [3]*sundomore@163.com*,
[4]*jzhou@us.ibm.com*

Abstract

Over Design (OD) and Under Design (UD) are two concepts which are used to express functional incompleteness, unexpected software quality degradation and other unexpected occurrence in design process. Measure of OD and UD is a big challenge which however lays the foundation of systematic usage of them. Based on Feature Modeling, we show in paper an approach to express UD and OD with logic connectives and several measure attempts.

Keywords*: Under design, Over design, measure*

1. Introduction

Since the concepts of Over Design (OD) and Under Design (UD) [1, 2] are well accepted in design evaluation, we choose them as the conceptual foundation for communication among stakeholders implementing either a quality driven [3] or quality aware [4] process, or concreting a Value Driven Design [5] ideology. To make full use of these two concepts, there are several challenges to be conquered including formalizing the semantics of them and design efficient and background of guiding unified value measure of a value driven process for both specific software products and software product families [6] projects.

The rest of the paper is organized as follows: Section 2 introduces empirical explanation of UD and OD under value criteria. Section 3 presents relevant hypotheses on defining measure. Section 4 introduces identifying UD and OD based on Feature Model description. Section 5 raises a logical semantic based approach on defining and measuring UD and OD. Section 6 presents related work. Section 7 concludes the paper with future directions.

2. Empirical Understanding of UD and OD

Based on extending value definition from value $\approx$ f unction/resources in Value Engineering [7] to

$$value = \frac{function \times quality}{resources}, \tag{1}$$

we define Under Design, Over Design and Ideal Design as follows:

Definition 2.1 (Over Design(OD)). The evaluated value of the design product or product family or intermediate design model in a certain design stage/state that is more robust or complicated than necessary for its whole investment by stakeholders. The produced extra quality or functionality will cost resources including: increased project time, increased effort, and deviation from optimized goals.

ISSN: 2005-4246 IJUNESST

Definition 2.2 (Under Design(UD)). The lowered value of the design product or product family or intermediate design model in a certain design stage/state that is less robust or complicated than expected by stakeholders. UD can be attributed to knowledge that is lost during the software development process. This may result in shortened effort or project time, but would adversely impact stakeholder value.

Definition 2.3 (Ideal Design (ID)). The value of the design product or product family or inter mediate design model in a certain design stage/state that matches stakeholders' expectations exactly. ID corresponds to the optimization of the profit and satisfaction of the targeted stakeholders at any stage during the whole design process.

3. Hypotheses for Validation and Variability Targets

3.1. Hypotheses for the Validation of Operations

To support the simplified value calculation including addition and subtraction among quality factors, functionalities and business concerns, every selected property (ESP) has to satisfy the following conditions:

- Every ESP needs to be reflected to a system value calculation related concern (SVCC) which does not subordinate to any other ESP whether partially or completely. That is to say,

$$\forall esp_i, esp_j \in ESP, svcc_z \in SVCC,$$
$$svcc_z(esp_i) \cap svcc_z(esp_j) = \emptyset, (i, j, z \in \mathbb{N}, i \neq j). \quad (2)$$

  This ensures that the addition and subtraction will not involve redundancy. For example, in common ticketing system the price property for vehicle covers the price property of either flight or car in a concrete transaction. Then the addition/subtraction operation is not applicable between vehicle and flight/car as between flight and car.

- The set of ESPs can reflect the main concerns of SVCC. That is, $\forall esp_i \in ESP, svcc_j \in SVCC,$

$$\sum svcc_j(esp_i, esp_2, \cdots, esp_j) \rightarrow SVCC, (i, j \in \mathbb{N}). \quad (3)$$

  This ensures that the division of ESP can be taken as the representation of the SVCC. For example, in ticketing system, the combination of the property of the price of flight and car will reflect the whole price for the vehicle.

- There are causal associations among different properties of ESP or they are independent from each other. That is, $\forall esp_i, esp_j \in ESP,$

$$esp_i \perp esp_j, (i, j \in \mathbb{N}). \quad (4)$$

  For example, in common ticketing system the quality of the payment verification is evaluated independently from that of the verification of the whole transaction. We make this assumption to ensure the validation of the value calculated for SVCC, and which can be used to identify characteristics of each property. However, in the next stage of the research, we will consider the causal relationship.

3.2. Hypotheses on Variability Target

In general, we can define various types of variability that are potentially related to OD and UD, such as the following:

- Personalization: Stakeholders have different standpoints on evaluating the influence of the structure of the model, the significance of a model in certain stages of the development process, the runtime performance of the system, the long term records of the business, the redistribution of the value added and the group level contribution of different services. Personalization will take in these influencing factors to improve the precision of shaping the individual interest. In this work, we only consider the personalization to the extent of different types of stakeholders in general.

- Model structure: The elements in the model in the structure will embody different variability types. The well known cases have been fully exposed in various design patterns [8].

- Runtime vs. design time variability from contract perspective: The interaction of services are through service contracts. At design time, the SLA and policies will serve as the source of the quality criteria for value derivation which include value scopes. However the value scopes sometimes cannot be set without an implementation routine. This kind of incompatibility will be exposed only through runtime monitoring. The variability management towards reducing this kind of incompatibility has to be planned with the consideration of runtime relationship among properties. We will explore this problem in future work.

- Long term vs. short term business pattern: To reach the best expectation of stakeholders, there are many business strategies including promotion, sellout, *etc*. The variability of the service system has to be planned to fit these strategies. This will require the variability of the system to be able to accommodate the required flexibility and proceed alongside the business strategy.

- Individual service vs. service set: Some variability of properties of individual services are related to the composition of the service system or stakeholder cooperation/competition. Then the variability can be set with the consideration of the composition implementation and the stakeholder organization.

4. Critiera of UD and OD in Feature Model and Proposition Logic

In this section, we focus on formally describing variabilities. Firstly, we translate the feature model into a logic expression within a design. Secondly, we measure the distance of different designs by calculating the steps of evolving one into another based on logic relationship. There are in fact two basic topics in solving UD and OD. They are:

- Identify UD and OD
- Measure the degree of UD and OD

The first topic includes the challenge on determining ID, which we leave to future work. To solve the second topic always means the process construction of evolving a design into another, thus, a transformation sequence is necessary. For example, transforming a design which requires both security and efficiency into a design which requires only high efficiency can be decomposed into two steps:

1. Give up the security requirements
2. Optimize the execution efficiency

On the other hand, value calculation is easier to operate on because judging value change according to a series of single change is always easier than judging value change as a whole. In this section we compare among different criteria and show the complexity in associating semantics of UD and OD.

4.1 Abstracting Design in Feature Model

Features[1] express variabilities and commonalities in a software product line. After abstracting variabilities into a feature model, the variabilities are limited within features in the feature model together with their valid combinations. Basically a feature model is an AND-OR graph and can be graphically viewed as a feature diagram. Figure 1 shows the semantics of feature constraints. According to the semantics, we transform a feature model into an equivalent propositional formula.

Table 1. Semantics of Feature Diagram Primitives

Feature Diagram Primitive	Semantics in logic expression
r is the root feature	r
f_1 is an optional sub-feature of f	$f_1 \rightarrow f$
f_1 is a mandatory sub-feature of f	$f_1 \leftrightarrow f$
$f_1, \cdots, f_n$ is a group of or sub-features of f	$f \leftrightarrow f_1 \vee \cdots \vee f_n$.
feature f_1 excludes f_2	$\neg(f_1 \wedge f_2)$
feature f_1 requires f_2	$f_1 \rightarrow f_2$

The set of features that satisfies the propositional formula of a feature model is called a valid feature configuration. A software product line with a feature model allows and only allows valid feature configurations. We use f to denote a feature model, $P(f)$ to denote the proposition associated with f, $L(f)$ to denote a product line with only permitted products. The process to determine the ID is the determination of an ideal feature model f_{ID}. We use D to show the design associated with feature model f_D. Then the process to minimize the OD or UD is related to the process from f_{UD} or f_{OD} to f_{ID} where f_{UD}, f_{OD}, f_{ID} separately represents a feature model assigned with UD, OD and ID.

To minimize UD/OD is formalized to minimize |ID − AD|, where AD is an actual design.This process is related to reaching the minimum of $|f_{AD} - f_{ID}|$. We assume that the difference between |AD − ID|has a positive correlation with $|f_{AD} - f_{ID}|$. The difference between feature model f_{AD} and f_{ID} is further related to the difference between $P(f_{AD})$ and $P(f_{AD})$. Thus we get the relation:

$$|\mathrm{AD} - \mathrm{ID}| \propto |f_{\mathrm{AD}} - f_{\mathrm{ID}}| \propto |P(f_{\mathrm{AD}}) - P(f_{\mathrm{ID}})|. \quad (5)$$

But the calculation of $P(f_{AD}) - P(f_{ID})$ meets the problem that the cardinal number of the set $P(f)$ is blooming with the increment of variants. To calculate the distance between two proposition formulas, we need to consider the space of all propositions sharing the same variables.Noting that feature model f with n features is mapped to a proposition formula $P(f)$ with n talked in subsequent subsections.

The calculation of the difference between two propositions can be modelled as the construction of path in the space where propositions share the same variables together with the inference relationship among them. So the calculation can be benefit from existing path planning algorithms [9].

[1] http://en.wikipedia.org/wiki/Feature_model

OD and UD has some important characteristics. When we are using the concept of UD and OD, we are in fact comparing the relationship of two design. This comparison derives an important binary relationship between two designs.

- If changing design D_{prev} to design D_{next} is an OD, then the reversed direction, changing design *Dnext* to *Dprev* , is obviously an UD. Modeling OD and UD as two binary relations over the set of designs $D = \{D_1, \cdot \cdot \cdot, D_n\}$, we get the conclude the first property on UD and OD:D_1 is an OD of D_2 , if and only if D_2 is an UD of D_1.

- Design D_1 is neither an OD of itself nor an UD of itself. It's only identical to itself. Thus, we get the conclusion: D_1 is ID to itself.

- Suppose that D_1 is an OD of D_2 and D_2 is an OD of D_3. Naturally we consider that design requirements of D_1 is also stricter than that of D_3. And the suppose indicates that D_1 is an OD of D_3, and is "more OD thanD_1 to D_2 ". Thus we get the conclusion: If D_1 is an OD of D_2 ,and D_2 is an OD of D_3, then D_1 is an OD of D_3 (with heavier degree).

Denote that $\alpha, \beta, \gamma \in D$ are three designs, and < means binary relation" is UD of", then above properties can be formalized as:

1. For all α,β, $\alpha < \beta$, $\alpha > \beta$, and $\alpha = \beta$ are alternative.
2. For all α, $\alpha = \alpha$.
3. For all α, β, γ with $\alpha < \beta$ and $\beta < \gamma$, then $\alpha < \gamma$.

The relationship < is transitive. The three relationship items indicate that the set of designs *D* together with the relationship "not UD" or "not OD" is a partial ordered set in mathematics. But the binary relation is difficult to further evolved into a total ordered set. If we remove a feature f_1 from design D_1 in the first step and add a feature f_2 (which is different from f_1) to the result of previous step then we get design D_2, without more information we could not determine whether the design is OD thanD_1 or UD than D_1.

4.2. UD/OD Criteria based on Proposition Inference

The Proposition Logic Theory[2] asserts that a proposition *P* (*f*) with n variables is decided at most 2^n variable assignments (each variable assignment gives the judgment whether a variable is set to `True` (`T` for short) or `False` (`F` for short). For example, if $P(f_{D_a}) = S \wedge (S \rightarrow (A \vee B))$, hen the assignment set where $P(f_{D_a}) =$ `T` is:

$$H(P(f_{D_a})) = \{(S, A, \neg B), (S, \neg A, B), (S, A, B)\}.$$

where $(S, A, \neg B) \in H(P(f_{D_a}))$ means that when setting $S =$ `T`,$A =$ `T` and $B =$ `F`, the proposition $P(f_{D_a}) =$ `T`.

If a series of feature models f_{D_i} is given, the relation among these $H(P(f_{D_i}))$ is further used to calculate the difference for feature families $\{f_{D_i}\}$.If two feature models have different features, we assume that extra variables is free to existing propositions. For example, proposition $P(f_{D_b}) = S \wedge (S \rightarrow A)$, indicates that $H(P(f_{D_b})) = \{(S, A)\}$. When comparing feature model f_{D_b} with f_{D_a} , we extends $H(P(f_{D_b}))$ into

$$H*(P(f_{D_b})) = \{(S, A, \neg B), (S, A, B)\}.$$

Both assignments in $H*(P(f_{D_b}))$ makes $P(f_{D_b}) =$`T`. So we used $H(P(f_{D_b}))$ as a synonym of $H*(P(f_{D_b}))$ in this section.

Basically we use the criterion of allowed products to decide whether UD or OD happens. To apply this comparison criterion, referred propositions are lifted with sharing same

[2] http://en.wikipedia.org/wiki/Propositional_calculus

variables as the method above. Then the basic relationships between two propositions $H(P(f_{D1}))$ and $H(P(f_{D2}))$ are identified into two types:

1. $H(P(fD_1)) \subseteq H(P(fD_2))$
2. $H(P(fD_2)) \supseteq H(P(fD_1))$

Since $H(P(fD_1)) \subseteq H(P(fD_2))$ indicates that the products of fD_1 is the subset of fD_2, the change from D_1 to D_2 is an UD. We use

$$D_1 \xrightarrow{MT|_{UD}} D_2$$

to represent the design change, then for the two cases above, we separately get:

1. $D_1 \xrightarrow{MT|_{UD}} D_2$
2. $D_2 \xrightarrow{MT|_{OD}} D_1$

In our above examples, $H(P(f_{D_a})) \supseteq H(P(f_{D_b}))$, so $D_a \xrightarrow{MT|_{OD}} D_b$.

This criterion judges UD and OD mainly based on the flexibility of software product line. The treatment of variable extension in the approach takes extra features as an optional part. Comparing with taking extra features as a mandatory part or an exclusive part, the strategy is more suitable for demanding new features, but still cannot reflect the extract semantics inpractice.

The more general case is that both

$$H(P(fD_1)) - H(P(fD_2)) \neq \Phi$$

and

$$H(P(fD_2)) - H(P(fD_1)) \neq \Phi$$

is true. In above expressions, $H(A) - H(B)$ means the variable assignment make A = `T` but B = `F`. In this situation, we compare D_1 and D_2 based on the intersection and union of $H(P(fD_1))$ and $H(P(fD_2))$. When transforming D_1 to D_2, the product line mapped with $H(P(f_1)) - H(P(f_2))$ is abandoned, at the same time the product line mapped in $H(P(f_2)) - H(P(f_1))$ is added. The missing of former part caused an OD, and the added part caused an UD.

However, the relationship between two feature model, not only relies the case where $P(f)$ = `T` , but also the case where $P(f)$ = `F` . Recommended that a feature model not only allows several products, but also forbids several products, especially the use of exclusion-features.Thus two sets are considered: one is $H_1(P(fD_1))$ and the other is $H_2(P(fD_1))$. The former makes $P(fD_1)$ = `T` and the later makes $P(fD_1)$ = `F`.

4.3. UD/OD Criteria based on Feature Model Structures

According to the previous work in [10], relationships among product sets are divided into four cases:

- Refactoring: No new products is added and no existing products is removed
- Generalization: new products are added and no existing products is removed
- Specialization: no new products is added and some existing products are removed
- Arbitrary: new products are added and at the same time some existing products are removed

In this case, we should firstly consider the feature set of a feature model or the variable set of a proposition. The UD and OD can be mapped with the following criteria:

- If D_1 is a refactor of D_2, then D1 is equivalent to D_2.
- If D_1 is a generalization of D_2, then D1 is an OD.
- If D_1 is a specialization of D_2, then D1 is an UD.

- For other situations, we decompose the change into several steps with each step satisfying the above 3 situations.

Suppose that feature model f_{Db} satisfied with $S \wedge (S \rightarrow A)$ has base set $\{S, A\}$, while feature model f_{Da} satisfied with $S \wedge (S \rightarrow (A \vee B)$ has the base set $\{S, A, B\}$. According to the criteria, D_a generalizes Db, then D_a is an OD to Db. The result is opposite with the result under criterion in above subsection. But both of them reflects one aspect semantics of UD and OD.

4.4. OD and OD from Basic Feature Expression

Basic level feature model uses some very special conditions and the method can be easily extended for general cases. Assuming that the relationship between M and its sub-modules belongs to one of the four cases: Mandatory (Man), Optional (Opt), Or and Alternative (Alt) If only one of these four conditions is applied to module M and its sub-modules $\{M_i\}$ and the restriction is Man, then $M_1 \wedge M_2 \wedge \cdots \wedge M_n$ is the semantic. The relationships among the four restrictions are shown in Table 2.

Table 2. Relationship of restrictions

	Mandatory	Optional	Or	Alternative
Mandatory	$=$	$\subseteq$	$\subseteq$	$\neq$
Optional	$\supseteq$	$=$	$\supseteq$	$\supseteq$
Or	$\supseteq$	$\subseteq$	$=$	$\supseteq$
Alternative	$\neq$	$\subseteq$	$\subseteq$	$=$

We use Feature Model [11] to model variability in software design. During many variants of feature models, we choose basic feature model[3] as our variability framework. The relationship between a root feature and its child features are divided into six basic classes in Table 1. With this translation, constraints are uniformly expressed as a logic proposition. Next we will use inference relationship between two propositions to get the comparison between two designs. In table 3, each column represents an ID and each row represents an actual design. Then whether UD or OD happens is shown in Table 3.

Table 3. Restrictions with respect to UD and OD

ID \ AD	Mandatory	Optional	Or	Alternative
Mandatory	$=$	UD	UD	$\neq$
Optional	OD	$=$	OD	OD
Or	OD	UD	$=$	OD
Alternative	$\neq$	UD	UD	$=$

For example, at design time, the restriction Or $(P \rightarrow (A \vee B))$ is the ID, but after model transformation, $\xrightarrow{MT}$ restriction changes to be logic and $S \rightarrow (A \wedge B)$, that is:

$$A \vee B \xrightarrow{MT} A \wedge B.$$

[3] http://en.wikipedia.org/wiki/Feature_model

In this case, the design is identified as OD. The only problem is the transformation $\xrightarrow{MT|}$, so denoting the design as OD is the same as stating that the above transformation has been implemented. A feature A changing to feature $A \vee B$ is another change style. We previously consider changes among features that consist of many child features, $A \vee B$ to $A \wedge B$ for example. If we add a new feature B to original feature A, the relationship between two features will be different. From the perspective of proposition logic, $A \vee B$ is weaker than A. Supposed that A is the ID, then we get:

$$A \xrightarrow{MT|_{UD}} A \vee B.$$

This approach relies on decomposing the process of the transformation into several basic transformations in one of the six basic form. With the help Table 3, every basic step of transformation is labeled with either ID, OD or UD. Then we are able to get a chain to meet measuring UD/OD like:

$$D_1 \xrightarrow{MT|_{OD}} D_2 \xrightarrow{MT|_{UD}} D_3 \xrightarrow{MT|_{UD}} D_4.$$

In each step a transformation is assigned with a value change, so this approach can further extends to a method to calculate the value change of OD and OD.

5. A Measure of UD and OD based on Logic Assignment

Suppose that there are n variables written as $v_1, v_2, \cdots, v_n$. After assigning all each of them with True of False and then connecting them with "and" relation, we get 2n basic propositions, $v_1 \wedge \cdots \wedge v_n$, $v_1 \wedge \cdots \wedge \neg v_n$ for example. According to logic theory, any proposition on these n variables can be written in a conjunctive normal form (CNF). So these 2n basic propositions propositions are written as

$$A \wedge B, A \wedge \neg B, \neg A \wedge B, \neg A \wedge \neg B.$$

A normal proposition on V is in fact equivalent to the or-relationship among basic propositions.

For example, $A \vee B$ is equivalent to

$$(A \wedge B) \vee (A \wedge \neg B) \vee (\neg A \wedge B).$$

Another example is that proposition $\neg A \vee B$ is equivalent to

$$(\neg A \wedge B) \vee (\neg A \wedge \neg B) \vee (A \wedge B).$$

Next the set of basic propositions BP = {$BP_1 \cdots BP_m$} together with the relationship "the number of variables" is an ordered set. The definition is that: denoting that r is the number of positive variables of a basic proposition, then define $BP_1 < BP_2$ if and only if $r(BP_1) < r(BP_2)$. we call the function r the rank of basic proposition. For example, $r(A \wedge B) = 2$, $r(\neg A \wedge \neg B) = 0$.

The distance between two basic propositions is defined as the derivation of their ranks, that is $\delta(BP_1, BP_2) = r(BP_1) - r(BP_2)$. Accepting this definition and assuming that adding features is always an OD, then we conclude that changing BP_1 to BP_2 is an OD $\Leftrightarrow r(BP_2) > r(BP_1) \Leftrightarrow \delta(BP_1, BP_2) = -\delta(BP_2, BP_1) < 0$. Further more, the degree of $\delta(BP_1, BP_2$ is regarded as the measure of UD and OD in among basic propositions.

We define the measure about UD and OD with general features and associated with general propositions as

$$\delta(P_1, P_2) = \frac{1}{|P_1| \cdot |P_2|} \sum_{i \in P_1} \sum_{j \in P_2} \delta(i, j), \tag{6}$$

where $i \in P_1$ means that i is a CNF item of P_1. $\delta > 0$ means design $D(P_1)$ is an OD of $D(P_2)$, otherwise $D(P_1)$ is an UD of $D(P_2)$ or equivalent to $D(P_2)$. For example,

$$\delta(A \vee B, A \wedge B) = -2/3,$$

then $A \vee B$ is an UD of $A \wedge B$. If $D(P_3)$ allows both feature A and B be optional, then $\delta(P_3, A \wedge B) = -1$. In this case we find that $D(A \wedge B)$ to P_3 is also an UD, and since that $|\delta(P_3, A \wedge B)| > |\delta(A \vee B, A \wedge B)|$, we identified that converting $D(A \wedge B)$ to $D(P_3)$ causes more serious UD than converting to $D(A \vee B)$. In this process we have the expression:

$$D(A \wedge B) \xrightarrow{\mathrm{MT}_{\mathrm{UD}=1}} D(A \vee B)$$
$$D(A \wedge B) \xrightarrow{\mathrm{MT}_{\mathrm{UD}=2/3}} D(P_3)$$

6. Related Work

From the perspective of process quality assurance, we have identified that constraint based modeling [12, 13] manages to accommodate abstract or temporarily not distinguishable information of design decisions. It can postpone the decision to a later stage while mitigating loss of quality. This will contribute to making choices that are relatively invulnerable to either OD or UD. Thereafter we propose that a corresponding solution lies in employing constraints as a major form of design activity which is called constraint driven design [14].

7. Conclusions and Future Work

As part of our goal to implement Value Driven design in Software Economics, we have planned our framework of unifying both functional and quality factors in design process uniformly through the concepts of UD and OD. The values of the UD and OD are used to bridge the business value calculation and the integration of the measurement of the quality factors and value properties. To serving the goal of automation of the identification and reasoning on UD and OD, we present in this work a systemic approach to define, identification and measure UD and OD in the background of Feature based modeling and first order logic. Three different measurement of UD and OD are presented of which two of them are independent measurements and one demands the presence of an ideal design.

There are still huge challenges left ahead in both refining this approach with more complex situations including the identification of the incomplete model vs. UD model, and quantifying the various quality factors in a consensus manner. Apart from traditional software development process where OD and UD are intertwined with many complex design errors, failure, and deficiencies, we will be more interested in adapting and applying our approach to relatively simple Mashup process of Web services.

Acknowledgment

The authors acknowledge the support of the Natural Science Foundation of of China (No. 61363007 and 61440019), Natural Science Foundation of Hainan (No. 20156234), Hainan University Research program (HDSF201310), Natural Science Foundation of Shanghai (No. 15ZR1415200), and Young University Teachers Training Plan of Shanghai Municipality (No. ZZSD13008).

References

[1] Shalloway, S. Bain, K. Pugh and A. Kolsky, "Avoid over- and under-design In Essential Skills for the Agile Developer", A Guide to Better Programming and Design, **(2011)**, pp. 248–263.
[2] J. Kerievsky, "Stop over engineering", Software Development, **(2002)**.
[3] T. A. Naeem, I. Gorton, M. A. Babar, F. Rabhi and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications", Proceedings of the 27th international conference on Software engineering, ACM, **(2005)**.
[4] M. L. Drago, C. Ghezzi and R. Mirandola, "A quality driven extension to the qvt-relations transformation language", Computer Science-Research and Development, **(2011)**, pp. 1–20.
[5] AIAA, Value-driven design, American Institute of Aeronautics and Astronautics, **(2008)**, pp. 109–109.
[6] K. W. Lee, K. C. Kang and J. J. Lee, "Concepts and guidelines of feature modeling for product line software engineering", Software Reuse: Methods, Techniques, and Tools, Springer, **(2002)**, pp. 62–77.
[7] "SAVE International", Value Standard and Body of Knowledge, **(2007)**.
[8] E. Gamma, "Design patterns - past, present & future", In Sebastian Nanz, editor, The Future of Software Engineering, Springer, **(2010)**, p. 72.
[9] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning", Robotics and Automation, IEEE Transactions, vol. 8, no. 1, **(1992)**, pp. 23-32.
[10] T. Thum, D. Batory and C. Kastner, "Reasoning about edits to feature models", Software Engineering, ICSE, IEEE 31st International Conference, IEEE, **(2009)**.
[11] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study", Technical report, DTIC Document, **(1990)**.
[12] A. Demuth, R. E. L. Herrejon and A. Egyed, "Constraint-driven modeling through transformation", Theory and Practice of Model Transformations, Springer, **(2012)**, pp. 248–263.
[13] A. Egyed and D. S. Wile, "Support for managing design-time decisions", Software Engineering, IEEE Transactions, vol. 32, no. 5, **(2006)**, pp. 299–314.
[14] K. Lano, "Constraint-driven development", Information and Software Technology, vol. 50. no. 5, **(2008)**, pp. 406–423.

Authors

Yucong Duan, he received a PhD in Software Engineering from Institute of Software, Chinese Academy of Sciences, P. R. China in 2006. He is currently a full professor and vice director of Computer Science department in Hainan University, P. R. China. His research interests include software engineering, service computing, cloud computing, and big data. He is a member of IEEE, ACM and CCF(China Computer Federation).

Honghao Gao, he received the Ph.D degree in computer application technology from the School of Computer Engineering and Science of Shanghai University, Shanghai, P. R. China, in 2012. His research interests include Web service and model checking.

Xiaobing Sun, he received his Ph.D from Southeast University in 2012. He is currently an assistant professor in School of Information Engineering at Yangzhou University. His research interests include software analysis, maintenance and evolution. He is a CCF and ACM member.

Nianjun Zhou, he is a research staff member (RSM) of IBMT. J. Watson Research Center. His interesting is using computer methodologies and technologies to innovate new ideas, develop new infrastructure and applications which enhance the computing resources utilities, efficiency of knowledge and information management.