Pakistan Journal of Applied Sciences 2 (3):356-365, 2002

A Preliminary Correctness Evaluation Model of Object-Oriented Software Based on UML

S. Wesley Changchien, Jau-Ji Shen and Tao-Yuan Lin
Department of Information Management, Chaoyang University of Technology
168 GiFeng E. Rd., WuFeng, Taichung County, Taiwan, R.O.C.

Abstract: Concurrent engineering is a philosophy that attempts to take into account of all the activities of a product life cycle early in the design stage. In the manufacturing industry, approximately 70% of a product's manufacturing and assembly costs are determined during the design stage. Similar to the software industry, system analysis and design has significant influence on later activities of the software development life cycle. Object-oriented approach has been the main stream for software development, and Unified Modeling Language (UML) integrates most of the object-oriented modeling methods and has become the standards. This paper incorporates the CE concept into the evaluation of object-oriented software development and proposes a Hierarchical Aggregation Model (HAM) to early evaluate the object-oriented software design quality based on UML. There are three advantages of using this model. First, this model can help reduce the project risk, cost, and time span, and eventually improve the software quality early in the software development life cycle. Secondly, this model facilitates the use of the standards of object-oriented modeling language, UML, which makes proposed model more applicable to real software development. Thirdly, this model is easy to implement, that can potentially be imbedded in CASE tools to directly support the project manager's decision making.

Key Words: Concurrent Engineering, Software Development, Object-oriented Software Metrics, UML

Introduction

In the manufacturing industry, approximately 70% of a product's manufacturing and assembly costs are determined during the design stage (Grady and Oh, 1991). In other words, if designers have sufficient knowledge about how design will affect manufacture and assembly in the later stage, the cost and risk across the product development life cycle will be significantly reduced. Similarly, if a methodology can take into account later activities in the software development life cycle early in the system analysis and design, it has the potential to improve the developed software quality. Concurrent Engineering (CE) is the philosophy to realize this concept.

Tom DeMarco (1987) asserts "You cannot control what you cannot measure." In order to effectively evaluate how CE is implemented on software development, the software measurement of system analysis and design is hence a key point. In general, for software development there are three issues to be examined that are process, product and resource. Based on the CE concept, in this paper we focus on the product especially the system analysis and design to evaluate the software correctness early in the software development life cycle. Currently, UML (Unified Modeling Language), which integrates the major object- oriented modeling methodologies, has become standards and been more and more adopted by the software industry and hence is a basis for evaluating object-oriented software design.

With UML, the key objective of this paper is to propose an preliminary evaluation model, hierarchical aggregation model (HAM), for object-oriented software using CE concept. This model can help the project manager evaluate the software product quality in terms of correctness early in the object-oriented software development life cycle.

Literature Review: In the 1990's, object-oriented technology is the main stream for software development. During this period, there is a good number of object-oriented modeling methodologies proposed and each methodology has its own modeling notation. UML, which integrates the major methods, provides the source data for evaluating object-oriented software design using CE approach. In the Following, we review the related literature in CE, UML, and software development measurement.

Concurrent Engineering (CE): The traditional activities of a product development cycle are usually implemented sequentially and independently. Any problem occurring in the later stage will require modification or repeat of prior activities. This consequently results in excessive time and cost of product development due to the isolated activities. Approximately 70% of a product's manufacturing and assembly costs are determined during the design stage based on the experiences in the manufacturing industry (Grady and Oh, 1991). "CE is a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from conception through to disposal, including quality, cost, schedule, and user requirements" defined by Winner (Winner, 1988). IBM, AT&T and Hotpoint implemented this concept successfully and benefits include reducing 30%-70% of the development time and 20%-90% time to market, increasing overall quality 200%-600%, productivity 20%-115%, and sales 5%-50%, etc. (Madhat and Rook, 1997). The literature shows that CE has been effectively applied to the product design in the manufacturing industry. Motivated by the CE concept, this paper presents a concurrent design evaluation method for object-oriented software design.

To implement CE, Swink (1998) recommends two aspects, which are to improve cross functional integration and to improve design analysis and decision

making. Therefore, the proposed HAM uses cross-functional aspect to improve decision making early in software development process.

Unified Modeling Language (UML): In the past decade, there is a great number of object-oriented methodologies purposed. Among those, there are three popular methods, OMT (Rumbaugh) (1991; 1997), Booch (1995; 1994), and OOSE (Jacobson et al., 1992), each method with its own value and emphasis. OMT is good for analysis, Booch emphasizes on design, and Jacobson facilitates behavior analysis. And each method has its own modeling notation. These three methods now have contributed and been integrated into UML such that standards can be used for software designers. Booch, Rumbaugh, Jacobson, Meyer, Harel, Wirfs-Brock, Fusion, Embly, Gamma, Shlaer-Mellor, Odell have all partaken in the UML growth (Eriksson and Penker, 1998; Quatrani, 1998). This paper takes advantages of UML's standards on which the developed CE approach to object-oriented software evaluation model is based.

Software development measurement: Since Rubey and Hartwick proposed software measurement in 1968, a good number of research has been conducted (Briand *et al.*, 1996; Gonzalez, 1995 and Henry and Kafura, 1981). Fenton (1991) classifies the software metrics objective into internal, external and less objective metrics. Henderson-Sellers (Henderson, 1996) takes one step further to describe the relation of internal and external in detail (Fig. 1). Both of internal and external characteristics can be measured or estimated directly. There are three ways of estimating external characteristics by internal measures. If the internal characteristics can be measured directly, paths A and B can estimate the external characteristics by internal characteristics. A uses correlation techniques and B uses underlying conceptual model (ex. Macall Quality Model) for estimation. If the internal characteristics can not be measured directly, path C first estimates the internal characteristics, then path AC employs correlation technique to estimate the external characteristics. In general, the internal characteristics include software size, structure, etc., while external characteristics include reliability, correctness, usability, etc. This research aims to propose a method to evaluate the external characteristics from estimated or measured internal characteristics.

Although there are many quality factors in software development, controlling all of the factors is, however, not practical. Goal-Question-Metric (GQM) (Fenton and Pfleeger, 1996) is developed to solve this problem. When a team starts a software project, firstly it must set its goals for the project, secondly it must answer the questions of each goal, and thirdly it must decide what metrics must be measured in order to answer the questions. Therefore GQM can help the software developer identify what quality factors must be measured. This concept motivates us to develop our HAM:

Similarly, in object-oriented software measurement there have been a good variety of software metrics proposed in the literature (Chidamber and Kemerer, 1998; 1994; 1991; Harrison *et al.*, 1998 and Li, 1998). Chidamer and Kemerer (1998; 1994 and 1991) have presented a good number of literatures. Their proposed theory is developed based on Bunge's (1977 and 1979) ontology theory and Roberts relational system. Chidamer and Kemerer (1994) purposed six class-based design metrics, whose definitions are described below.

1. Weighted Methods per Class (WMC): Consider a class C_1, with methods M_1, ..., M_n that are defined in the class. Let $c_1,\ldots,c_n$ be the complexity of the methods. Then WMC of C is defined as:

$$WMC = \sum_{i=1}^{n} c_i$$

if all method complexities are considered to be unity, then WMC equals n, the number of methods.
2. Depth of the Inheritance Tree (DIT): Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.
3. Number of Children (NOC): NOC is the number of immediate subclasses subordinated to a class in the class hierarchy.
4. Coupling between Object Class (CBO): CBO for a class is a count of the number of other classes to which it is coupled.
5. Response for a Class (RFC): RFC=| RS| where RS is the response set for the class.
6. Lack of Cohesion in Methods (LCOM): Consider a Class C1 with n methods M_1, M_2,..., M_n. Let $\{I_j\}$be the set of instance variables used by method M_i . There are n such set $\{I_1\},\ldots,\{I_n\}$. Let $P = \{ (I_i, I_j) \mid I_i \cap I_j = \varnothing \}$ and $Q = \{ (I_i, I_j) \mid I_i \cap I_j \neq \varnothing \}$. If all n set $\{I_1\},\ldots,\{I_n\}$ are $\varnothing$ then let $P = \varnothing$.

According to the discussions of the six class based design metrics (Chidamber and Kemerer, 1994), it is found that WMC, DIT, and RFC are related to system design complexity. Therefore, they are the internal characteristics can potentially be used to evaluate external characteristics, such as correctness for object orient software design.

$$LCOM = |P| - |Q|, \text{if } |P| > |Q|$$
$$= 0, \text{otherwise}$$

Proposed Evaluation Model: The main difference between traditional software development approach and CE approach is that CE incorporates the knowledge or concerns of later activities early in the software development life cycle. As shown in Fig. 2, CE approach introduces a software design evaluation model to reduce the software development risk in terms of correctness early in the software development life cycle.

Hierarchical Aggregation Model (HAM): Card (1990) mentions the relation between system design complexity and error rate and concludes that high complexity will lead to high error rate. Clearly, the complexity is a critical factor that affects the correctness of a software product. Therefore, the proposed HAM hopes to support the manager to use the object-oriented design metrics, complexity, to estimate the correctness of a developed product early in the software development life cycle. The correctness in the proposed approach is defined as "the probability that a developed software functions correctly based on the metrics contributing to the complexity of

the software design" Based on the CE concept, considering the downstream processes' problems in the system analysis and design stage can reduce the probability that errors may occur. In short, in the design stage this model can preliminarily evaluate the developed object-oriented software quality in terms of correctness.

Prior to defining the HAM model, two basic assumptions must be made:

Assumption 1: Early in the system analysis and design stage, the future operations of the software can be precisely understood and predicted.

User requirement volatility is a common problem in software project development. But in order to reduce the risk due to uncertain requirements, sufficient communication with user and precise determination the future operations in analysis and design stage is necessary. Therefore, based on the above assumption the proposed method assumes that the project manager can precisely understand the future operations of the software. Under the determined standard operations, the HAM can evaluate the software correctness early in design stage.

Assumption 2: The system design complexity affecting the downstream processes of the software development life cycle is the major factor concerned which affects the developed software correctness.

Based on the relation between system design complexity and error (Card, 1990), this research assumes the inferiority of the software correctness is mainly due to the increase of complexity of software design.

The concept of HAM is to take user's aspect to interpret object-oriented software correctness. From the user's aspect, software is expected to provide high correctness when each standard operation is practically performed. Thus the software correctness depends on all the correctness of the standard operations being performed on the software. From the design aspect, if all entities in the standard operation can be controlled, the operation correctness can be estimated. In object-oriented software design, Class (C) is the basic entity. Therefore, metrics measured based on classes are the basis for the proposed HAM. HAM hierarchically decomposes an object-oriented software into four layers: OOP, UC, SD, C, defined as follows.

Definition 1:

a. SD (Sequence Diagram) is a set of Classes ordered in a specific sequence according to how classes are triggered to perform the corresponding task (user's standard operation).
b. UC is a set of all SD's belonging to the corresponding function. When a function is modeled using UML, it can be regarded as a Use Case (UC). Each function consists of a number of tasks, hence a UC is composed of a number of Sequence Diagrams to perform those tasks.
c. OOP is a set of all the Use Cases contained in the Object-Oriented software Product. Each OOP is composedáof a number of functions. Therefore an OOP consists of a number of Use Cases.

HAM employs the probability that a software executes properly to derive the finished software correctness. In Definition 2, the OOP correctness is defined by aggregating the correctness of the components of the four layers as described in Definition 1 (Fig. 3).

Definition 2: For a given M_j and Class i,

Class Correctness for Class i= 1 to t

$$CC_i = 1 - \left| FE(M_j) \right| \quad (1)$$

where
Class Diagram i belongs to Sequence Diagram k.
M_j is an object-oriented metrics j that contributes to the system design complexity.
FE, the function of probability that errors may occur, is a function of M_j.
t is the last class being triggered in SD_k.
For a Sequence Diagram k,
Sequence Diagram Correctness for Sequence Diagram k= 1 to r

$$SDC_k = \prod_{i=1}^{t} CC_i \quad (2)$$

Where
Sequence Diagram k belongs to UCm.
r is the total number of classes involved in UCm.
For a Use Case m,
Use Case Correctness UCC_m = for Use Case Diagram m= 1 to p

$$\sum_{k=1}^{r} (PSD_k)(SDC_k) \quad (3)$$

where
PSD_k is the probability that users may execute this sequence diagram k of UC_m.
p is the total number of use cases involved in OOP.
Object-Oriented Product Correctness for object-oriented metrics j = 1 to n

$$OOPC_j = \sum_{l=1}^{p} (PUC_m)(UCC_m) \quad (4)$$

where
PUC_m is the probability that Use Case m is used.
n is the total number of object-oriented metrics considered.

The structure of hierarchical aggregation of correctness is shown in Fig. 3. Note that in Definition 2, only an object-oriented metric j that contributes to the system design complexity is considered correctness evaluation and ignore the correlation. If there are more object-oriented metrics considered in the software project, with weighted average method, OOPC may be defined as the aggregation of correctness of all the metrics.OOPC =

$$\sum_{j=1}^{n} (W_j)(OOPC_j) \quad (5)$$

where
W_j is the weight of M_j.

Procedure of implementing the proposed HAM: To implement the HAM, focusing on preliminary correctness evaluation the procedure includes five steps:

1. Accomplish Class Diagrams, Sequence Diagrams and

Use Case Diagrams using UML at analysis and design stage.
2. Identify the object-oriented metrics that contribute to the system design complexity.
3. Obtain the probability function of error (FE), probabilities of PSD, PUC, and weights.
4. Calculate and aggregate all the correctness (Equations 1-5) for each design alternative.
5. Analyze the results and select the best design alternative or provide recommendations for redesign.

In step 1, constructed Class Diagrams, Sequence Diagrams and Use Case Diagrams following UML at analysis and design stage are the resources to estimate the software correctness. In Step 2, software project manager has to make decision of what object-oriented metrics that contribute to the system design complexity need to be included in the measurement. How does the project manager choose the metrics? There are a number of papers proposed on object-oriented class design metrics (M), For instance, manager can use GQM (Fenton and Pfleeger, 1996) to select the suitable metrics. This paper adopts a metrics suit by Chidamer and Kemerer (1994)and applies WMC, DIT and RFC for demonstration because they directly relate to class complexity. In Step 3 we obtain the required data FE(M), PSD, PUC and W. Regarding the probability function of error FE(M), the appropriate empirical curve may be obtained by experiments and curve fitting. To simplify, this paper assumes a probability function of error below for illustration (Fig. 4).

$$FE(M) = e^{\lambda(M - M_{\max})} \quad 0 \le M \le M_{\max} \quad , \quad (6)$$

In Equation 6, M_{max} denotes the assumed maximum values of M based on the experiences of the developers involved in the software project being considered, and they have to be preprocessed first such that they increase in the same direction. To use this function, we must set up the proficient boundary (PB) of class metrics (M) to derive the value of λ. The PB of a project team is to mark the proficiency level in the downstream development processes within a specified tolerance of error. PSD and PUC means the probabilities that sequence diagrams and use cases will be operated in practice, respectively. These probabilities may be estimated according to prior statistics, experienced users, and system analysts and designers. However, the total of PSD for each UC and the total of PUC for each OOP must equal 1. Step 4 calculate all the correctness required for analysis in the last step. The final OOP correctness will be an early indication of how a system can function correctly. Since at this stage, only system analysis and design has been completed, the preliminary evaluation of the expected software quality may provide valuable references for the decision maker especially when the system is a highly risky and costly project.

An Illustrative Example: The HAM has been demonstrated on two alternative designs of the same software project for supporting the project decision making early in the software development process. In the examples, we compared two design proposals of a simple sales information system in terms of software correctness. Here we follow the implementing procedure mentioned in Section 3.2 to describe the illustrative example.

1. Obtain the use case diagrams, class diagrams and sequence diagrams of the two design alternatives that can provide information for later measurement. In Fig. 5, we show the same Use Case Diagram for the two design alternatives. Fig. 6 and 7 are the Class Diagrams for the two design alternatives. Fig. 8, is the Order Sequence Diagrams for design alternatives 1. Other sequence diagrams for alternative 1 and those for alternative 2 are omitted.
2. The selected metrics in the example include WMC, DIT and RFC (Chidamber and Kemerer, 1994). (The definitions of the metrics please refer to Section 2.3)
3. First calculate the M's from class diagrams, the results of design alternatives are shown in Table 1. Secondly define the EF's. The ranges of M's are obtained based on the experiences of the software developers involved in the analysis and design process as WMC(0,25), DIT(0,15), and RFC(0,50) and PB are 10, 5, 25, respectively. Thirdly, we according to users' operational experiences and system requirements define PUC's as order 35%, payment 35%, Customer data 15%, and product data 15%. PSD's are shown in Table 2. Fourthly, define the weight (Wj's) for WMC, DIT and RFC which are 0.4, 0.25 and 0.35, respectively in this example.
4. Calculate all data using equations in Definition 2. The results of CC, SDC, UCC, and OOPC are shown in Tables 3 and 6, respectively. At last, the correctness of the two design alternatives 1 and 2 are aggregated and the results are 0.83334 and 0.80163, respectively.
5. After comparing the results of the two design alternatives, it is found that design alternative 1 is better than 2. Two results are close possibly due to the simple illustrative example. As project size goes larger the differences between two alternatives may become significant. Review the original data of system analysis and design of the two alternatives. It is found that alternative 2 uses more complex classes "order" and "payment" to achieve the same function. However, class inheritance exiting in the class diagram of alternative 1 has resulted in the increase of design complexity. HAM final results indicate that alternative 1 slightly superior than alternative 2 with better software correctness. In conclusion, the HAM suggests design alternative 1 which can avoid unnecessary risk for our project.

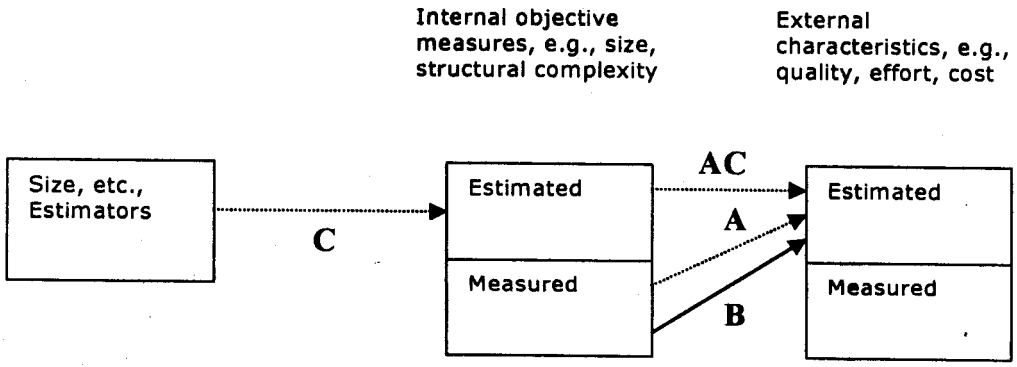

Fig. 1: The Relation of Internal and External Characteristics(Source: Henderson-sellers, 1996)

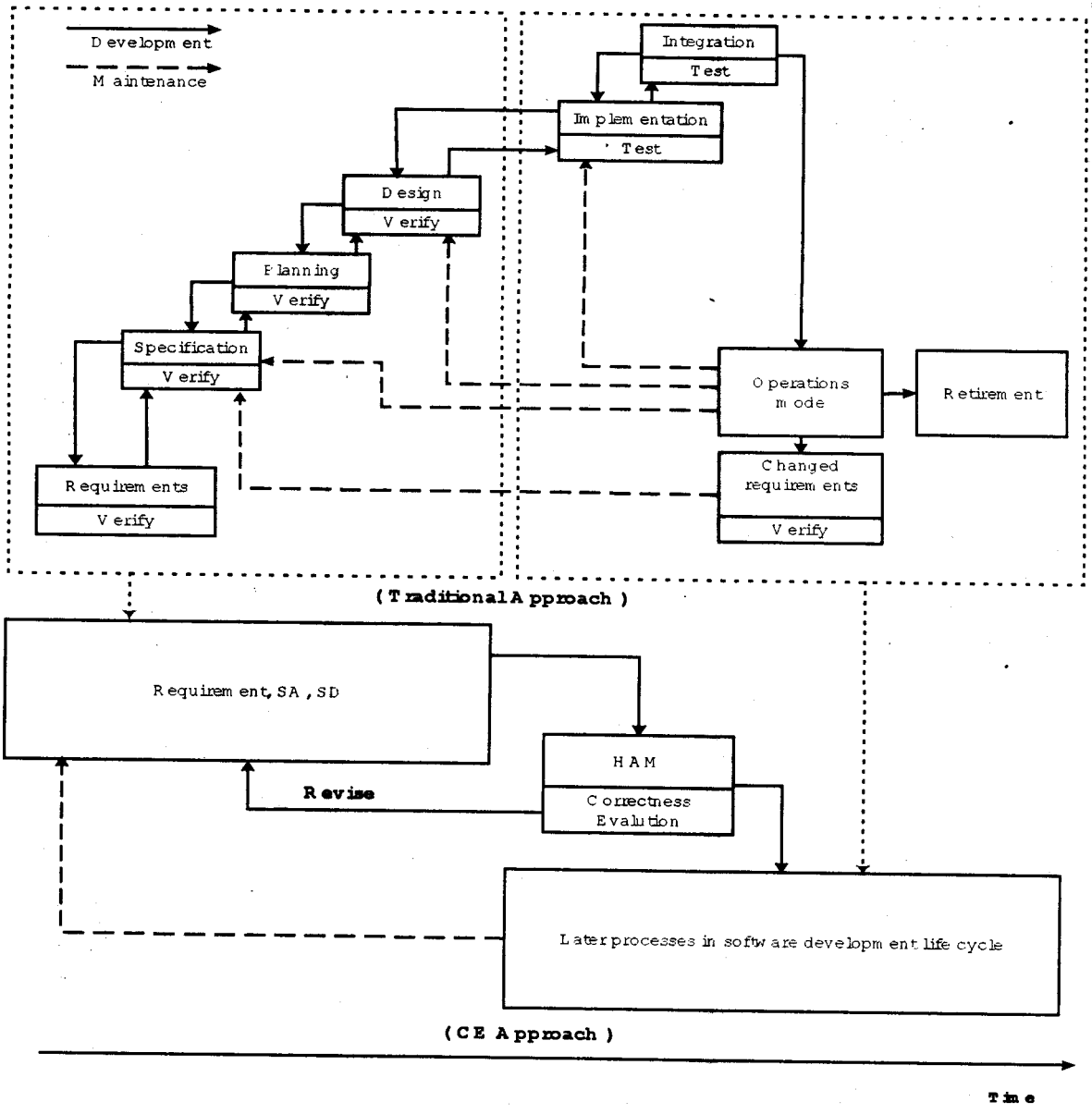

Fig. 2: The Differences Between Traditional and CE Approaches to Software Development Life Cycle

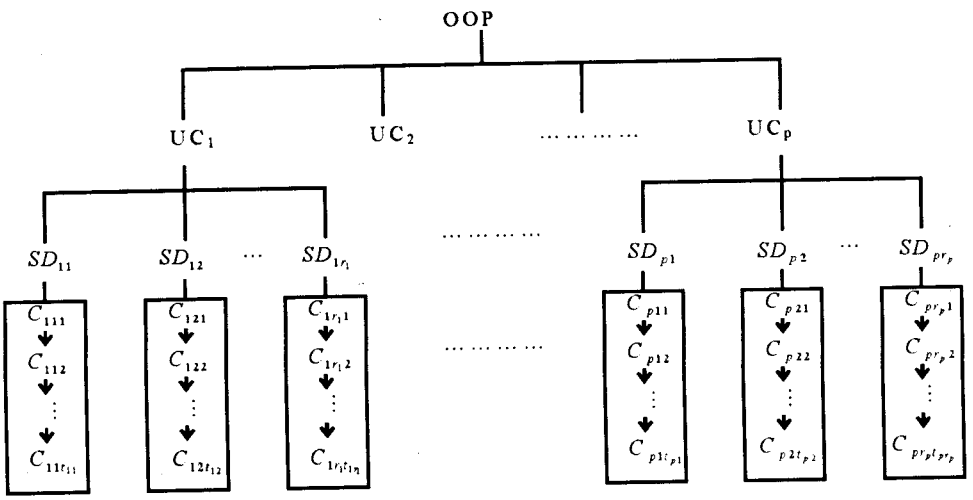

Fig. 3: The Hierarchical Aggregation Model for Evaluating Correctness

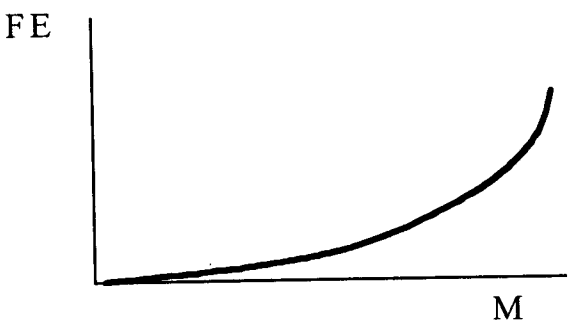

Fig. 4: Probability Function of Error Vs. Metrics

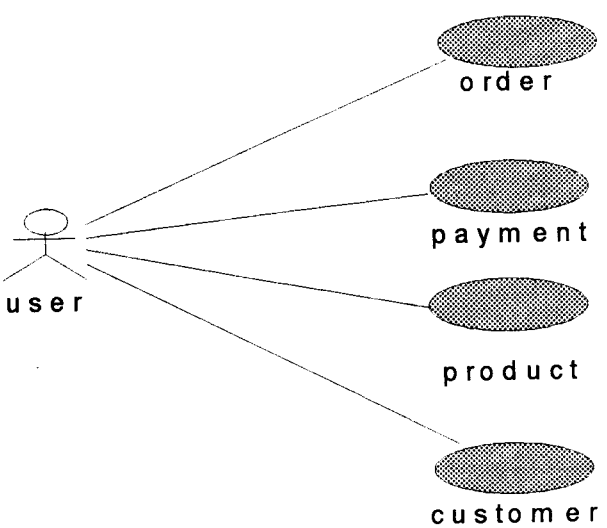

Fig. 5: Use Case Diagram for the Two Design Alternatives

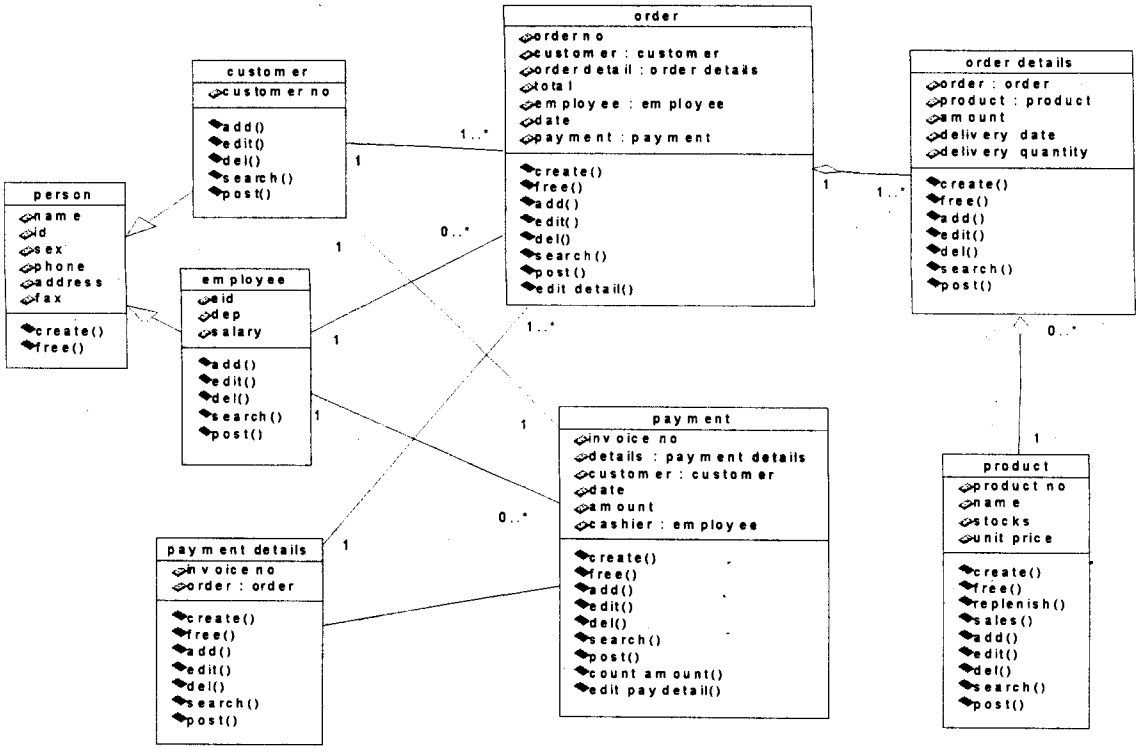

Fig. 6: Class Diagram For Design Alternative 1

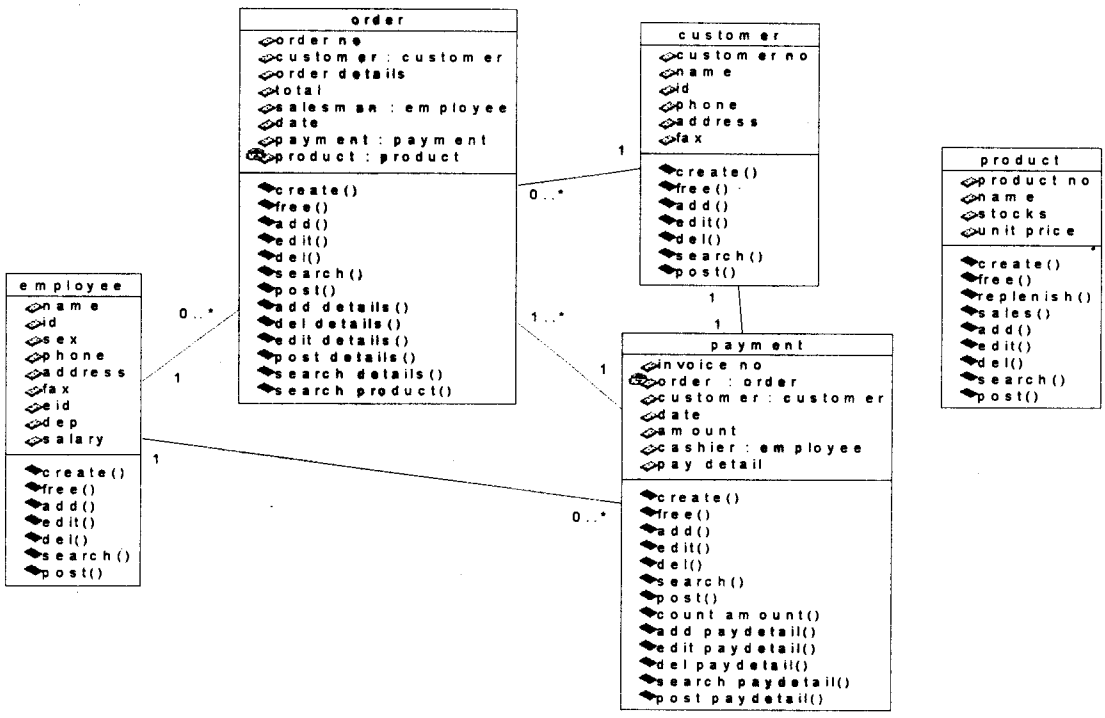

Fig. 7: Class Diagram For Design Alternative 2

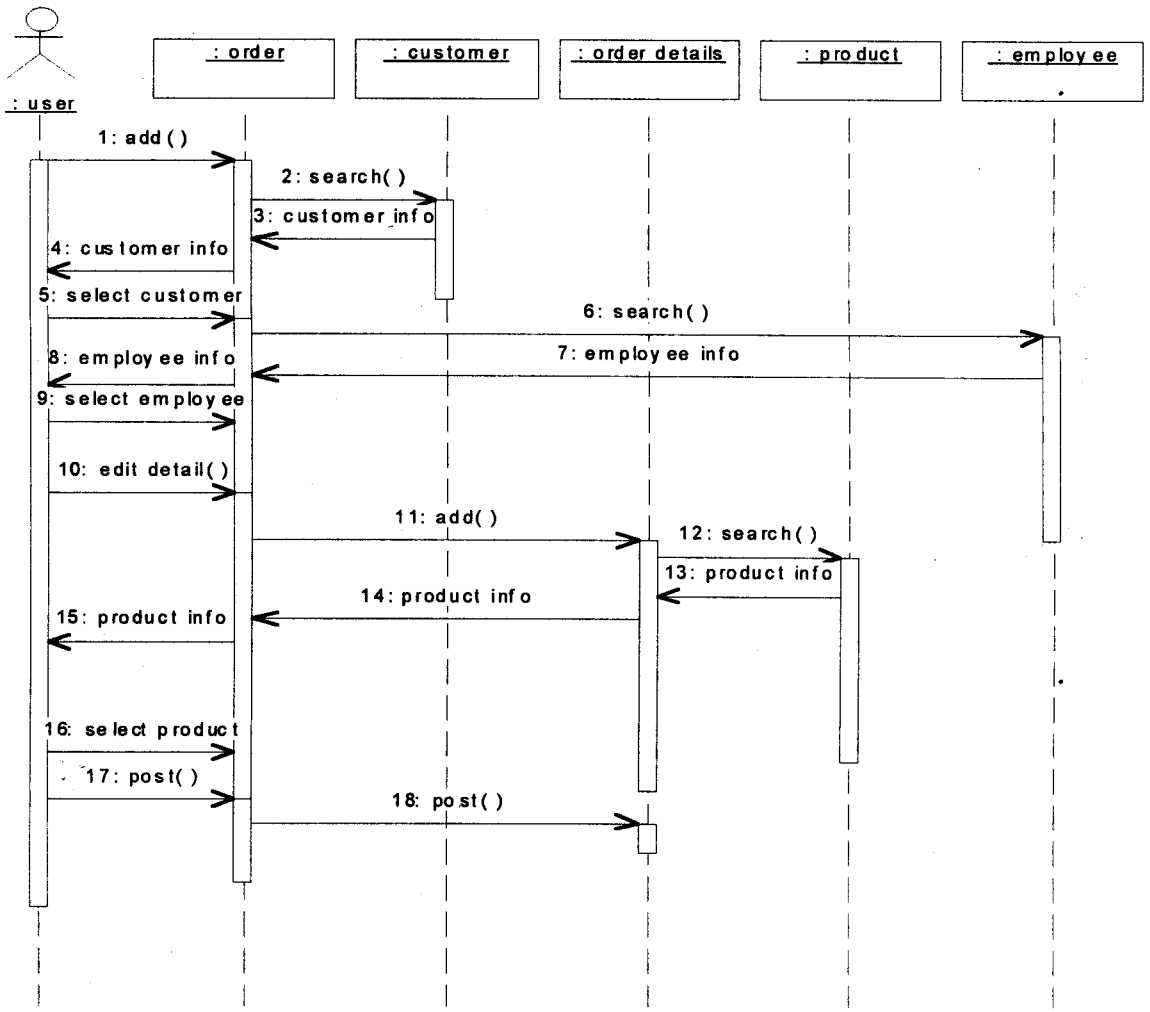

Fig. 8: Order Sequence Diagram For Design Alternative 1

Table 1: The M's of the Two Design Alternatives

	Alternative 1			Alternative 2		
	WMC	DIT	RFC	WMC	DIT	RFC
customer	5	1	5	7	0	7
order	8	0	15	13	0	16
pay	9	0	17	13	0	16
Order detail	7	0	8	0	0	0
product	9	0	9	9	0	9
employee	5	1	5	7	0	7
pay detail	7	0	8	0	0	0
person	2	0	2	0	0	0

Table 2: The PSD's of the Two Design Alternatives

UC	PSD
Order	order40%□edit order25%□del 10%□search25%
Payment	pay 70%□edit 5%□del 5%□search20%
Customer data	add 40%□edit 20%□del 10%□search30%
Product data	add 40%□edit 30%□del 10%□search10%

Table 3: The CC's of the Two Design Alternatives

	Alternative 1			Alternative 2		
	WMC	DIT	RFC	WMC	DIT	RFC
customer	0.953584	0.960189	0.984151	0.936904	1.000000	0.980945
order	0.926435	1.000000	0.960189	0.841511	1.000000	0.956348
payment	0.914230	1.000000	0.952136	0.841511	1.000000	0.956348
Order detail	0.936904	1.000000	0.979107	1.000000	1.000000	1.000000
product	0.914230	1.000000	0.977091	0.914230	1.000000	0.977091
employee	0.953584	0.960189	0.984151	0.936904	1.000000	0.980945
payment detail	0.936904	1.000000	0.979107	1.000000	1.000000	1.000000
person	0.970713	1.000000	0.987977	1.000000	1.000000	1.000000

Table 4: The SDC's of the Two Design Alternatives

	WMC	DIT	RFC	WMC	DIT	RFC
Order-order	0.580243	0.921963	0.803137	0.338645	1.000000	0.752152
Order-edit	0.638105	1.000000	0.829213	0.458452	1.000000	0.817334
Order-del	0.867982	1.000000	0.940128	0.708140	1.000000	0.914602
Order-search	0.867982	1.000000	0.940128	0.841511	1.000000	0.956348
Payment-payment	0.516591	0.921963	0.732721	0.370415	1.000000	0.769787
Payment-edit	0.679700	1.000000	0.834480	0.421986	1.000000	0.799983
Payment-del	0.856546	1.000000	0.932244	0.708140	1.000000	0.914602
Payment-search	0.856546	1.000000	0.932244	0.841511	1.000000	0.956348
Product-add	0.835817	1.000000	0.954707	0.835817	1.000000	0.954707
Product-edit	0.835817	1.000000	0.954707	0.835817	1.000000	0.954707
Product-del	0.914230	1.000000	0.977091	0.914230	1.000000	0.977091
Product-search	0.914230	1.000000	0.977091	0.914230	1.000000	0.977091
Customer-add	0.909323	0.921963	0.968553	0.877790	1.000000	0.962254
Customer-edit	0.909323	0.921963	0.968553	0.877790	1.000000	0.962254
Customer-del	0.953584	0.960189	0.984151	0.936904	1.000000	0.980945
Customer-search	0.953584	0.960189	0.984151	0.936904	1.000000	0.980945

Table 5: The UCC's of the Two Design Alternatives

	Alternative 1			Alternative 2		
	WMC	DIT	RFC	WMC	DIT	RFC
Order	0.695417	0.968785	0.857603	0.531263	1.000000	0.835742
Payment	0.609735	0.945374	0.787690	0.484099	1.000000	0.815850
Customer	0.867183	1.000000	0.963661	0.867183	1.000000	0.963661
product	0.922601	0.933431	0.973233	0.895524	1.000000	0.967861

Table 6: The $OOPC_i$'s of the Two Design Alternatives

	Alternative 1	Alternative 2
WMC	0.725271	0.619783
DIT	0.959971	1.000000
RFC	0.866386	0.867785

Conclusion

Currently, object-oriented approach is a very important software development methodology for both the academia and industry. However, there are few amount of research proposed focusing on how to improve the quality of software project development. This research attempts to incorporate the CE concept into object-oriented software development. The paper proposes a Concurrent Engineering Hierarchical Aggregation Model for preliminarily Evaluating Object-Oriented Software. With this model, the object-oriented software developer can preliminarily evaluate the software quality in terms of its correctness early in the analysis and design stage.

There are three advantages of using this model. First, this model can help manager reduce the project risk, cost, and time span, and improve the product quality early in the software development life cycle. Secondly, this model facilitates the use of the standards of object-oriented modeling and design methodology, i.e., UML, that makes HAM more applicable to real software development. Thirdly, this model is very easy to implement. Since the measurement is conducted directly from the UML models of the software project, no additional work needs to be performed. Therefore, the HAM can potentially be imbedded in software engineering tools that can support decisions such as the selection of outsourcing team during the software development process. This is significantly important due to the rapid growth of acquirement of highly costly and risky software development. However, the proposed HAM functions fundamentally based on software characteristics and their influences across the software development life cycle. A successful software development consequently lies in these important characteristics which result in ease of development, ease of maintenance, reliability, and use friendliness, etc. In conclusion, this research proposes a new CE preliminary evaluation model for which can help improve the software quality early in the analysis and design stage.

References

Booch, G., 1995. object Solutions, Redwood city, CA: Addison-Wesley.

Booch, G., 1994. Object-Oriented analysis and design with application, 2nd, Redwood city, CA: The Benjamin/Cummings.

Briand, L. C., S. Morasca and V. R. Basili, 1996. "Property-Based Software Engineering Measurement," IEEE Transactions on Software Engineering, 22: 68-86.

Bunge, M., 1977. Treatise on Basic Philosophy: Ontology I: The Furniture of the World. Boston: Riedel.

Bunge, M., 1979. Treatise on Basic Philosophy: Ontology II: The Furniture of the World. Boston: Riedel.

Card, D. N. and R. L. Glass, 1990. Measuring Software Design Quality, Englewood Cliffs, NJ: Prentice Hall.

Chidamber, S. R., D. P. Darcy and C. F. Kemerer, 1998. "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," IEEE Transactions on Software Engineering, 24: 629-639.

Chidamber, S. R. and C. F. Kemerer, 1994. "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, 20: 476-493.

Chidamber, S. R. and C. F. Kemerer, 1991. "Towards a Metrics Suite for Object Oriented Design," OOPSLA, 197-211.

DeMarco, T., 1987. Controlling Software projects, New York, Dorset House.

Eriksson, H. E. and M. Penker, 1998. UML Toolkit, John Wiley.

Fenton, N. E., 1991. Software Metrics: A Rigorous Approach, London, Chapman and Hall.

Fenton, N. E. and S. L. Pfleeger, 1996. Software Metrics, ITP.

Gonzalez, R. R., 1995. "A Unified Metric of Software Complexity: Measuring Productivity, Quality, and Value," J. of System Software, 29:17-37.

Harrison, R., S. J. Counsell and R. V. Nithi, 1998., "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," IEEE Transactions on Software Engineering, 24: 491-496.

Henderson-Sellers, B., 1996. Object-Oriented Metrics Measures of Complex, Prentice Hall PTR.

Henry, S. and D. Kafura, 1981, "Software Structure Metrics Based on Information Flow," IEEE Transactions on Software Engineering, 7: 510-518.

Jacobson, I., M. Christerson, P. Jonsson and G. overgaard, 1992. Object Oriented Software Engineering, Addison-Wesley.

Li, W., 1998. "Another Metric Suite for Object-Oriented Programming," The J. of Systems and Software, 44:155-162.

Medhat, S. S. and J. L. Rook, 1997. "Concurrent Engineering-Processes and Techniques for the Agile Manufacturing Enterprise," 5th international Conference on Factory 2000, 2.

O'Grady, P. and J. S. Oh., 1991. "A Review of Approaches to Design for Assembly," Concurrent Engineering, 1: 5-11.

Quatrani, T., 1998. Visual Modeling With Rational Rose and UML, Addison Wesley.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, 1991. Object Oriented Modeling and Design, Prentice-Hall, INC.

Rumbaugh, J., 1997. "Modelling & Design: Models through the Development Process," NewYork: SIGS, J. of Object-Oriented Programming.

Swink, M. L., 1998. "A tutorial on implementing concurrent engineering in new product development programs," J. of Operation Management, 16:103-116.

Winner, RI. 1988. "Ther Role of Concurrent Engineering in Weapons Systems Acquisition," IDA Report R-338, Institute of Defence Analysis, Alexandria, Virginia, USA.