International Journal of Multimedia and Ubiquitous Engineering
Vol.10, No.12 (2015), pp.279-288
http://dx.doi.org/10.14257/ijmue.2015.10.12.28

# Formalizing Over Design and Under Design

Yucong Duan[1], Honghao Gao[2], Jingbing Li[3] and Mengxing Huang[4]

*[1,3,4] College of Information Science and Technology, Hainan University, Haikou, China [2]Computing Center, Shanghai University, Shanghai, China*
*[1,3]{duanyucong,jingbingli2008}@hotmail.com,[2] gaohonghao@shu.edu.cn,*
*[4]huangmx09@gmail.com*

***Abstract***

*Over Design (OD) and Under Design (UD) are two common concepts which are used to express negative construction of functionality, quality and even value. Software requirement satisfaction can be decomposed a routine of constant steps of minimizing UD and O D. How-ever currently they don't have formal semantics from existing literature. The situation results in inefficient identification, description and subsequent modeling activities centering them. In his paper, we work towards formalized abstract semantics of OD and UD from the perspective of knowledge introduction in a development process. We also show the deduction steps of describing and eliminating UD and OD.*

***Keywords:*** *Over Design; Under Design; Conceptualization; Knowledge management*

## 1. Introduction

While the concepts of Over Design (OD) and Under Design (UD) [1-2] have been extensively used in informal situations of discussion and evaluation of a design, there is no existing formal explanations of the meaning of them which can shape a unified understanding of them. From literature survey, we have observed an increasing usage of them recently for development management especially in agile methodology. A formalization of these concepts will improve the efficiency and precision of the using them for communication. It also will potentially open a modeling direction for solving the long existing challenge of sharing communication between stakeholders on the two groups of technical domain and management/business domain [3] in the process of implementing quality driven [4] or quality aware [5], and Value Driven Design [6] since all three of them suffer in essence an absent of efficient shared concepts which support the fluent flow of information between two sides. Motivated by finding core concepts for communicating among stakeholders from both technical domain and management/business domain concreting a software economics project [7], we discuss in this paper the formalization of the UD and OD from the knowledge introduction perspective in a software design process. The knowledge introduction process is cognitively general enough which guarantees the not lowering the generality of the explained concepts based on them. We also show the deduction steps of describing and eliminating UD and OD abstracted from a running example.

The following part of the paper is arranged as follows: Section 2 introduces empirical understanding of UD and OD, and formalization motivation for implementation of Value Engineering. Section 3 shows the running example of UD and OD scenarios. Section 4 abstracts the expression formulas of UD and OD from a knowledge introduction perspective. Section 6 discusses related work. Section 7 concludes with future directions.

ISSN: 1975-0080 IJMUE

## 2. Empirical Understanding and Motivation

We extend the value formula in Value Engineering [8] to

$$value \approx \frac{valueof function \times valueof quality}{valueof resources} \quad (1)$$

Based on this value definition, we define Under Design (UD) and Over Design (OD) as follows

Definition 2.1 (Over Design (OD)) the evaluated value of the design product or product family or intermediate design models 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.

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.

Ideal Design (ID) happens when the value of the design product or product family or intermediate design model in a certain design stage/state matches stakeholders' expectations exactly.ID corresponds to the optimization of the profit and satisfaction of the targeted stake holders at any stage during the whole design process.

The motivation for this work is that there is a gap in the implementation process of Value Driven Design methodology. The gap lies between the implementation from business planning layer to the technical design layer. There is a need to direct the value measure in the business strategy to the change of the modeling and implementation of the technical system. There are many ways to model and implement a system. To show the detail of our solution, we restrict that the system or system family [9] is built with a model driven development process. In this process, there are many technical factors which can be managed to influence the business value of a system for stakeholders. However, we identified that the concepts which are used in the business planning layer are mostly business terms which are different from the concepts used in the technical design which are mostly about the system modeling and implementation detail. There is the need to bridge the communication between the business management with the technical modeling/implementation.

To bridge the concepts at the two layers while confirming to value driven ideology, typical solutions include:

1. Mapping the concepts directly between the two layers;
2. Introducing intermediate concepts.

However, since the terms in either the business layer or the technical layer have not been standardized enough to unify the understanding, the direct mapping will need to be adapted for many applications to address misunderstandings. Thereafter, we propose to introduce intermediate concepts as a solution.

 

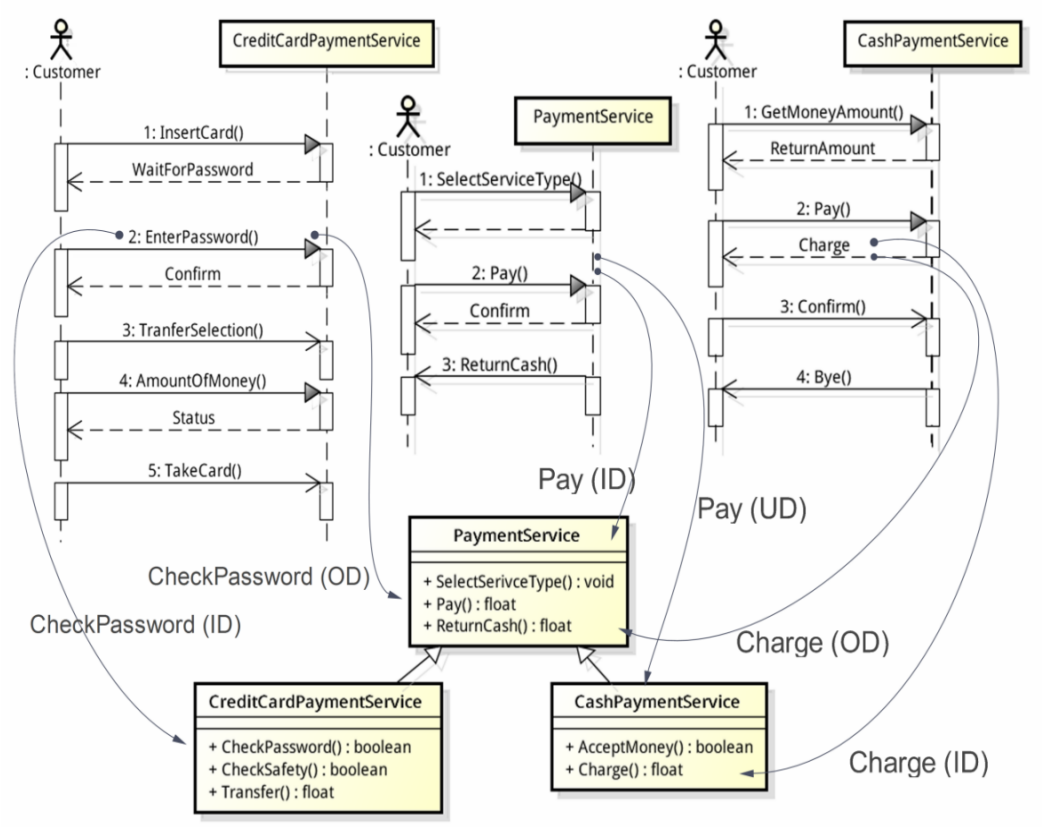


**Figure 1. UD and OD in Payment Service**

## 3. Running Example

Figure 1 illustrates UD and OD in modeling a payment service. In order to provide better payment services, a customer can choose one of the two options: pay using credit card or using cash. By using credit card payment service, entering password to a POS terminal is necessary. By using cash payment service, charging for small change is necessary. Both services have an abstract operator called "pay". When modeling this system, implementing Pay() in class CreditCardPaymentService or CashPaymentService caused an UD, and implementing CheckPassword() in class PaymentService caused an OD.

## 4. Revelation from Knowledge Introduction Perspective

### 4.1. Basic Modes

From the knowledge management perspective, we abstract the information flow of a software development process in general as the repeated mode of firstly introducing knowledge from stakeholders/context and then transform it to existing model/product:

$$introduction(knowledge) \mapsto transformation(knowledge) \tag{2}$$

The knowledge transformation from requirement specification to design model can be modeled as repeated mode of from known knowledge on requirement specification (RE) to known knowledge in models (MD):

$$known|_{RE} \mapsto known|_{MD} \tag{3}$$

However if this mode is actually performed by human, they will make the situation much more complex according similar to categories False Positive (FP), False negative (FN), True Positive (TP), and True negative(TN) in statistical hypothesis testing. We have also found from our survey that most existing work focus on the knowledge transformation in the form of

$$known|_{RE} \mapsto known|_{MD} \tag{4}$$

Which omitted the significance of keeping the variability or empty space which need further knowledge introduction at a proper time during the design process before it can be specified in the models. In contrast to

$$known|_{RE} \mapsto known|_{MD} \tag{5}$$

We define this request as

$$unknown|_{RE} \mapsto unknown|_{MD}. \tag{6}$$

### 4.2. Map to Positive/Negative and True/False

We relate the categories in knowledge transformation and statistical hypothesis testing starting from the mapping from known/unknown knowledge to Positive/Negative. Known is mapped to Positive since known is expected in software system. The more known information in the models the nearer the accomplish of the system constructions in general.

$$known|_{RE} \mapsto Positive, unknown|_{RE} \mapsto Negative \tag{7}$$

The ideal knowledge transfer from requirement specification will confirm to the restriction that known information will be transferred to the intermediate models incrementally until the final system is accomplished. The process is expected to be correct which means that the known knowledge will be transferred to determined design decisions such as the “Tiger A shares all properties in Animal” in requirement specification is embodied in “Class (Tiger) inherits Class (Animal)” in Class Model. And the unknown knowledge will be conserved as unknown/blank information as well during the design process before enough knowledge is provided to fill the blank. The mapping from known/unknown to True and False is as follows The True represents the situations which confirm to the ID as follows:

$$(known|_{RE} \to known|_{MD}) \xrightarrow{ID} True \tag{8}$$

$$(unknown|_{RE} \to unknown|_{MD}) \xrightarrow{ID} True \tag{9}$$

The violation of the ideal knowledge transfer including:

(a) The loss of the known knowledge during the transfer and is replaced with unknown. This indicts UD since it means the loss of the expected increase of the business value while investment increase;

(b) The increase of the “known” knowledge not relying on known knowledge transferring but by mistake or subjectively added. The expected situation is that the unknown situation will be kept in the models at that design stage. This unexpected replacement indicts OD since that the replacement of unknown information with probabilistic not optimized option, which is determined by the not guaranteed ”known” information, will means the missing of the chance to introduce the optimized which is determined by the proper known knowledge at a later stage. The gained value represented by the taken ”known” information will be probabilistically less than the gained value in

 

the optimized solution which is determined by keeping the unknown information in the model or postponing the decision until sufficient information is provided. This kind of evaluated loss belong to OD.

The False represents the situations of UD and OD as follows:

$$(known|_{RE} \rightarrow unknown|_{MD}) \xrightarrow{UD} False \tag{10}$$

$$(unknown|_{RE} \rightarrow known|_{MD}) \xrightarrow{OD} False \tag{11}$$

### 4.3. Map to FP, FN, TP and TN

Based on the mapping to Positive/Negative and True/False, we can construct the mapping to FP, FN, TP and TN as follows.

$$(known|_{RE} \rightarrow known|_{MD}) \mapsto TP \tag{12}$$

$$(unknown|_{RE} \rightarrow unknown|_{MD}) \mapsto TN \tag{13}$$

$$(known|_{RE} \rightarrow unknown|_{MD}) \mapsto FP \tag{14}$$

$$(unknown|_{RE} \rightarrow known|_{MD}) \mapsto FN \tag{15}$$

### 4.4. The Mapping from FP, FN, TP, TN to ID, UD and OD

The mapping from FP, FN, TP, TN to ID, UD and OD has the following basic patterns:

$$((known|_{RE} \mapsto known|_{MD}) \mapsto ID) \Rightarrow (TP \mapsto ID) \tag{16}$$

$$((unknown|_{RE} \mapsto unknown|_{MD}) \mapsto ID) \Rightarrow (TN \mapsto ID) \tag{17}$$

$$((known|_{RE} \mapsto unknown|_{MD}) \mapsto UD) \Rightarrow (FP \mapsto UD) \tag{18}$$

$$((unknown|_{RE} \mapsto known|_{MD}) \mapsto OD) \Rightarrow (FN \mapsto OD) \tag{19}$$

We derive the following patterns mapping statistical hypothesis to UD/OD transformation:

$$MT(TP \rightarrow TN) \Leftrightarrow MT(ID \rightarrow ID) \tag{20}$$

$$MT(TP \rightarrow FN) \Leftrightarrow MT(ID \rightarrow OD) \tag{21}$$

$$MT(FP \rightarrow FN) \Leftrightarrow MT(UD \mapsto OD) \tag{22}$$

$$MT(TN \rightarrow FN) \Leftrightarrow MT(ID \mapsto OD) \tag{23}$$

## 5. Case Study

We use a Tourism Service System as an illustration. In Tourism Service System, different services including TravelTool service, Car service, Hotel service, Payment service, *etc.*, should be composed with each other. The general scenario of Tourism Service System is shown in Figure 2. In Figure 2, → stands for workflow direction, + is the parallel operator, ∗ is the chosen operator, and R denotes a repeated operation.

There are three main modules in Figure 2, namely TravelAgent, AccommodationAgent and payment. In the first part of this section, we present a general workflow WF1 and two sub- workflows. The complete workflow is written in the form (WF4):

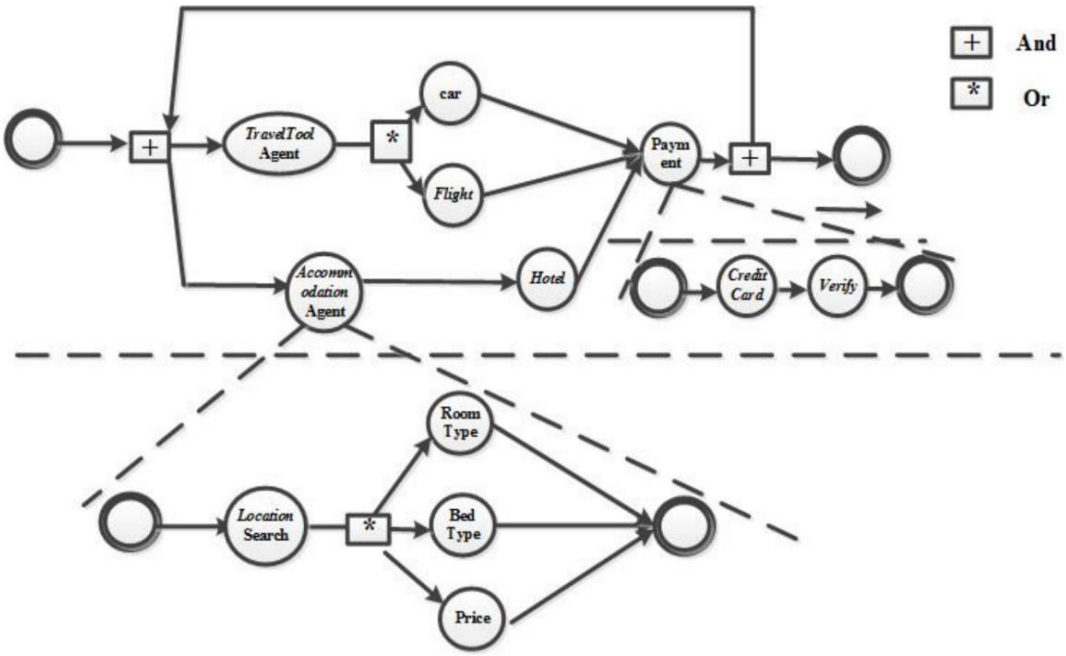


**Figure 2. The General Scenario of Tourism Service System**

$$
\begin{aligned}
S \rightarrow & (((TravelTool \rightarrow (Car * Flight)) \\
& +((Accommodation \rightarrow LocationSearch \rightarrow (RoomType * BedType * Price)) \rightarrow Hotel)) \\
& \rightarrow (Payment \rightarrow CreditCard \rightarrow Verify))^R \\
\rightarrow & E
\end{aligned}
\tag{24}
$$

Next step is to convert this scenario to features in design. Part of this work is shown in our running examples on model-driven design of TravelAgent and its subclasses, Car and Flight. Suppose that every service in Figure 2 has a function module with the same name. Then the relationships of services shown in Figure 2 are formulated as:

$$
\begin{aligned}
& ((TravelAgent \rightarrow (Car \vee Flight)) \wedge \\
& ((AccommodationAgent \rightarrow LocationSearch \rightarrow (RoomType \vee BedType \vee Price)) \rightarrow Hotel)) \\
& \rightarrow Payment \rightarrow CreditCard \rightarrow Verify
\end{aligned}
\tag{25}
$$

In the formula above, A → B means that A is implemented by B. That means both A and its subclass B are required to be implemented. so A → B is in fact A $\wedge$ B, *i.e.*, both A and B are present in the design.

Given another design D2, whose features follow the following relationships:

$$
\begin{aligned}
& ((TravelAgent \rightarrow (Car \vee Flight \vee Train)\ ) \wedge Price \rightarrow Hotel) \\
& \rightarrow Payment \rightarrow CreditCard \rightarrow Verify
\end{aligned}
\tag{26}
$$

D2 means that designer of Tourism Service System wants to enhance its business in providing as much transportation as possible. We want to compare the difference between

 

original design (D1) and D2. A natural way is to calculate how D2 changes to D1. Thus, we get the following steps:

Step 1 (*S1*):

$$((TravelAgent \rightarrow (Car \vee Flight)) \wedge Price \rightarrow Hotel) \\ \rightarrow Payment \rightarrow CreditCard \rightarrow Verify \tag{27}$$

This step cuts off the design of TrainService. Since a service is removed from Tourism Service System, step 1 caused a UD of design D2. After this step, the Train service is removed. This caused value decreasement for the adaption of Tourism Service System. But this simplification make design and coding in software development much easier. For the former reason, value of D2 decreases by 20, and for the later reason, value of D2 increases by 12. Then the total value of design D2 increases by −20 + 12 = −8.

Step 2 (*S2*):

$$((TravelAgent \rightarrow (Car \vee Flight)) \wedge (BedType \vee Price) \rightarrow Hotel) \\ \rightarrow Payment \rightarrow CreditCard \rightarrow Verify \tag{28}$$

This step increases the design of BedType. Since a service is added to Tourism Service System, Step 2 caused an OD of design in Step 1. Appending extra features is OD to original design. The value that brings to the design depends on how the quality properties changes according to the new feature. For example, the design of BedType increases cost of development by value8, but increases running payback by value 80, then the value of design in step 2 increases by80 − 8 = 72 compared to value of design in step1.

Step 3(*S3*):

$$((TravelAgent \rightarrow (Car \vee Flight)\ ) \wedge (RoomType \vee BedType \vee Price) \rightarrow Hotel) \\ \rightarrow Payment \rightarrow CreditCard \rightarrow Verify \tag{29}$$

This step increases the design of RoomType in the Tourism Service System, which is an OD of design in Step 2. Principles of value variation is the same as in step 2. We directly give the value increases by 80 in this article.

Step 4(*S4*):

$$((TravelAgent \rightarrow (Car \vee Flight)\ ) \wedge \\ ((LocationSearch \rightarrow (RoomType \vee BedType \vee Price)) \rightarrow Hotel)) \\ \rightarrow Payment \rightarrow CreditCard \rightarrow Verify \tag{30}$$

This step increases the design of superclass LocationSearch. This increases the design from

$$RoomType \vee BedType \vee Price$$

to

$$LocationSearch \rightarrow (RoomType \vee BedType \vee Price).$$

And this is a OD of design in Step 3. Value variation in this step is a little complicated. Considering that the design of superclass enhances code reuse, the value increases by 40. At the same time, it simplifies maintenance, thus the value increases by 60. Finally, integrating all the factors, change in step 4 increase the value of design by 100.

Step 5 (*D2*):

$$((TravelAgent \rightarrow (Car \vee Flight)) \wedge \\ ((AccommodationAgent \rightarrow LocationSearch \rightarrow (RoomType \vee BedType \vee Price)) \rightarrow Hotel)) \\ \rightarrow Payment \rightarrow CreditCard \rightarrow Verify \tag{31}$$

This step increases the design of superclass AccommodationAgent. This increases the design in step 4, so it is an OD of design in step 4. This change is useful for business extension,but at now it is debated among designers. Some think that increases the value, but others do not think so. Supporters give value increment by 120, but others give value decrement by 80.Finally, the value of design in step 5 increases the total value by $120 - 80 = 40$.

Note that this design is the same as design D1. If ideal design is set to D1, then reverse the steps above, we get design D2. In each step, OD or UD is reversed. Thus, we get:

$$D_2 \xrightarrow{UD-8} S_1 \xrightarrow{OD+72} S_2 \xrightarrow{OD+80} S_3 \xrightarrow{OD+100} S_4 \xrightarrow{OD+40} D_1 \tag{32}$$

and

$$D_1 \xrightarrow{UD-40} S_4 \xrightarrow{UD-100} S_3 \xrightarrow{UD-80} S_2 \xrightarrow{UD-72} S_1 \xrightarrow{OD+8} D_2 \tag{33}$$

That is,

$$D_2 \xrightarrow{OD+284} D_1, \quad D_1 \xrightarrow{UD-284} D_2 \tag{34}$$

## 6. Related Work

Barry Boehm created Constructive Cost Model (COCOMO) [10] to relate the software development effort in terms of Person-Months (PM), to program scale in terms of Thousand Source Lines of Code (KSLOC). However lines of code does not apply to model driven approach where efforts differs much more on the architecture content including design patterns, style, framework, *etc*., instead of amount of modeling element.

Carvajal *et al.* [11] proposed the problem that when the design artifacts deviate from the initial requirements, it is hard to detect these differences. Kerievsky [2] proposed to avoid the OD and UD in the form of Over engineering (or over-engineering) which refers to the product designing more robust or complicated than is necessary for directly planned applications, through understanding the evolutionary of design patterns. Designing software requires an understanding of domain specific design problems. Tang *et al.* [12] pointed out that the misuse of a design strategy can result in less effective designs. Shalloway *et al.* [1] proposed the refactoring which improves the internal structure of code while keeping its external behavior as a means of coppering the problem of OD and UD from the perspective of agile development. Dao *et al.* [13] presented a problem solution framework for relating features to properties of non-functional requirements (NFR). They support architecting specific systems with focused features binding to properties of NFRs. Alebrahim and Heisel [14] proposed a UML profile to model NFR embedded problem description and bridge the transfer to solution description with the problem to solution mapping of feature modeling. Most existing approaches are at the methodological level and based on post-active analysis which does not guide the proactive design activities in a system development process. In [15], Czarnecki *et al.* identified gaps in mapping Functional Model to Decision Model. The authors discussed the gap in binding time which originates in the missing of an explicit cognition link between the implementation of binding strategies and quality properties. In addition, the authors ignored the avoidance of the practical situation of UD/OD at functional level, which are main sources of quality issues from a software development perspective.

 

## 7. Conclusion and Future Work

As part of our goal to implement Value Driven design in Software Economics, we plan to use values of the UD and OD to indicate the deviation between ideal design and actual design. This mode eventually constructs a development pattern incrementally filling the gap between system development requirements and the final system. As a first step, we formalize the concept of UD and OD from an abstract level based on a knowledge introduction scenario of a development process. This formalization can be used to accomplish systemic procedures which are difficult if not impossible to be related in design processes. We show its efficiency with an OD and UD eliminating process of a running example. In the future, we would like to construct more efficient and expressive mechanisms to modeling, measuring and calculating OD and UD.

## Acknowledgments

The authors of the paper 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] A. 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, pages, **(2011)**, pp. 248–263.
[2] J. Kerievsky, “Stop over engineering”, Software Development, April **(2002)**.
[3] “2013 IEEE 20th International Conference on Web Services”, Santa Clara, CA, USA, 2013. IEEE, June 28 - July 3, **(2013)**.
[4] T. Al-Naeem, I. Gorton, M. Ali Babar, F. Rabhi and B. Benatalah, “A quality-driven systematic approach for architecting distributed software applications”, In Proceedings of the 27th international conference on Software engineering, ACM, **(2005)**, pp. 244–253.
[5] 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.
[6] “AIAA. Value-driven design”, American Institute of Aeronautics and Astronautics, **(2008)**, pp. 109–109.
[7] B. W. Boehm and K. J. Sullivan, “Software economics: a roadmap”, In Proceedings of the conference on the future of Software engineering, ACM, **(2000)**, pp. 319–343.
[8] “SAVE International”, Value Standard and Body of Knowledge, **(2007)**.
[9] K. Pohl, G. Bockle and F. V. Der Linden, “Software product line engineering”, Springer, vol. 10, **(2005)**, pp. 3–540.
[10] B. Boehm, “Software cost estimation with Cocomo II”, Prentice Hall, Upper Saddle River, NJ, **(2000)**.
[11] L. Carvajal, A. M. Moreno, M. I. S´anchez Segura and A. Seffah, “Usability through software design”, IEEE Trans. Software Eng., vol. 39, no. 11, **(2013)**, pp. 1582–1596.
[12] A. Tang and H. V. Vliet, “Design strategy and software design effectiveness”, IEEE Software, vol. 29, no. 1, **(2012)**, pp. 51–55.
[13] T. M. Dao, H. Lee and K. C. Kang, “Problem frames-based approach to achieving quality attributes in software product line engineering”, In Software Product Line Conference (SPLC), 2011 15th International, IEEE, **(2011)**, pp. 175–180.
[14] A. Alebrahim and M. Heisel, “Supporting quality-driven design decisions by modeling variability”, In Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures, ACM, **(2012)**, pp. 43–48.
[15] K. Czarnecki, P. Gr¨unbacher, R. Rabiser, K. Schmid and A. Wasowski, “Cool features and tough decisions: a comparison of variability modeling approaches”, In Proceedings of the sixth international workshop on variability modeling of software-intensive systems, ACM, **(2012)**, pp. 173–182.

 

## 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.

**Honghao Gao**, he received the PhD 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.

**Jingbing Li**, he received his PhD from Chongqing University, Chongqing, P. R. China, in 2007. He is currently a full professor in Electronics Department in Hainan University, Hainan, P. R. China. His research interests include Image Processing, Artificial Intelligence and Multimedia Technology.

**Mengxing Huang**, he received his PhD degree in computer application from Northwestern Polytechnical University, Shanxi, P. R. China, in 2007. He is currently a full professor and the dean of College of Information Science and Technology in Hainan University. His research interests include knowledge engineering, Big Data, tourism information management and e-commerce.