Variability based Approach for Minimizing over Design and under Design

Yucong Duan[1], Nanjangud C Narendra[2], Honghao Gao[3], Mingdong Tang[4], Abdelrahman Osman Elfaki[5], Shixiang Wan[1], Junxing Lu[1]

[1]*Hainan University, Haikou, China*
[2]*Cognizant Technology Solutions, Bangalore, India*
[3]*Computing Center, Shanghai University, 200444 Shanghai, P.R. China*
[4]*School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China*
[5]*Department of Computer Science, University of Tabuk*

Abstract. In this paper, we have considered Over Design (OD) and Under Design (UD) as two basic forms of negative variability construction that lead to both functional incompleteness and unexpected software quality degradation during design time and run time. To make our approach economically practical to the quality assurance at the whole life cycle of software production process, we aim to ensure that OD and UD are minimized in terms of values under economical goals. These requirements are concretized with the functionality implementation and the quality management of a quality lifecycle.

1 Introduction

In [1], Boehm et al. identified that software development is a value creation process. Usually, in a software development lifecycle, there are gaps between technical decisions and targets of value creation processes. They summarized related topics as an interdisciplinary subject: Software Economics. Software economics seeks to enable significant improvements in software design and engineering through economic reasoning about product, process, program, portfolio and policy issues.

The economical influences of Software Economics on construction and evaluation of a software project can be summarized in four dimensions:

- Ultimate implementation: The composition of quality models determined by economic goals defines the business success of a software project. Quality models are built to reflect the business interests with quality properties and represent the business values with quality values.
- Integrated evaluation: The evaluation of fulfillment of quality properties shall usually be considered as a well-organized whole as considering quality properties independently might potentially impede the process of pursuing another quality target. Thus, a tradeoff between over pursuing and under pursuing a quality target should be identified under goals of Software Economics.
- Dynamic management: During software construction, the configuration of information flow and control flow may influence both efficiency and outcome of value creation process[1]. This demands a dynamic management to flexibly cater the maximization of the created value under the available resources constraints.
- Lifecycle accumulation: Software development lifecycles outline completeness for value modeling and calculation in the temporal dimension. Values accumulated alongside a complete development lifecycle or business lifecycle form the value identities for further comprehensive measurement.

Most software quality implementation serves the ultimate economic goal of software projects on the customer side. Also integrated quality evaluation is subordinated with accomplishment of economic goal towards which a software project is proposed. For example, if designing a chatting software including video chatting function, it will be unnecessary for African to use it because of bandwidth limited. Hence, a simple text chatting software is enough to ensure optimal quality and economic goal.

1.1. Integrating values of process and product quality

In software lifecycle, mismatch between technical decisions and value creation originates usually as they are considered separately in a project management perspective and a classic economic analysis. The value of process quality of a software project is calculated dividedly from values of the software product which is measured partially by its quality fulfillment. This is a practical solution from an engineering perspective.

To proceed towards harvesting the full benefit proposed in software economics, we need to consider the interrelationship among technical decisions and value creations. For example, a design decision leading to well managed late binding[2] can potentially contribute to a project with improvement on both quality properties and business values. These include higher scalability of deployment, enhanced flexibility for extension, and reduced cost for composition according to E-Service contract[3] during service composition[4] through E-Service value brokers [5]. More comprehensive solutions can be designed to integrate values of both process and product quality after exposing association between technical decision and value creation.

1.2. Variability modeling

In the past several years great achievement have been made in the area of design and implementation of quality driven or quality aware [6] for both specific software products and software product families [7]. Prominent success has been attained in practice of software product line and feature modeling [7] , MDA [6]. In such systems, a focus has been given to variability modeling that considers various superficial quality issues presented as features. In this paper, we detail the variability modeling of design variation from stage to the whole process to reveal the implementation of the economical business goal in terms of quality properties connected to technical decisions. This involves identification of quality properties related to technical implementation and leverage evaluation from stages based on temporary quality properties to the entire life cycle following process based quality model reflecting economical goal of a software project.

In a waterfall development process, a quality model is assumed to exist completely at the end of requirement specification. However, in the reality this experience does not fit for iteration based process models such as RUP or agile in which the software may suffer from constant modification. To cope with these kinds of situations and improve software quality implementation, there is a need to consider quality model evolution and identify variability management.

We propose an approach called "Problem-Quality-Constraint" (PQC), as a solution strategy for quality driven/aware software modeling. PQC is a constraint based modeling solution to manage variability towards satisfying the requirement of quality model that bears a process or lifecycle affiliated to the development process. Our approach inculcates stakeholders' assessment, agreement and economic values in quality life cycle.

The rest of the paper is organized as follows: Section 2 analyzes and models scenarios related to quality from the perspective of introduction of information, quality lifecycle and economics. Section 3 presents our strategy "Problem-Quality-Constraint". Section 4 shows an initial case study on a scenario of Over-design and Under-design and experimental results. This is followed by related works in Section 5 and in Section 6 conclusions and future directions.

2 Running example

In this section, we describe the ideal design, actual design, over design and under design in detail by an example of a bank in system.

As per Figure 1, we have demonstrated a banking system to illustrate variability based approach. The whole product process is divided into three processes including basic process, middle process and final process. Empirically, x-axis is identified as product process of development. We have confirmed an ideal line, in which seven targets belong to three processes representing different parts of system implementation. Two different colored lines (the orange line and purple line) separately shows two designs that are away from the ideal line. The parameter "cost" is chosen to calculate the deviation between actual and ideal lines, which is reflected in y-axis. In order to measure the cost of each process better, we assume that some numerical values represent the different levels of each target. Besides, there are two dotted lines on both sides of ideal line, and the variability space comes into being between them.

ID: means Ideal Design

AD: means Actual Design

ND: means Negative Design

VUB: means Variable upper bound

VLB: means Variable lower bound

SPL: means Software Product Line

Table I. The value of all targets

Factor SP	Security	Stability	Authority distributio n	Emergency capability	Response time	Throughpu t	Satisfactio n
ID	10	11	8	9	10	10.5	9.5
AD	10	10.5	9	8	11	10	10
ND	10	13	5	11	8	11	9
VUB	11.5	12.5	9.5	10.5	11.5	12	11
VLB	8.5	9.5	6.5	7.5	8.5	9	8

DOI: 10.1051/matecconf/20165407002

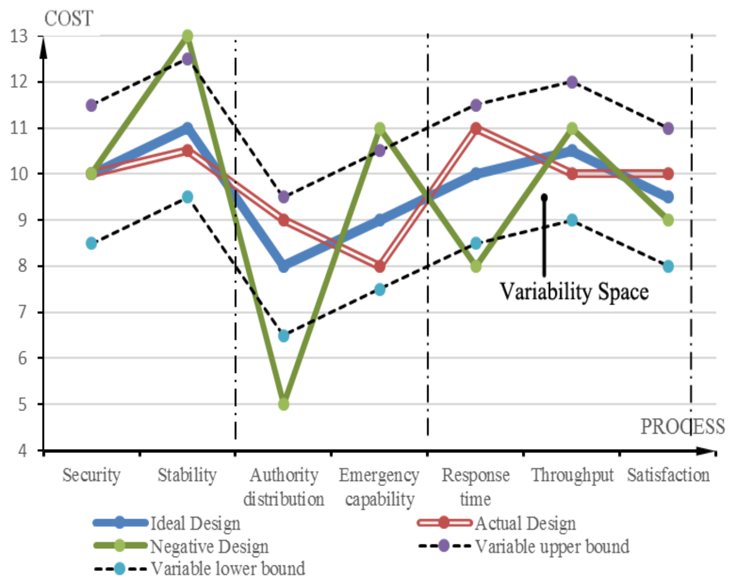

Figure 1. Idea design and actual design

Hence, the quality S_{PR-QU} of a software process line can be formulated as:

$$S_{PR-QU} := \sqrt{\frac{\sum_{i=1}^{n}(C_i - C_{ID})^2}{n}}. \qquad (1)$$

Where:

- C_i: actual cost of the target
- C_{ID}: ideal cost of the target
- n: number of all targets

Table II. The values of S_{PR-QU}

Target	Ideal Design	Actual Design	Negative Design	Variable upper bound	Variable lower bound
S_{PR-QU}	0	0.732	1.75	1.5	1.5

The numerical value of SPR-QU shows degree of deviation during the software process line. The larger it is, lower the overall product quality, and vice versa. By this quality index SPR-QU, a comprehensive solutions can be designed to maximize both process and product quality based on association between technical decision and value creation.

3 Analysis of fundamental issues

3.1. Criteria of Information Introduction

In software product lines, it is not easy to set priority among several modeling decisions. But, the use of constraint based approach helps in such cases. The enforcement of creating a priority among difference modeling decisions without concrete proof will bear the risk of Over-design (OD) and Under-design (UD).

Definition 1: [Over Design]. The design introduces unnecessary information for directly promoting the intended goal. This surplus information bears the risk of refraining quality improvement in subsequent stage and may reduce the variability space of subsequent modeling.

Definition 2: [Under Design]. The design lacks some necessary information for the demand. This lost information extends the variability space of software modeling decision, which adds the software product cost.

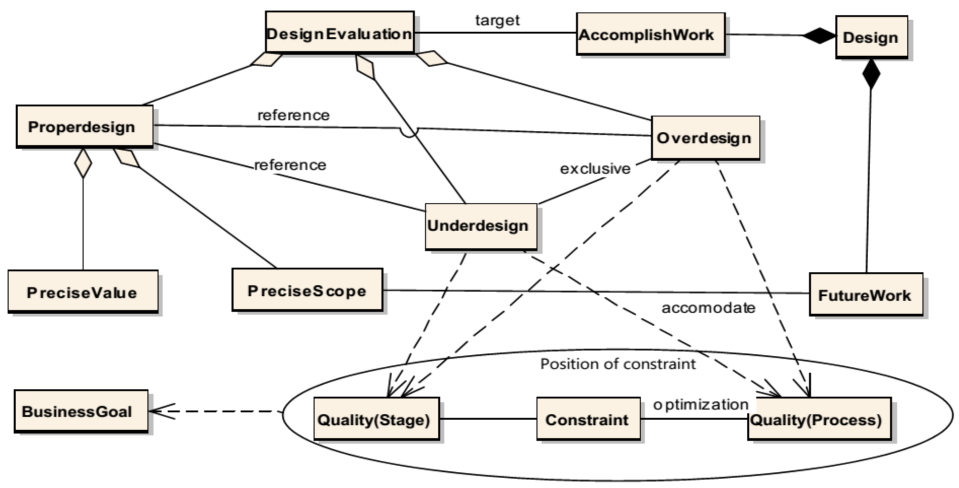

Figure 2. Constraint driven design

Over Design/OD is a form of Negative Information Introduction (NPII) [8]. Considering system design from perspective of dichotomy, the deviation (ΔD) in design is the discrepancy between the ideal design (IDD) and the actual design (ACD). It is denoted as:

$$\Delta D = IDD - ACD$$

There are two situations associated with ΔD:

- If ΔD>0, then ΔD belongs to IDD, which implies that ΔD is a space of Under-design (UD). i.e., the actual design is quite below the ideal design of system.
- If ΔD>0, then ΔD belongs to ACD, which implies that ΔD is a space of OD. i.e., the actual design of the system is beyond the expectation of ideal system design.

Then the problem facing a design/implementation is trying to reduce the space of ΔD denoted as ΔPD which is the combination of space of under and over design as formalized below

$$\Delta P_{\mathrm{D}} = \Delta P_{OD} \cup \Delta P_{UD}. \qquad (2)$$

Where

- UD space ΔP_{UD} represents the unaccomplished design space that can be filled later on.
- ΔP_{OD} is an OD space.

Figure 2 illustrates the relationships between IDD, OD and UD.

3.2. Essence of quality: quality lifecycle

3.2.1. Hypothesis on quality

The quality properties considered in this work are assumed to be orthogonal, i.e., they do not affect each other.

3.2.2. Quality lifecycle

Software Quality is defined mostly in terms of functional, quality attributes/property and resource specifications. The last two specifications are related to the non-functional requirement of system. Quality assessment has to be done throughout the software life cycle and considered as quality lifecycle (QL):

- Design stage
- Execution stage
- Maintenance stage
- Economic analysis stage

At each stage (STG) of QL, quality properties (QP) are of interest to various Stakeholders (SK) and are evaluated using various criteria (CR). Table I shows the quality properties QP and associated evaluation criteria CR. An agreement (AG) is usually a compromise after a series of adaptation/change among stakeholders. During a QL, the agreement of a specific quality property may be changed from stage to stage. The agreement on quality properties will be the reference of quality driven modification on the implementation of software system. Figure 2 shows that UD and OD influence both quality of process and quality of final output. In this work, we propose to integrate these as per the principles software economics to bridge the implementation of UD and OD at individual stages. And UD and OD are evaluated according to the quality model of the whole process as presented next.

Table III. Software quality criteria

Quality property (QP)	Software Quality Criteria (CR)
Correctness	Traceability, consistency, completeness
Reliability	Error tolerance, consistency, accuracy, simplicity
Efficiency	Execution efficiency, storage efficiency
Integrity	Access control, access audit
Usability	Operability, training, communicativeness, input/output volume, input/output rate
Maintainability	Consistency, simplicity, conciseness, modularity, self-descriptiveness
Testability	Simplicity, modularity, instrumentation, self-descriptiveness
Flexibility	Simplicity, modularity, instrumentation, self-descriptiveness
Portability	Modularity, self-descriptiveness, machine independency, software system independence
Reusability	Generality, modularity, software system independency, machine independency, self-descriptiveness
Interoperability	Modularity, communications commonality, data commonality

3.2.3. Quality Economics

In the section II, we create a running example about ideal design and actual designs. From the value analysis perspective and based on the running example, we derive the goal of quality driven design in three steps:

1. Tradeoff for each stage - at the tradeoff point of quality negotiation, stakeholders reach agreement on quality properties of a stage. A general tradeoff point is the state where the sum of the satisfaction of all stakeholders reaches maximum. Thus, the tradeoff point of a quality property among stakeholders in a stage is formulated as:

$$AG(qp_i)|_{STG} ::= \max\left(\sum_{j=1}^{n} qp_i \times w_{SK_j}\right). \qquad (3)$$

Where:

- $qp_i \in N$ – the ith quality property
- w_{SK_j} : weight given to the jth stakeholder

To reach the $AG(qp_i)|_{STG}$, a change has to be made on the design, implementation or the processing itself. This kind of change is denoted as CG-STG and its variability space is denoted as $CP_{CG\text{-}STG}$.

2. Tradeoff on evolving stages - Demands on a quality property from a stakeholder may be evolved from stage to stage in a QL, as well as the agreement of quality properties among stakeholders. During a QL, the aim is to maximize overall quality, and this can be formulated as:

$$AG(qp_i)|_{QL} ::= \max\left(\sum_{k=1}^{m}\left(\sum_{j=1}^{n} qp_i \times f_k\left(w_{SK_j}\right)\right)\right) \tag{4}$$
$$== \max\left(\sum_{k=1}^{m} AG(qp_i)|_{STG=k}\right).$$

Where

- $qp_i \in N$ –a quality property
- $SK_j \in N$ – a stakeholder
- $k \in N$ –a stage
- w_{SK_j} : weight given to the jth stakeholder
- $f_k(\ )$ is the modification function at the k^{th} stage on the weight given to the jth stakeholder

To reach the $AG(qp_i)|_{QL}$, a change need to be made in a holistic manner on the distribution of design, implementation or processing decisions as interrelated across stages. This kind of change is denoted as *CG-QL*. The variability space of this kind of change is denoted as $CP_{CG\text{-}QL}$.

3.2.4. Transformation from short run to long run

From the problem solution perspective, the optimization action (OA) of design/implementation decision on quality properties is a form of change which belongs to either CPCG-STG or CPCG-QL. CPCG-QL is not a simple sum of individual CPCG-STG: $\sum(\mathrm{CP}_{CG-STG})$, instead it is expected to be an optimization of $\sum(\mathrm{CP}_{CG-STG})$ based on the global business strategy. From the long run vs. short run point of view [9], the ultimate goal of a design process is maximizing the global business value related to the project (GBV) which can be implemented through adaptation on CPCG-STG towards forming CPCG-QL. This process is denoted as: $OA(CP_{CG-STG}) \Rightarrow CP_{CG-QL}$. Existing quality driven approaches have been introduced to handle individual CPCG-STG [26].We would like to expand it from individual CPCG to a whole CPCG-QL.

4 Problem-quality-constraint

Our solution of lifecycle quality assurance assistance for software projects which is called Problem-Quality-Constraint (PQC) is composed of three main components:

4.1 Problem modeling

The problem model associated to a software project can be modeled using function or process modeling.

- Functional modeling [7] at an abstract level - Actions in a system construction are expected to contribute positively to fill the gap between project schedule and final ideal system. We model the reference to modeling directions as deviations from the ideal model which bears the maximization of satisfaction of quality requirements. The reference is distinguished as over-design and under-design. Reducing the spaces of over-design and under-design will aid in attaining the satisfaction of quality goals.
- Process modeling - we model the optimal variability space to make it correspond to stages and lifecycle from the perspective of economic quality. For example, we shall close to ideal design line when designing actual line by the process including technical decision and value creation.

4.2 Quality model

Keeping the intended space of variability is of interest to some stakeholders who work to meet non-functional requirements (NFR) of a system or its design process. These represent a form of quality requirement and denoted as NFRV. In addition to explicitly describe NFRV the most important goal of design is to reduce the variability space of quality lifecycle by reducing the space of under-design and avoiding increment of the space of over-design. We identify this as a quality requirement in the development process by transforming design knowledge/ information from stakeholders to model. It is denoted as: NFRP. A simple quality model (QMS) is adopted following works in [10], which use the aggregation of NFRV and NFRP as precondition to meet system implementation. It is formulated as.

$$QM_S := NFR_V \cup NFR_P. \tag{5}$$

In Figure 3, quality stages are modeled as three common targets towards minimizing the quality loss of modeling practice before the accomplishment of a software project:

- Maximize the scope of known information in the models. The excess in the boundary of proper design will turn into OD practice.
- Minimize the scope of unknown information in the models. The excess in the boundary of proper design will turn into UD practice.
- Maintain the intended variability or scopes in models which might be used for satisfying various quality properties.

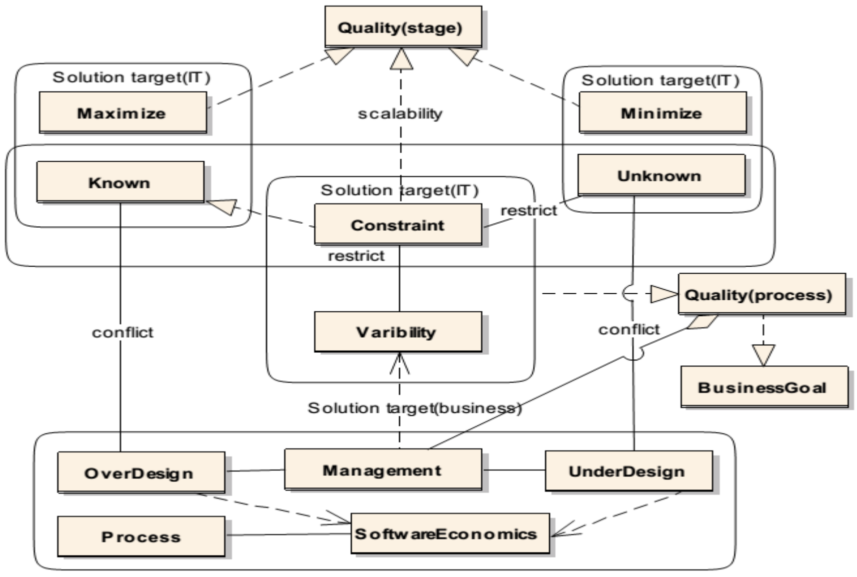

Figure 3. Constraint based solution framework

4.3 Constraint Based Solution

There are various understandings of the concept of constraint for design:

In [11] constraint is defined as the relationship between set of design variables that are related to each other. A relation between independent design variables represents a simple constraint whereas a relation between dependent design variables represents a complex constrain. This can be implemented by using constraint modeling language – OCL in UML. It is defined as expressions typically specifying invariant conditions that must hold for the system being modeled or queries over objects described in a model[1].

In this work, constraints are defined as expressions of relationships among modeling elements for restricting the unknown space in a constraint space (CRP) to form a product which minimizes the chances of OD and UD. The OD and UD are measured according to an evolutionary quality lifecycle which concretizes the goals of software economics.

From a general sense, relationships among modeling elements can be viewed as implicit form of constraints. For example, among UML relationships [12] - *association*, *aggregation* and *composition*, without a specific project context, *association* accommodates a CRP which contains both *aggregation* and *composition*. Using *association* instead of either *aggregation* or *composition* leaves space for choosing a fitting implementation with desired quality without incurring OD.

Constraint used to construct a variability space or a CRP avoids the drawback of increasing ΔP_{OD} with a prudence decision.

CRP accommodate multiple choices which might make a difference in terms of influencing AG(qpi)|STG. However, if the choices have different influences on CPCG-QL, CRP contributes to following two aspects:

- The package of choices alleviates the risk of increasing ΔP_{OD}.
- Postponing of a choice effectively reduces the risk of increasing ΔP_{OD}

Technically CRP is the source of many design patterns [5], [13] and can be used to plan quality aware later binding [2]. To assure that CRP contributes positively, the benefit of reducing risk of ΔPOD shall surpasses the loss of performance such as in later binding. The concrete planning relies on the specific tradeoff among quality properties.

5 Case study

In this section, we share our intermediate results through practicing PQC and experimental value.

5.1. Feature based modeling of e-commerce

We adapted the feature model of [15] to create Figure 3 in which service is implemented from up to down. There are two types of structures in Figure 3: Static and Dynamic. In Static, all components are static service choices. On the contrast, we propose the new concept of Dynamic structure, which is a sequential services bundle. Obviously, the sum of all cost and value-added includes all selected components cost and value-added in the variability construction.

Variant of "Validation" in design (B) with optional and is OD relative to design (A) with mandatory as validation is required in every E-commerce process.

Variant point of "Translation" in design (B) represented with "alternative" is UD relative to "Or" in design (A) as several of languages may be selected at a time.

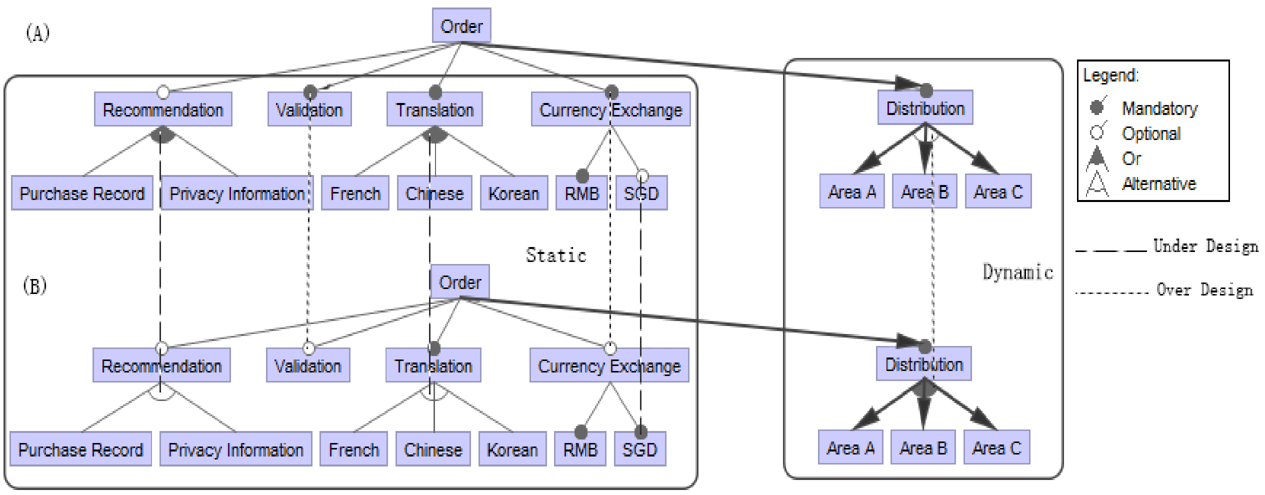

Figure 4. Ideal-design (A) vs. under-design and over-design (B) in a feature model

Table IV. Value added to the design vs. cost tradeoffs

Component	Cost	Value Added
Order	200	2
Recommendation	100	1
Validation	200	2
Translation	300	3
Currency Exchange	400	4
Distribution	100	1
Purchase Record	200	2
Privacy Information	200	2
French	100	1
Chinese	100	1
Korean	100	1
RMB	300	3
SGD	200	2
Area A	300	3
Area B	300	3
Area C	300	3

Variant point of "Distribution" in design (B) represented with "Or" is OD relative to "Alternative" in design (A) as only one distribution broker will be selected at a time.

The above analysis is based on quality criterion of precision of design from the perspective of a design stage. From the real project perspective, precision decisions are expected. However it is not practical or feasible for many situations. The coping strategy is "design for change" i.e., there is a need to leave spaces to positively accommodate "unknown" to avoid easily fall into the category of either OD or UD. We can identify the case that seeking a precision decision is very expensive, "Optional" in design (A) is better than "Alternative" in design (B). This also is in line with the commonsense knowledge "unknown" is more precise than "wrong" or "improper".

To elaborate the difference between ideal design and actual design including OD and UD, an experiment is demonstrated by building a calculating model about getting the sum of all component values. We utilize the following tradeoff for quality metrics: Cost vs. Value Added. Cost refers to the general expense of design. Value added refers to the gains of price or value brought by a design implementation. An increase in cost produces more values added features to the design. This is provided for the design elements in Table IV.

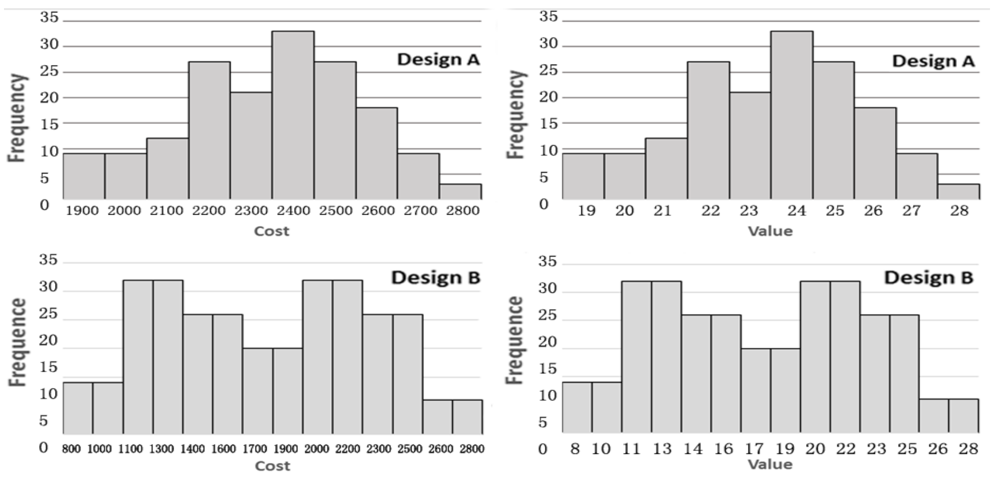

Figure 5. Histograms of cost and value provided by exhaustive combinations

As shown in Figure 4, we see there are many difference about the cost and value added between design (A) and design (B). We assume that design (A) is the ideal design, then we get design (B) in the case of UD and OD. Obviously, the design (B) has a greater volatility than the design (A). Besides, the interval of design (B) is bigger than design (A), which shows the actual design including UD and OD is less instable than the ideal design. In other words, UD and OD severely affects quality of the software system. Hence, UD and OD are nagative design models, we should do our best to make the reachment: $\Delta P_D \to 0$.

6 Related work

Dao et al. [17] presented a problem solution framework for relating features to properties of non-functional requirement(NFR). It supports architecting specific systems with focused features binding to properties of NFR. Alebrahim and Heisel [18] proposed a UML profile to model NFR embedded problem description and bridges the transfer to solution description with the problem to solution mapping of feature modeling. Most of 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. However, our approach provides systemic guide for designing directly by leveraging a constraint driven design approach.

Eiffel language implemented contract based implementation [19]. Boogie language [20] allows specifying pre- and post- conditions to accommodate nondeterministic choices in program flows. However, both of them are restricted to code level and do not fully cover software design activities that span across processes.

In [15], Czarnecki et al. identified gaps in mapping Functional Model to Decision Model. The authors discussed the gap in binding time. This is partially because of 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 over-design vs. under-design at functional level, which are main sources of quality issues from a constructive perspective. Our approach can be applied to fill such gaps as it models the quality assurance based on the quality properties derived from functional level analysis. In particular, QPC supports the bridging of this gap with quality property identification and quality model construction steps.

Demuth et al. [21] proposed constraint based approach to accommodate the choice facing model transformation in a side-effect free manner. Our approach leverages the constraint based modeling from aiding a design to driving the main process of a design.

7 Conclusion and future work

Enlightened by quality mapping and modeling stated in [17], [18], this paper presents a strategy for assuring quality in the software development process, called "Problem-Quality-Constraint" (PQC), where business value constrains are defined. We aim to fill the gap between the schedule and final ideal system using well defined quality goals throughout the software lifecycle. The proposed approach will be usable in incremental iterations process model, such as RUP. Thus, PQC runs in iterations taking into account stakeholder's participation in assessing quality attribute. Stakeholder participation is important to reach agreement both at given stage and evolving stages which are more difficult to quantify.

The key contribution of this paper can be summarized as follows.

- We introduced software economics driven approach that adopt a holistic view to integrate achievements on quality considering criteria on quality tradeoff, quality modeling, quality evaluation and tradeoff, during a quality lifecycle.
- Instead of mapping quality properties to functions at higher level in a retrospective and top-down manner, we propose to start from model the basic form of design quality analysis at the level of over-design vs. under-design.

Over-design (OD) should be avoided by postponing some decisions to later stages as "unknown" which is more precise than "wrong" or "improper".

The introduction of software economics perspective helps relating quality properties that are considered as unrelated and categorized as process quality and product quality. This fits for both proceeding towards quality modeling and retrospective modification of a software product. From the basic form of quality analysis in the form of UD and OD, we can obtain the clue to make quality aware design decisions.

We have identified that OD deserves more attention. OD is a result of rigidly fulfilling precision design ideology by making design decisions which introduces potentially more loss of quality than gains in a quality lifecycle. From the perspective of process quality assurance, we have identified that constraint based modeling [22], [23] manage to accommodate abstract or temporarily not distinguishable information of design decisions. It can postpone the decision to a later stage while balancing the loss of performance. This will contribute directly inevitable choices which are invulnerable to either over-design or sub-design of functions which initiates the deficiency of software quality. Thereafter we propose that a corresponding solution lies in employing constraint as a major form of design activity which is called constraint driven design [24].

In the future, we would like to see PQC contributes directly to the design, composition, runtime, maintenance and evaluation of corresponding systems integrating both functional and non-functional aspects of a system under the constraints of expected business value.

Since PQC is based on modeling related to change of quality directly, it can be used to supplement the implementation of higher level decisions as presented in [17], [18]. PQC will also fit the emerging trend of system design, such as MASHUP [25] in which systems are built on top of existing services with independent functions and corresponding QoS documented in their e-Contracts, and service broker patterns (SVB) [5] which embodies the maximization of business value of a system design.

This paper has considered quality assurance that spans over the entire life cycle of a project focusing on software economics and variability techniques. At the next stage, we will explore the open question of evaluating the level of OD and UD through measurement related to design.

Acknowledgment

This paper was supported in part by CNSF grant 61363007 and by HNU Research program grant KYQD1242 and HDSF201310.

References

1. B. W. Boehm and K. J. Sullivan, "Software economics: a roadmap," in Proceedings of the Conference on The Future of Software Engineering, pp. 319–343, 2000.
2. C. Pautasso and G. Alonso, "Flexible binding for reusable composition of web services," in Proceedings of the 4th international conference on Software Composition, pp. 151–166, 2005.
3. Y. Duan, "A Survey on Service Contract," in Proceedings of the 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 805–810, 2012.
4. A. Kattepur, A. Benveniste, and C. Jard, "Negotiation Strategies for Probabilistic Contracts in Web Services Orchestrations," in Proceedings of ICWS2012, pp. 106–113.
5. Y. Duan, A. Kattepur, H. Zhou, Y. Chang, M. Huang, and W. Du, " Service Value Broker Patterns: An Empirical Collection," in SNPD2013, 675-682.
6. M. L. Drago, C. Ghezzi, and R. Mirandola, "A quality driven extension to the QVT-relations transformation language," Computer Science, vol. 27, no. 2, 2012.
7. K. Lee, K. C. Kang, and J. Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," Springer Berlin Heidelberg, pp. 62-77, 2002.
8. Y. Duan and R. Lee, "Knowledge Management for Model Driven Data Cleaning of Very Large Database," Ed. Springer-Verlag, pp. 143–158, 2012.
9. M. Feldstein, "Domestic saving and international capital movements in the long run and the short run," European Economic Review, pp. 129-151, 1983.
10. L. Chung, J. Cesar, and S. P. Leite, "Non-functional requirements in software engineering," Springer Berlin Heidelberg, pp. 363-379, 2009.
11. A. Nassaj and J. Lienig, "A New Methodology for Constraint-Driven Layout Design of Analog Circuits," pp. 996–999, December 2009.
12. O. M. Group, "OMG Unified Modelling Language (OMG UML), Infrastrucgture,".
13. E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides, "Design Patterns: Abstraction and Reuse of Object-Oriented Design," in Proceedings of the 7th European Conference on Object-Oriented Programming, pp. 406–431, 1993.
14. D. Garlan "Software architecture: a raodmap," in Proceedings of the Conference on the Fureture of Software Engineering, pp. 91-101, 2000.
15. K. Czarnecki, P. Grünbacher, R. Rabiser, and K. Schmid, "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, pp. 173-182, 2012.
16. P. Y. Schobbens, P. Heymans, and J. C. Trigaux, "Feature Diagrams: A Survey and a Formal Semantics," in Proceedings of the 14th IEEE International Requirements Engineering Conference, pp. 136–145, 2006.
17. T. M. Dao, H. Lee, and K. C. Kang, "Problem Frames-Based Approach to Achieving Quality Attributes in Software Product Line Engineering," Proceedings of the 2011 15th International Software Product Line Conference, pp. 175–180, 2011.
18. A. Alebrahim and M. Heisel, "Supporting quality-driven design decisions by modeling variability," Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures, pp. 43–48, 2012.
19. M.B. Eiffel,"A language and environment for software engineering," Journal of Systems and Software, pp. 199-246, 1988.
20. M. Barnett, B. E. Chang, R. Deline, B. Jacobs, and K. R. Leino, "Boogie: A modular reusable verifier for object-oriented programs," in Formal Methods for Components and Objects: 4th International Symposium, FMCO2005, LNCS, pp. 364–387, 2006.
21. A. Demuth, R. E. Lopez-herrejon, and A. Egyed, "Constraint-driven Modeling through Transformation," Springer Berlin Heidelberg, pp. 248-263,2012.
22. A. Demuth, R. E. Lopez-Herrejon, and A. Egyed, "Constraint-Driven modeling through transformation," in Proceedings of the 5th TPMP, pp. 248–263, 2012.
23. A. Egyed and D. S. Wile, "Support for Managing Design-Time Decisions," IEEE Trans Softw Eng, vol. 32, no. 5, pp. 299–314, May 2006.
24. K. Lano, "Constraint-driven development," Inf Softw Technol, vol. 50, no. 5, pp. 406–423, April 2008.

25. M. Ogrinz, "Mashup Patterns: Designs and Examples for the Modern Enterprise," 1st ed. Addison-Wesley Professional, 2009.
26. T. Al-Naeem, I. Gorton,M. Babar, F.A. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," ICSE2005, pp. 244-253.
27. Y. Duan, A. Kattepury, F. Getahun, A. Elfaki, and W. Du, "Releasing the Power of Varibility: Towards Constraint Driven Quality Assurance," AAI2013. pp. 15-20.