The Collaborative Process: How Do We Deploy User Requirements to the Design of Component Models?

In, Joon-Hwan°, Lim, Joa-Sang
ComponentBasis Co., Ltd., Sangmyung University
E-mail : junani12@componentbasis.com, jslim@smu.ac.kr

Abstract

Since their first inception a few decades ago, software components have received much attention mainly due to their alleged benefits of quality and productivity improvement. Despite this, it is yet to be agreed upon what and how components should be designed. This paper aimed to bridge the gap by proposing a collaborative process where the voice of the customer is captured and documented by employing the event and entity models. These requirement elements (WHAT) are cross-tabulated in three relation matrices in accordance with the weights provided by the business users. The requirements are fed into the algorithm invented by the authors to optimize the component grouping (HOW). This collaborative process has been successfully validated at an enterprise wide software development project. The process was effective to help the users more actively involved in the design of the system and made the whole process faster and more adaptive to the changes.

1. Introduction

Software crisis, first coined in 1968 at the NATO conference, has long prompted any silver bullet to remedy the deep-rooted dilemma of schedule and cost overruns and low quality products. Although there have been remarkable advancements in project management and related technologies over the last decades, they do not appear to pay off sufficiently up to the expectation [1]. As some leading vendors have marketed their products (e.g., Sun J2EE, Microsoft .NET) and object-oriented technologies become more robust and stabilized, software component has recently attracted much attention and been favored by the practitioners mainly due to its claimed improvement in quality and productivity [1][2][3]. Despite such penetration of component technology into the market, however, there seems little consensus as to what software components are and how effectively components should be designed and developed in order to fully take advantage of its alleged benefits. We wish to address these missing gaps by raising two questions. Firstly, how do we get the users involved more actively in the process of collecting their requirements? Software project often fails to deliver what has promised to deliver and most of the failures are attributable to user involvement [2][3]. Some of the problems may be due to the fact that the user requirements are often vague and spoken in business jargons, which could not easily get across to the system designer. More importantly, the process to deploy the user requirements to the system is hardly open to the

users and left to the hands of system designers as it is claimed to be technical [4]. This will certainly bring in more frequent changes to the system that would worsen the whole software development process. Thus this leads us to the second question as to how we iteratively reflect the changes of user requirement in the design of component models. Given this, this paper aims to propose a semi-automatic collaborative process to capture business requirements with the users more actively involved and deploy them seamlessly to software components. This paper shall start by reviewing prior literature regarding user requirements and component identification. Then we shall present the collaborative process with an algorithmic framework to be validated in this paper.

2. Related Literature

User requirements are often refined through an iterative process and documented as a set of scenarios for the component based development (CBD) (e.g., use cases) [5]. Whereas this process is relatively well understood in the body of 'requirement engineering' literature [5], it is still vague the way the user requirements are led to system design. Some of the suggested methods to discover business objects include either linguistic or categorical approach [7]. The former approach is to find candidate objects/classes from the nouns. The latter is to locate such semantic categories as place, roles and containers among others. Despite such practical guidance, Kaindl [4] argued that it is still difficult to transit the classes discovered in the requirement analysis to the ones to be used in the design phase. Furthermore there exist only a few studies on grouping these fine-grained classes into coarse components. IEEE suggests one of the crucial activities in software design is to decompose the whole system as long cherished by the divide and conquer principle. In this regard, QFD (Quality Function Deployment) may be useful to develop a system to meet business requirements and translate them to design requirements [8]. The QFD was invented in the 1960s and since then has been used in various industries such as production, manufacturing and software development [6]. The QFD have been reported to be valuable us in managing conflicting views of stakeholders in software development [7][8][9]. Kim et al. [10] proposed the formal approach to decompose the HOQ (House of Quality) of QFD into smaller problems combined with the multi-attribute value theory and formulated the quadratic model that minimizes the overall dissatisfaction level due to the out-of-group entries as a result of the grouping. This has been reflected in the research in that the legacy programs are parsed and analyzed to extract some independent modules of source code. For example, Etzkorn et al. [11] calculated some meaningful metrics from the legacy object-oriented source code to automatically identify components. Whereas this line of research relied on reverse engineering from the source code, some studies shifted the focus onto the requirement artifacts from which to identify software components in a forward way. Jain et al. [12] proposed a business component identification method where the business objects were related to each other and their static and dynamic relationships were fed into the clustering algorithm and semi automatic heuristics. Lee et al. [13] also used the analysis model and the functional use cases and classes were cross-tabulated with each other to extract a set of reusable components. Whereas Lee et al. [13] emphasized the coupling and cohesion of use cases and classes considered independently, Jang et al. [14] challenged to relate use cases with business objects using the affinity analysis technique. They sorted use cases in a logical

affinity sequence and related them with a set of classes in a matrix. Affinity analyses were performed for any intersections between use cases and classes and the type of transactions (e.g., Create, Read, Update & Delete) was analyzed. Then most associated group of classes was to be identified as a component. Finally, Albani et al. [15] proposed a procedural algorithm based on the functional decomposition diagram and the data model, which associated relevant tasks and information objects in consideration of their relationships. In contrast to the earlier studies where the legacy source code is reverse engineered, the studies using the analysis model certainly provide a vehicle to identify components at the earlier phase of software development life cycle. However, these studies do not appear to offer sufficient guidelines for the practitioners to cope with detailed requirements of the larger software development projects. Indeed, it is often experienced that thousands of functional requirements and hundreds of entities are to be explored for component modeling. This study offers the collaborative process where the users are allowed to set up the policy (preferences) with regards to system design and associate the functional requirements with design requirements to identify software components.

3. The Collaborative Process

The collaborative process consists of three phases – requirement analysis, overall design and detailed design as seen in Fig. 1. Of these phases, the second overall design highlights the nature of the collaborative process where the user and the system views are met and coordinated as shaded in Fig. 1. Thus the role of the user is not confined to the earlier phase of requirement, but extended to the later stages of system design. The first activity as suggested in the QFD literature [16] should identify the stakeholders of the systems and then capture the voice of the customer. In the object-oriented development, use case modeling is the most favored approach to document both functional and non-functional user requirements. Then logical entities are sought and decomposed in relation to the functional requirements of the use cases. This is followed by the overall design of relating both the use cases and the entities in accordance to the policy as to the types and strengths of the interrelationships and their optimality. An algorithm was invented here with a metric to evaluate the satisfaction level as the ratio of the associations included in the identified components. As the focus of the collaborative process, this is detailed in the following sections. The components and interfaces discovered in the overall design are realized in the subsequent phase of detailed design.

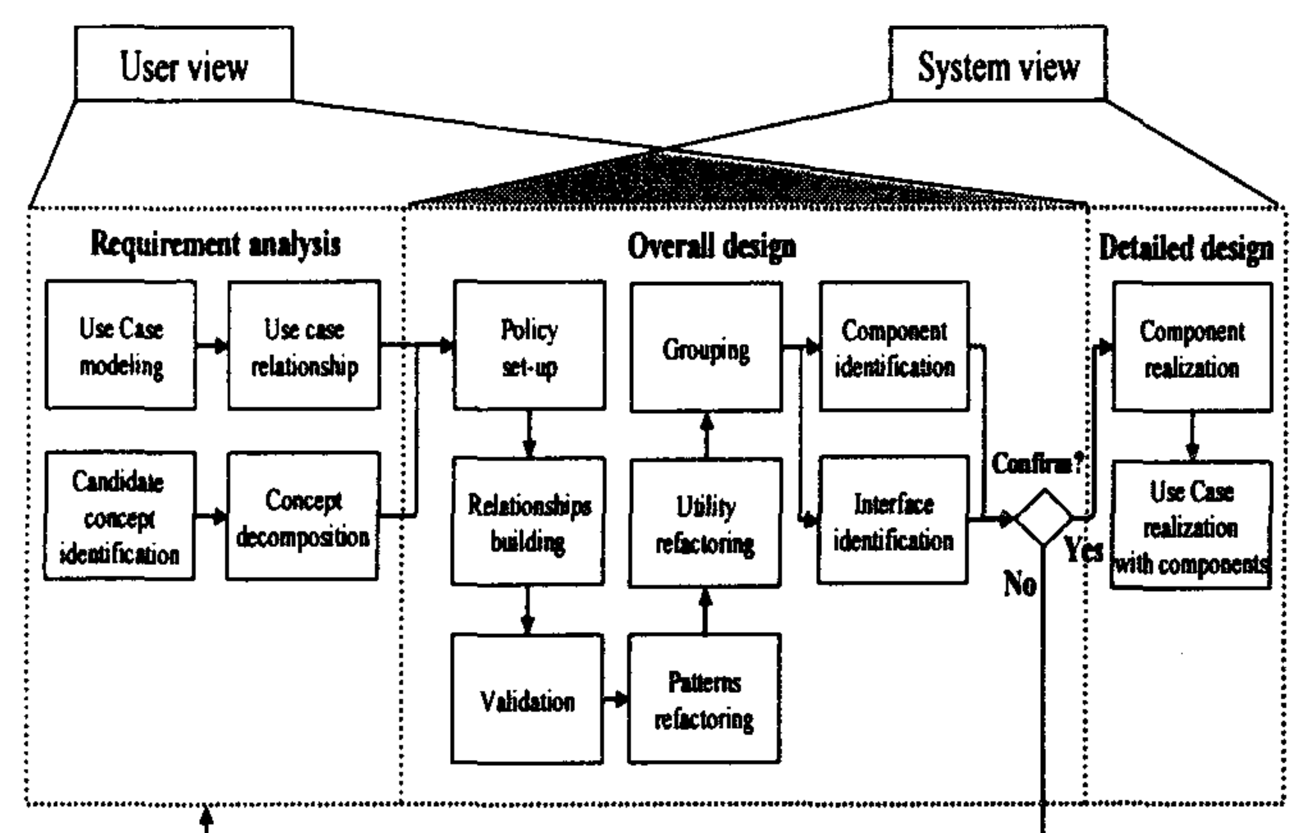

Fig. 1. The collaborative process with the user view extended to the overall design phase (shaded), which repeats until the requirements are fully deployed to the component model

3.1 Building relationships of use cases and classes

As modeled in the earlier phase, the two model elements of use cases and business objects play a critical role to construct the relation matrix. The details as to how to relate them are described in the policy set-up activity. We propose three possible cross-tabulations as follows:

1. The use case x use case relationship matrix: This

relates use cases with each other to find commonly used functional components and use case packages depending on the extent of correlation (CU) as in Equation (1) where the subscript represents the use case i and j respectively. Considered the most correlated are 'include,' 'precondition' and 'generalize,' being certainly stronger relations than the 'extend'. The least correlated use cases refer to those hardly used together. These strongly coupled use cases serve a locus of control to identify components.

$$CU_{ij} = \{\text{the correlation between } UC_i \text{ and } UC_j\} \quad (1)$$

2. The use case x entity relationship matrix: The second matrix is established with four different transactional types of relations (RE) such as C (create), R (read), U (update) and D (delete) (see Equation 2). Non-functional requirements may also be considered into the matrix (e.g., transaction frequency). The weight is computed as Equation (3) for all relevant use cases and entities where is the weight of transaction type x.

$$RE_{ij} = \{C, R, U, D\} \quad (2)$$

$$W(RE_{ij}) = W(C) + W(R) + W(U) + W(D) \quad (3)$$

3. The entity x entity relationship matrix: The last matrix is concerned with the relationships among entities as represented in Equation (4). The different weights may be given according to such possible relations among classes as inheritance, composition, aggregation, association and dependency.

$$CC_{ij} = \{\text{the correlation between } C_i \text{ and } C_j\} \quad (4)$$

It should be noted that the weights of three matrices be normalized. With these cross-tabulations, we may then proceed to the component graphs and the computation of edge weights with MST (Minimum Spanning Tree), which is iterated to minimize the dissatisfaction level of identified components as shall be discussed in the following stage.

3.2 Grouping of use cases and classes into components

With the relationship matrices constructed in the earlier activity, we proceed to group the use cases and the entities into components. Fig. 2 shows the algorithm, which starts with the use case graph to find a seed solution and then moves to 'grouping' to minimize the loss of relationship between use cases and entities.

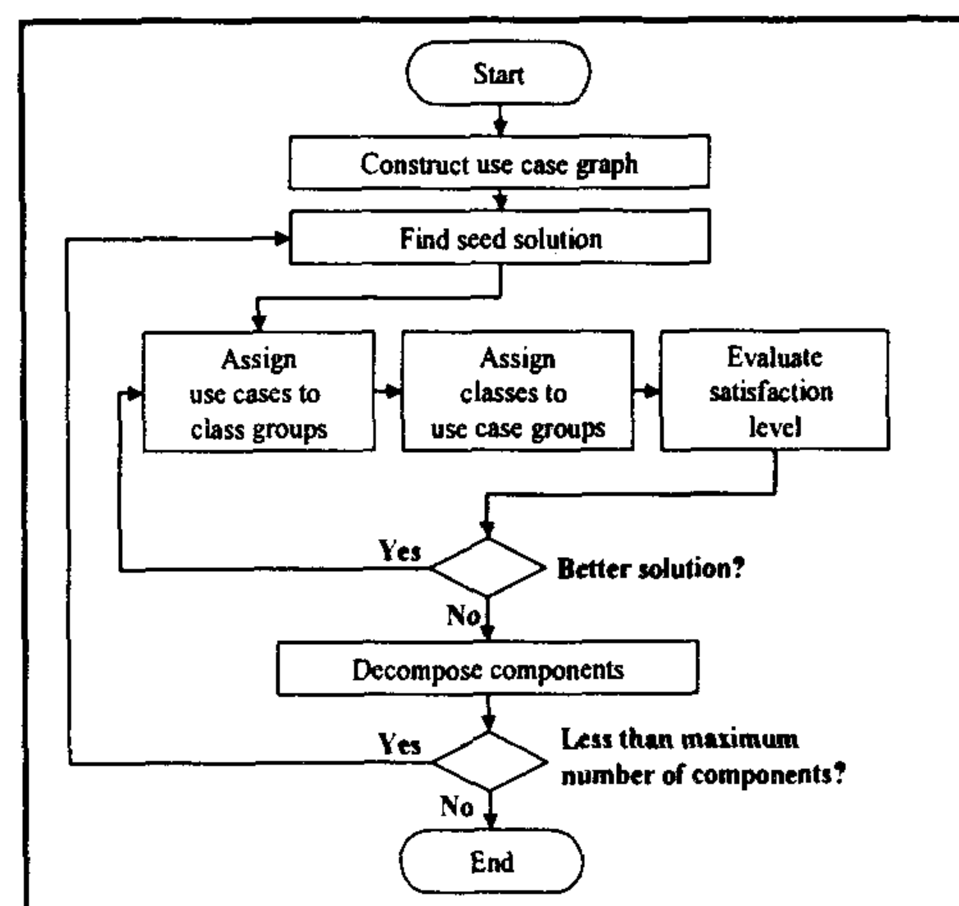

Fig. 2. The algorithm that iteratively assigns the use cases and the classes into component groups to find the best solution that minimizes penalty to lose the relationships

Firstly, the relationship matrices are visualized with graph notations. As seen in Fig. 3, the use case is denoted as the 'uc' node whereas the class, as the 'c' node. The edge represents the association between the nodes. To make the algorithm computationally efficient, the use case-class graph (the left-most of Fig. 3) is transformed to the use case graph with all involved classes removed

(the middle of Fig. 3). The information loss caused from the transformation is to be recomputed in the edge weights of the use case graph. Then the cyclic nodes are transformed to a tree with the strongest nodes remained as two cyclic nodes as in the right-most of Fig.3 – that is, (1) UC1,5 and (2) UC2, 3, 4.

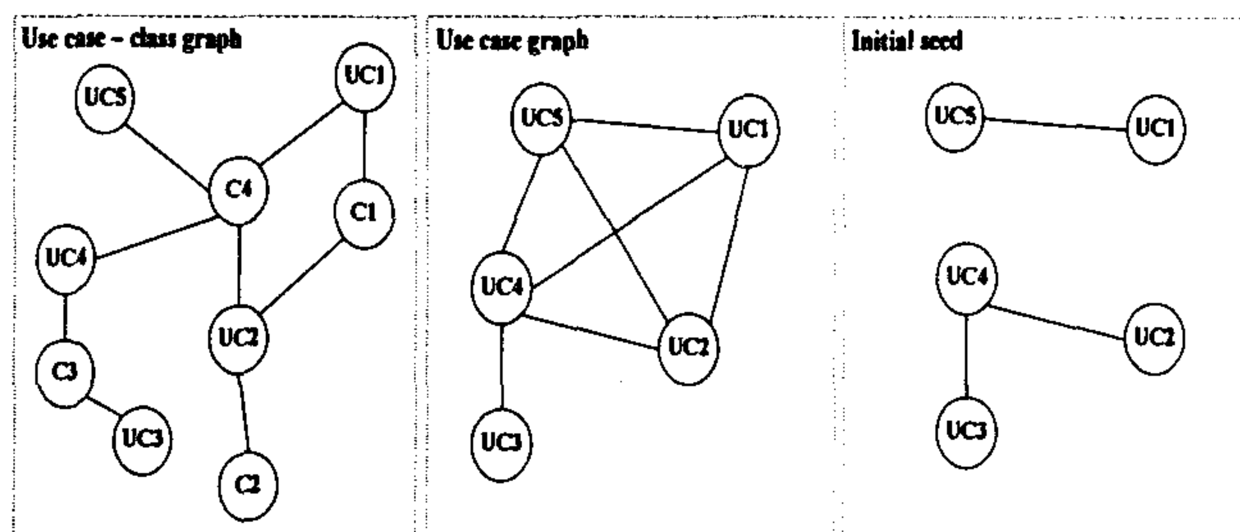

Fig. 3. Finding an initial seed solution: The use case-graph is drawn with the relationships and transformed to an initial seed with the classes removed and the edges cut with the MST rule.

As seen in Fig. 2, the next step is to find a seed solution by utilizing MST (Minimum Spanning Tree). This group shall serve as a seed container to which the relevant classes are assigned according to the dispatching rule. Then use cases are assigned to the class groups in a reverse way. This iterative process stops at a point where the dissatisfaction level does not decrease. Then, one edge is cut and the step is iterated until all components are identified as represented in the later part of Fig. 2. As discussed earlier, the edge weights are the sum of three possible relationships (i.e., use case x use case, use case x entity, entity x entity) as computed with the distance and the similarity rule. For example, suppose that a set of use cases are related and thus grouped into a package to which in turn we found any related entities. Here the use cases and the entities are considered similar (distant) and thus may well (not) be grouped together in case there exist similar (distant) relations at the intersections between use cases and entities. This is illustrated by the edge weight (EWij) as in Equation (1).

$$EW_{ij} = \frac{\sum_{\forall k \in Classes} RE_{ik} \cdot RE_{jk}}{\sum_{\forall k \in Classes} \left(RE_{ik} - RE_{jk}\right)^2} \quad (1)$$

Taken together, the total edge weight (TEW) for all three matrices is defined as Equation (2) where $w_1 + w_2 + w_3 = 1, 0 \le w_i \le 1$

$$TEW_{ij} = w_1 \cdot \frac{CU_{ij}}{\sum\sum CU_{ij}} + w_2 \cdot \frac{EW_{ij}}{\sum\sum EW_{ij}} + w_3 \frac{\sum_{k,l \in UCi,UCj} \sum CC_{kl}}{\sum\sum CC_{kl}} \quad (2)$$

Then the dispatching rule is employed to assign the classes to the use case groups and vice versa by computing the relation sum (RS) between group i and class j as in Equation (7).

$$RS_{ij} = \sum_{k \in Groupi} RE_{kj} \quad (3)$$

Firstly, this rule generates an initial feasible solution in which each group contains one class at least. The first step proceeds over the following steps.

1. Calculate the relation sum for all use case groups.
2. Count the number of assignable classes for each group.
3. Assign the class to the use case group that has the minimal number of assignable classes. Ties are broken by assigning the class that has the larger sum of relations.
4. Repeat the above steps 2 and 3 until each group contains one class.

Then we assign classes to use case groups that have the largest RE. The following steps are performed.

1. Choose the class that has the maximal relation sum.
2. Assign the class to the corresponding use case group.
3. Repeat the above steps 1 and step 2 until unassigned

class does not remain.

$$ObjValue = \frac{\sum_{(\text{all i and j}) \notin Group} RE_{ij}}{\sum \sum RE_{ij}} \quad (4)$$

The objective value (ObjValue) in Equation (4) is defined as the dissatisfaction level of identified components, which minimizes the sum of relation weights that fall outside the resulting groups. For any iteration, the objective value is calculated and this process is repeated until its outcome value does not improve any more.

4 A Field Experience with the Collaborative Process

4.1 The case details

The proposed process was run at a field for a motor sale corporation in Korea, which was locally the first to specialize in auto sales and service. The corporation experiences difficulties due to sluggish economy and surging of bad consumer credit and its sales have been steadily decreasing from $3.5 billion in 2002, $3 billion in 2003 to $2.8 billion in 2004. To launch more aggressive sales programs, the top manager decided to renovate the legacy system written in COBOL to be redesigned in UML and implemented using component tools and technologies. The company has served customers through well-designed business processes operated since its foundation in 1966, which comprises human resource, accounting, installments, sales, procurement, logistics, branch management, account receivable, marketing and used-car sales. A total of 657MM were used over the 14 month long development process of eight iterations. The estimated function points of the project were 18,476. The number of use cases and entities was 1,008 (3,805 functions) and 3,593 respectively. The following section presents the core sales business process (3,203FP, 84MM) as to how user requirements were collected, coordinated and deployed into component models with the proposed process in this paper.

4.2 Activity details for requirement analysis

Requirement analysis for the sales business was performed to identify use cases and entities to be associated in a subsequent phase. A careful analysis of the sales produced a total of 492 functions, which were then utilized as input to use cases. In this case, we limited the size of use cases under 50 FPs and a total of 113 use cases were identified. Remember another prerequisite to the algorithm was a set of logical entities. A total of 81 logical entities were identified from candidate nouns dug out of various sources such as use case descriptions, business glossary and interviews with business users. The candidate nouns and entities were refined and further decomposed depending upon if they had any relevant attributes. The requirement analysis was often revisited depending upon additional analysis we had with business users in the overall design phase (see the reverse loop of Fig. 1). Thus, any further analysis with relationships matrices was conducted with 113 use cases and 81 entities.

4.3 Activity details for building relationships between use cases and classes

Once use cases and entities were prepared, these ingredients were cross-tabulated to generate the relation matrices. The first matrix related all identified use cases each other to group them into use case packages depending upon the strength of their relations. For example, the stereotype 'include' relation was regarded

as stronger than the 'extend' one. Another relation matrix was concerned with any possible relationships among entities such as aggregation, composition and inheritance. The last relation matrix of primary interest to this study dealt with the relationships between use cases and entities. This matrix recorded all weights for the functional requirements of intersections between use case and entities (see Fig. 4). The matrix was examined if there existed any black holes (i.e., an entity never used by any use cases) and miracles (a use case that did not use any entities). Also read-only entities were examined if they were created in other domains (faulty otherwise). The parameters were set as follows:

• The number of components to be identified was set between 10 and 20.

• Four important functional types considered in this study included create (C), read (R), update (U) and delete (D). The weights given to the transaction types were set as 2 for R and 8 for C, U and D.

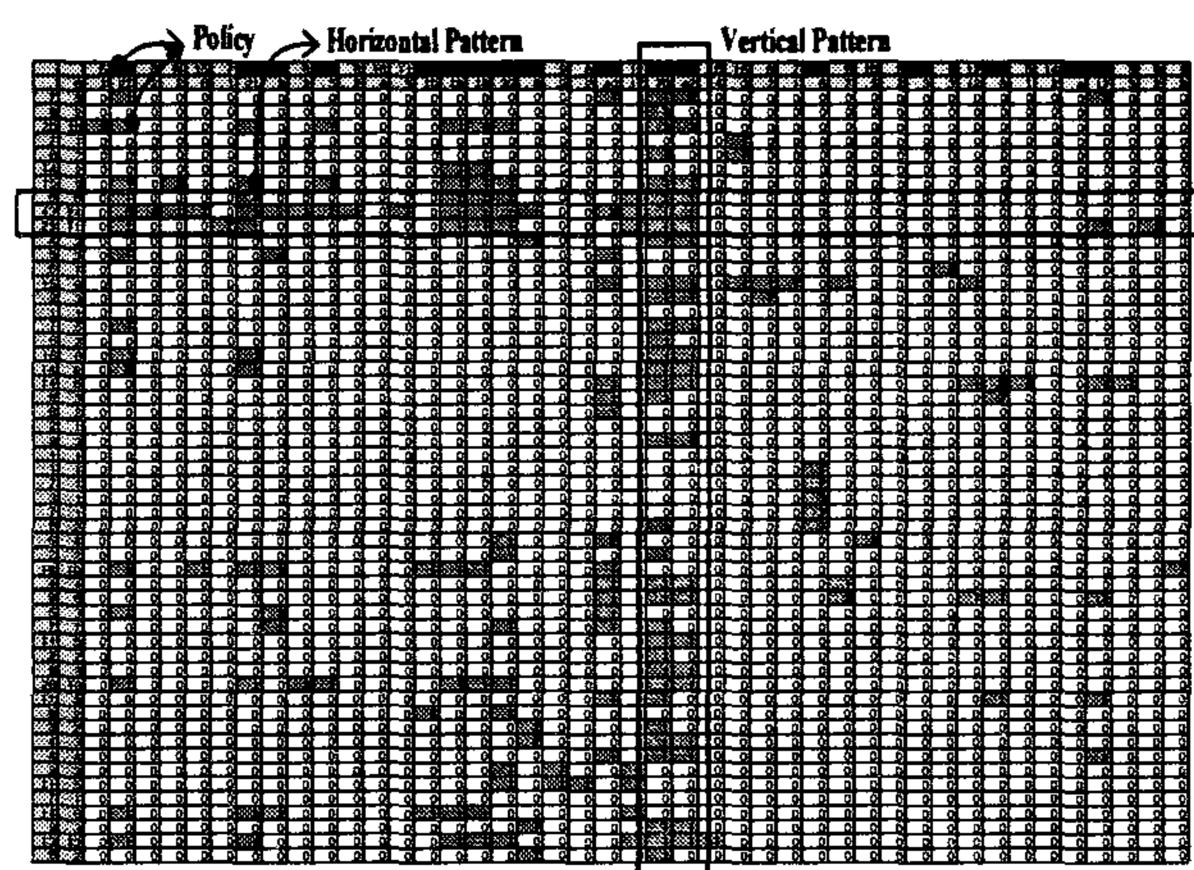

Fig. 4. Relationships between use cases (row) & classes (column). The scores were given according to the policy. Partial data are presented here with 54 use cases and 43 entities

The relations of use cases to classes were well scattered as seen in Fig. 4 and it seemed not easy to identify any significant components. The horizontal pattern occurred for the relationship of a use case with many entity classes. This pattern was often observed for any batch processing which required access to many relevant entities. On the other hand, the vertical pattern referred to the other case where an entity class was used heavily by many use cases as often witnessed between the base use case and the included use case.

4.4 Activity details for grouping of use cases and classes into components

The data as seen in Fig. 4 were fed to the algorithm. This resulted in a total of 13 components (i.e., gray rectangles) as seen in Fig. 5. Remember the objective was to minimize the penalty that would have on the component group by removing any relationships out of the group. Thus any further searching for any meaningful components would be abandoned due to the dissatisfaction level minimized with 13 components as explained earlier. The dissatisfaction level was 30.57% in this case.

Fig. 5 shows that the biggest component contained 13 use cases and six classes. There also existed two of fine-grained components with one to one relationship between use case and class. We had a number of 'semantic' sessions with the business users and developers if the grouped components were meaningful in their business operations. The semantic session proceeded smoothly with the relation matrices as there recorded all penalties in numbers and all stakeholders could easily understand what to lose by getting in and out any classes. The results of the semantic process are displayed in thick white rectangles in Fig. 5 and any loss of meaningful relationships were insignificant in comparison to the algorithmic result (i.e., the gray rectangle).

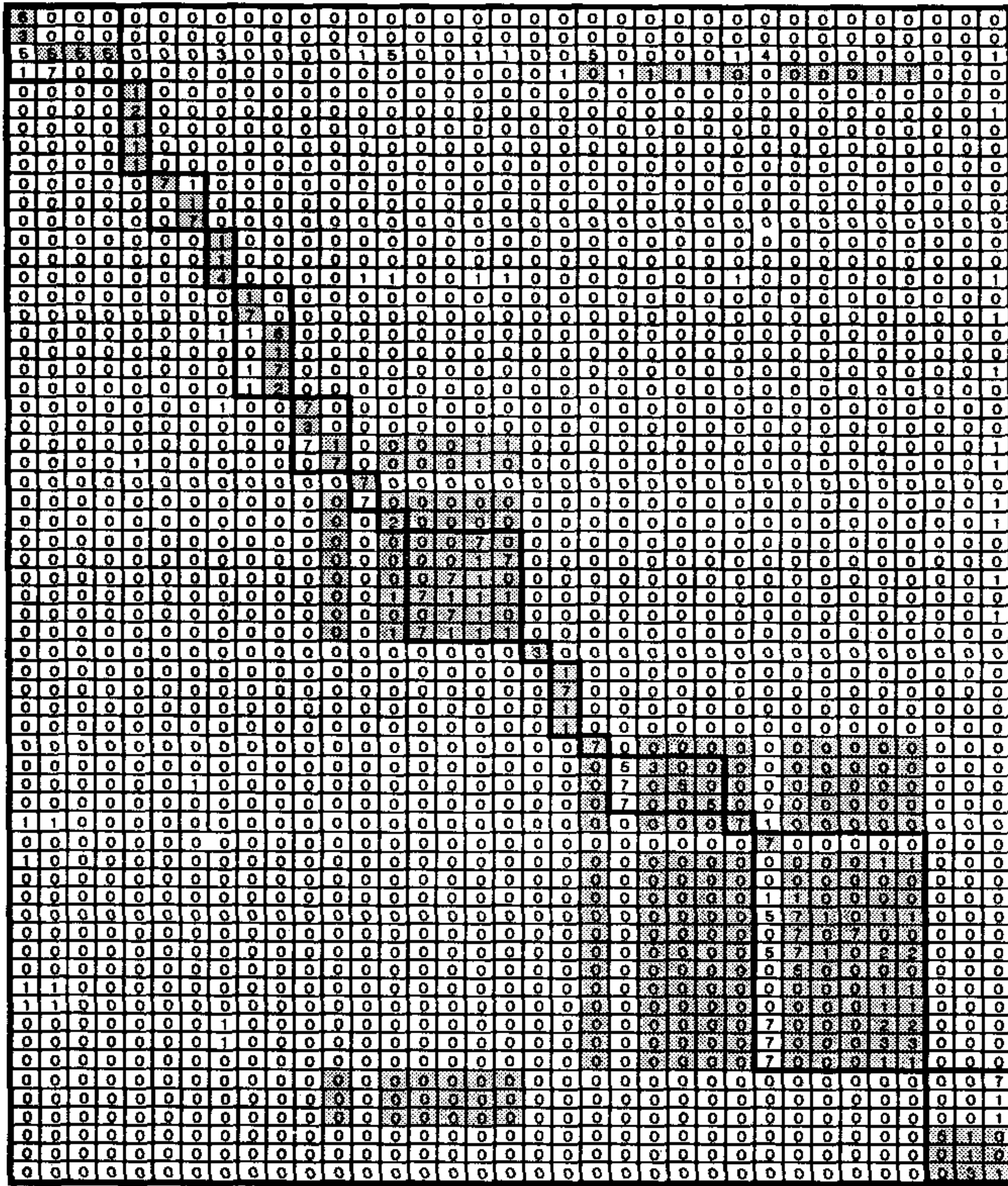

Fig 5. Running grouping algorithm with 113 use cases & 81 entity classes. The gray rectangle ( ) denotes the automatic approach whereas the thick white ( ), the semantic process

4.5 An exemplary component model for the detailed design

The diagrams that could be produced in reference to the grouping result may include component diagrams, assembly diagrams, class diagrams and sequence diagrams. Fig. 6 shows two of these UML diagrams. The identified components were sufficiently and correctly specified in the detail design phase so that the use cases of not only the sales business but also other domains could be realized in reference to them. As seen in Fig. 6, the provided interfaces were drawn from the functional requirements of a component. On the other hand, the required interfaces were functionalities of other components. Presented in the right-side class diagram of Fig. 6 were the details of a class inside the component.

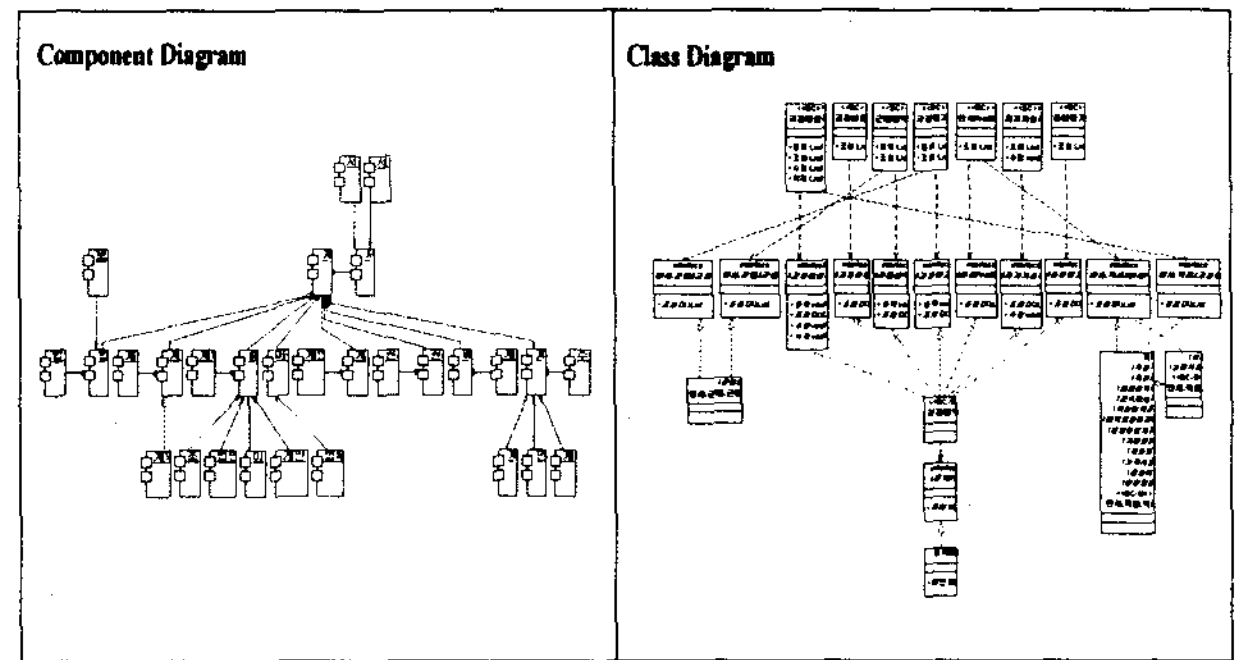

Fig. 6. Component & class diagrams. The left component diagram shows all provided and required interfaces. The right one detailed classes for a component with functional operations

5. Discussion

Although CBD has been widespread in practice, little is known as to the process user requirements could be captured into reusable software components. This task, however, requires much cognitive effort to take into account all possible interactions among classes and components. Firstly, for large and complex systems, there are often more than hundreds of classes and functional requirements to deal with and thorough examination of such data would be simply impossible and error prone. Secondly, the iterative and incremental approach as often adopted in recent CBD practice, changes in user requirements also force modification to components identified earlier. Thus more systematic approach is required to reduce error-proneness in component identification. As proven to be useful in managing functional and non-functional requirements of the users in various industries, the QFD process was employed in this paper to identify software components. Three potential relationship matrices were cross tabulated with use cases and entities and an algorithm was developed by the author to semi-automatically deploy user requirements into software components. The algorithm was validated for the first case and resulted in an acceptable solution with appropriate degree of

granularity and dissatisfaction. We further validated the algorithm with the case that contained a horizontal pattern and thus would possibly lead to a 'king-kong' component. Expectedly we observed a coarsely grained component. Any decoupling approach would be valued to decompose such big sized components. The algorithm was designed to allow to easily plug any expert opinion into play in the form of weights to be given for any intersections among those elements of use cases and classes that are so crucial in object-oriented analysis and design. The QFD based algorithm can be run repeatedly until the solution satisfies the expected quality of reusable components. Considering that enterprise software development often requires analysis of hundreds of use cases and classes and its manual handing is a daunting task, further research is required with more practical cases to validate and improve the algorithm. It is also useful to study the impact of parameters and the way to easily incorporate expert opinion into the systems.

[References]

[1] Glass, R. L.. The Realities of Software Technology Payoffs. Communications of the ACM 42 (2), 74-79., 1999

[2] Terry, J., Standing, C. . The Value of User Participation in E-Commerce Systems Development. Informing Science Journal 7, 31-45., 2004

[3] Hilbert, D. M., Robbins, J. E., Redmiles, D. F.. Supporting Ongoing User Involvement in Development via Expectation-Driven Event Monitoring. Technical Report UCI-ICS-97-19, Department of Information and Computer Science, University of California, Irvine, 1 - 11., 1997

[4] Kaindl, H.. Difficulties in the transition from OO analysis to design. IEEE Software 16 (5), 94-102. , 1999

[5] Kaindl, H.. A Design Process Based on a Model Combining Scenarios with Goals and Functions. IEEE Transactions on Systems, Man , and Cybernetics 30 (5), 537-551., 2000

[6] Gu¨lcin, B., Orhan, F.. Group decision making to better respond customer needs in software development. Computers and Industrial Engineering 48, 427-441., 2005

[7] Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd Edition. Prentice-Hall. , 2002

[8] Akao, Y.. New product development and quality assurance deployment system. Standardization and Quality Control 25 (4), 243-246., 1972

[9] Kudikyala, U. K., Vaughn, R. B.. Software requirement understanding using Pathfinder networks: Discovering and evaluating mental models. The Journal of Systems and Software 74, 101-108., 2005

[10] Kim, K. J., Moskowitz, H., Shin, J. S.. Design Decomposition for Quality Function Deployment. Essays in Decision Making, Springer-Verlag, Berlin, 215-236., 1997

[11] Etzkorna, L. H., Hughes, Jr., Davisa. C. G.. Automated reusability quality analysis of OO legacy software. Information and Software Technology 43, 295-308., 2001

[12] Jain, H., Chalimeda, N., Business component identification. Enterprise Distributed Object Computing Conference, Proceedings. Fifth IEEE International, 183-187.

[13] Lee, J. K., Jung, S. J., Kim, S. D., Jang, W. H.. Component identification method with coupling and cohesion. Software Engineering Conference, APSEC. Eighth Asia-Pacific, 79-86., 2001

[14] Jang, Y. J., Kim, E. Y., Lee, K. W.. Object Oriented Component Identification Method Using the Affinity Analysis Technique. Lecture Notes in Computer

Science 2817, 317-321. , 2003

[15] Albani, A., Keiblinger, A., Turowski, K., Winnewisser, C.. Domain Based Identification and Modelling of Business Component Applications. Lecture Notes in Computer Science 2798, 30-45. , 2003

[16] Karlsson, J. Managing software requirements using quality function deployment. Software Quality Journal 6, 311-325. , 1997