Reusing UML Specifications in a Constrained Application Domain

Maurits C. Blok
Business Information Technology
University of Twente
Enschede, The Netherlands
m.c.blok@student.utwente.nl

Jacob L. Cybulski
Department of Information Systems
University of Melbourne
Parkville, Vic 3052, Australia
j.cybulski@dis.unimelb.edu.au

Abstract

This article describes a method of reusing computer software designed in UML (Unified Modelling Language) with the aid of a domain model. The method's main strength is the possibility of software reuse at the earliest stages of the development life cycle, i.e. specification of use cases and their event flows. The article provides details of the representation techniques used for storing and retrieving reusable design components to and from a large collection of UML specifications. It describes approaches to reduce the complexity of dealing with very large domain models. And it describes the method of assessing the conceptual similarity between event flows and use cases.

1: Introduction

UML (Unified Modelling Language) is the new emerging standard for the specification of object-oriented systems [8]. UML combines several different diagramming techniques, which together allow smooth transition from the most abstract description of the system behaviour, through the system architecture, down to the level of detailed design. The most important element of the UML approach is its ability to specify the interaction between the system and its users (actors) with the aid of use case diagrams (Cf. Figure 1). Each use case represents a number of possible flows of events constituting a single kind of system use [5]. The primary, alternate and exceptional flows of events are expressed with UML sequence diagrams, which show the series of events occurring between the system and its users and, on the design level, also the events triggered by the system components (Cf. Figure 2). Collaboration diagrams are used typically to summarise all event sequences into a cohesive model of class interaction. The structure of interacting system objects, their classes and relationships are also captured in the UML class diagrams (Cf. Figure 3). The behaviour of individual system objects is given in state diagrams, which in turn, can be used to generate program code. Finally, component and deployment diagrams specify the system modularity and its installation details.

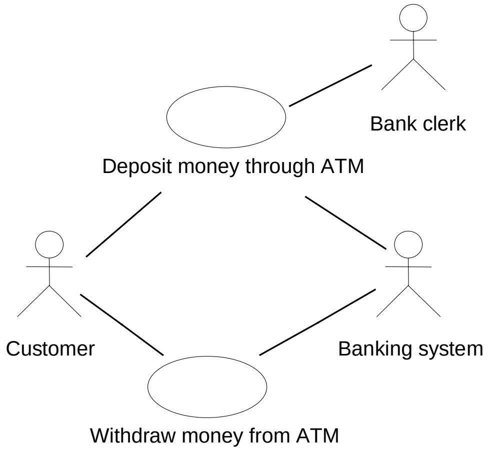


Figure 1: Example of a use case model

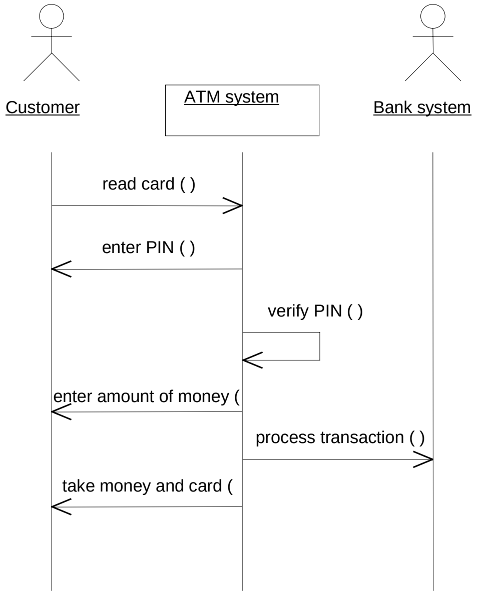


Figure 2: Sample sequence diagram 'Withdraw money'

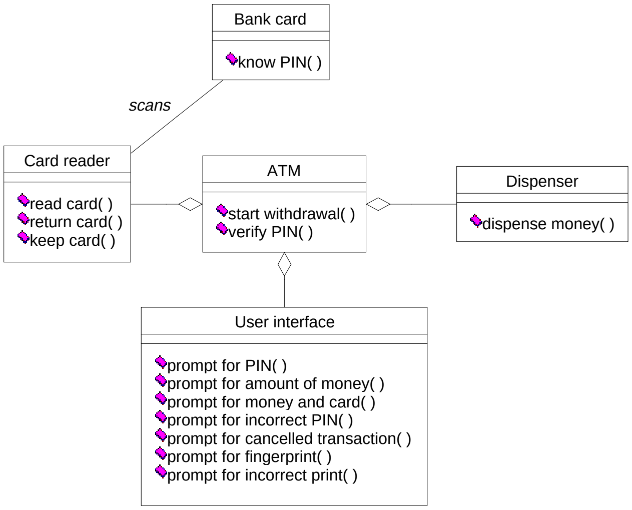

Bank card
know PIN( )
scans
Card reader
read card( )
return card( )
keep card( )
ATM
start withdrawal( )
verify PIN( )
Dispenser
dispense money( )
User interface
prompt for PIN( )
prompt for amount of money( )
prompt for money and card( )
prompt for incorrect PIN( )
prompt for cancelled transaction( )
prompt for fingerprint( )
prompt for incorrect print( )

Figure 3: Sample class diagram

To date, UML has been mainly utilised in the specification and design of the function and structure of individual software systems. Nevertheless, the technique is also applicable to the process of domain engineering, i.e. analysis, modelling and organisation of knowledge about the entire class of problems and their solutions in a systematic way [1]. Domain engineering results in a domain repository that may contain not only specification and design models, such as those produced with UML, but also the selected development work-products including requirements documents, designs decisions, data objects, source and binary code, libraries, etc. Domain models act as the libraries of ready-made solutions to typical problems in a given application domain. They help in keeping consistency in the design of previously developed and new systems. They allow a more thorough and complete specification of the system functions and data. They facilitate systematic reuse of software across the entire development life cycle, and hence, make the development of systems in a given problem area more effective and efficient.

The UML notation and some of the tools that support UML-based development, such as Rational Rose, System Architect or Together/J, can be used as the effective means of organising domain models. With the use of such tools, domain knowledge can be grouped together and set aside as UML packages. Packaged domain models would typically contain a large collection of high-level, abstract class definitions, use cases and interaction diagrams. They could also contain low-level designs that are shared between numerous system specifications and which also associate reusable program code. While code reuse is certainly an attractive option to software developers, reuse of the most general specifications, i.e. use cases and interaction diagrams, will prove to be more effective [3]. In particular, reuse of requirement specifications can lead to a significant decrease in development time and cost, as it usually leads to reuse of all work-products down-stream the development process [2]. Our work, hence, focuses on the possibility of reusing the most promising of the UML models, i.e. use cases and their event flows expressed in sequence diagrams.

2: Reuse Model for the UML Specifications

One of the key factors for the successful reuse of software is to equip its developers with the capability of effective scanning, browsing and searching of the domain model to locate design artefacts suitable for the newly constructed systems [4]. However, in order to search and find the required artefacts, we first need to be able to:

- represent and store the essential properties of all domain model components,
- formulate a query against the domain model,
- match the artefact representation against the query,
- order and select the artefacts by the degree of their proximity to the search criteria,
- retrieve the selected artefacts.

To address these issues we have to decide how to describe the properties of UML objects. We consider two alternative approaches, i.e.

- lexical, in which we infer the meaning of domain objects from their names; and
- semantic, in which we infer the meaning of domain objects from the model itself.

The lexical methods that consider text and word matching alone are not very useful in our situation since the properties of use cases and event flows are associated with the structure and the semantics of their models rather than the words used in their description. Hence, in order to judge the similarity of UML artefacts it is also necessary to ground the search and classification techniques in the semantics of reusable artefacts. To make the best of the two approaches, in our work, we combine the lexical and semantic approaches to judge the *similarity* of artefacts searched for. We use the semantic method when the artefact attributes are known to be in the domain model and so we can compare their *semantic distance*. And we use the lexical method when artefact attributes are not found in the domain model and so their *affinity* can only be guessed based on the lexical properties of their names. In our explanation, we have used some terminology that needs to be formally defined (Cf. Figure 4).

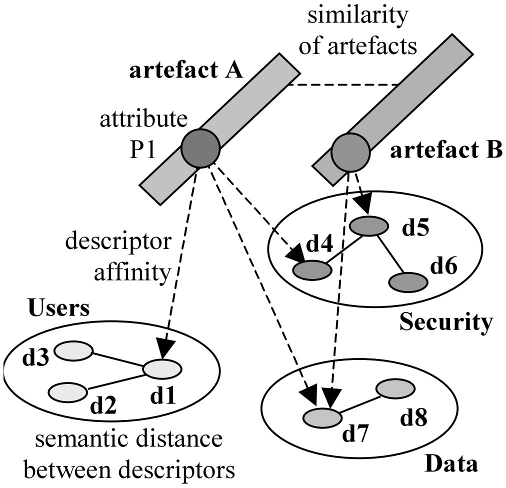


Figure 4: Explanation of terminology

Similarity is the measure of artefact closeness, based on the affinity of their attributes and similarity of their sub-components. E.g. the selected attributes of artefacts A and B may be compared based on their attributes described in terms of descriptor classes “Users”, “Security” and “Data”. The similarity measure can also be used to determine the match between the query and stored artefacts.

Semantic distance is a measure defining semantic closeness of descriptors within the same class. E.g. “Check Password” (d4) and “Authorise User” (d5) are in the same descriptor class “Security”, and they are semantically very close (as opposed to d4 and d6 which are further apart). Note that in the current version of the system, to simplify the operation of our system, all descriptors in the same “cluster” are regarded as identical, i.e. their semantic distance is zero.

Affinity is the level of natural attraction between an artefact attribute and a descriptor used in its classification. E.g. an attribute “Client Password” (P1) will have a natural affinity towards event clusters “Security” (the descriptor “Check Password”) and “Users” (the descriptor “Customer”), simply based on the words used in its name.

For the purpose of comparing use cases, we will consider them simply as collections of event flows (Cf. Figure 5). Hence, use case similarity is based entirely on the similarity of its component event flows. Since an event flow is simply a sequence of events, thus, event flow descriptor can be considered a vector consisting of event descriptors, being their names. When defining a new event flow vector (Cf. Figure 6), we can either utilise the event descriptors already defined in the domain model or we can create new event descriptors (and hence their associated events) using the terms in a domain lexicon. Having event flow vectors, we can combine them into a use case representation. We can also use such vectors to compare one event flow against the other. We can determine use case similarity by determining the number of matching event flows and the degree of their match. Queries can also be represented in a vector form and then compared against the domain model containing similar vectors in search for the elements of the model most suitable for retrieval and possible reuse.

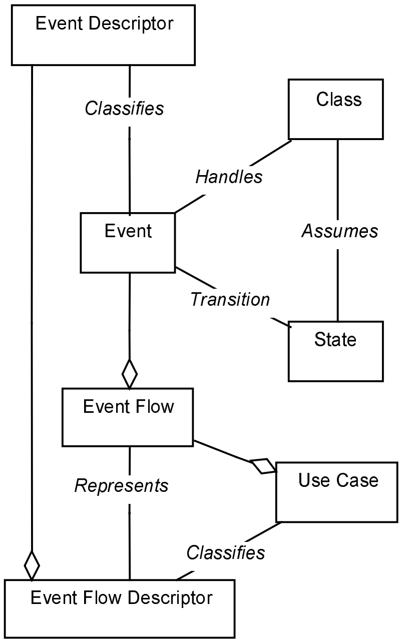


Figure 5: Semantic classification of UML models

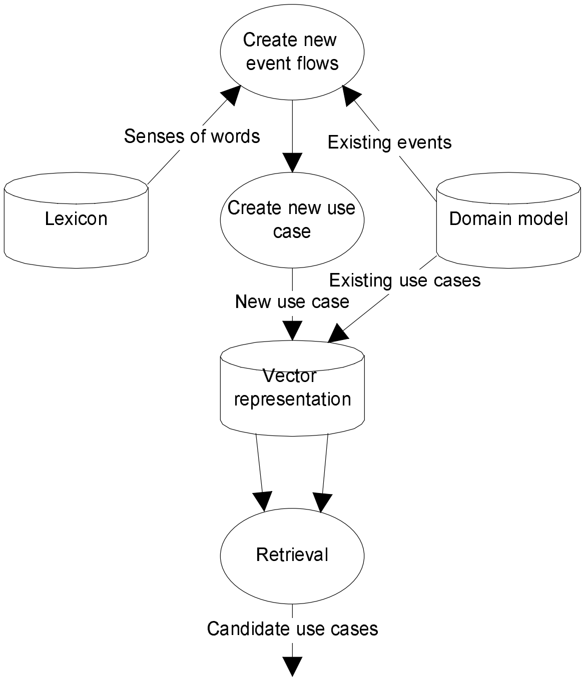


Figure 6: Overview of the method

3: Representation of event flows and use cases for classification

We believe that event flows are the most fundamental component of use cases and hence should be used as building blocks of the retrieval process. The existence of certain functionality is more important than the sequence of their occurrence in the object's life, hence we discard the flow sequences as unimportant to the classification of UML specifications. Besides, events often bear some implicit information about sequence, i.e. certain events are always in the same sequence e.g. the events in a login procedure.

Event flows can be described at different levels of abstraction, i.e.

- as a sequence of external events between the system and actors,
- as a sequence of external events and some main internal events represented as reflexive events, and
- as a collection of external and all internal events between system components and actors.

Currently, we only consider the external events and some of the main internal events. However, if it were necessary to broaden or narrow the search, another level of abstraction could be used.

There are also different ways of comparing events flows. One approach could be to compare event flow events. However, considering the large number of different events and event flows in a domain model, the calculation of possible conceptual distances between them would be quite infeasible. In order to reduce the complexity of the matching process, we decided to cluster events into event descriptor groups. Such clustering can be automated using techniques drawn from the area of information retrieval or it can be done manually. At this stage of our project we cluster events manually at considerable expense. The advantage of manual clustering is that events are clustered in a very reliable and accurate fashion. A serious disadvantage of simplistic clustering techniques is the lack of any empirical or theoretical basis for their use, e.g. statistical, for setting the semantic distances between events belonging to the same cluster. We assume that whenever two events are in the same cluster, then the semantic distance between them is considered to be zero. This yields a less accurate comparison of event flows. In general, the decision on the selection of methods and techniques for event clustering should be left up to the 'development-for-reuse' expert [7]. Table 1 shows an example of an event cluster that also represents an event descriptor.

In our representation of events, all events that are part of the domain model are already clustered. It can happen, however, that events used for the classification of a newly created event flow are not part of the domain model. In this situation the lexical description of the event is used to determine its most appropriate cluster.

Each event can be described by a number of words. In order to find an accurate cluster, the synonyms and the meaning of words will be taken into consideration. A lexicon, such as WordNet [6], can be used in order to achieve this. The lexicon is able to show senses of each word, i.e. all different meanings of the word and for each meaning its synonyms. Each event can thus be described by a number of senses. A cluster is described by the collection of senses used to describe all its events. For a new event the affinity to each cluster can be determined (Cf. Figure 7).

Optionally it might also be possible to show the software developer a list with candidate clusters so that he can choose the best representation of event information. The cluster of the new event will be used to calculate the similarity between the new event flow and the event flows from the domain model.

Authorization cluster
Enter PIN
Verify PIN
Verify credit card number
Re-enter pin
Enter credit card number
Enter expiring date
Sign receipt

Table 1: Event cluster

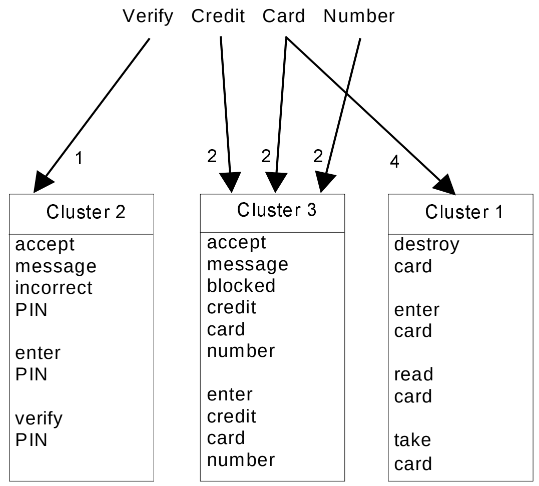


Figure 7: Affinity to each cluster

An entire event flow can thus be represented as a vector where each dimension represents the number of events in a particular cluster. Figure 8 and Figure 9 show an example.

Withdraw money from ATM	C	Deposit money through ATM	C
read card	1	read card	1
enter PIN	2	enter PIN	2
verify PIN	2	verify PIN	2
request to withdraw money	3	request to deposit money	3
select account	6	select account	6
enter amount of money	4	enter amount of money	4
approve transaction	7	enter money	4
deduct money from account	5	check amount of money	4
take card	1	add money to account	5
take money	4	take card	1

Figure 8: Sample event flows (C represents the cluster number of an event)

Cluster	Withdraw money from ATM	Deposit money through ATM
1	2	2
2	2	2
3	1	1
4	2	3
5	1	1
6	1	1
7	1	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0

Figure 9: Vectors of sample event flows (represented in terms of event clusters)

A use case is represented as a collection of event flows and thus as a collection of event flow vectors. These vectors are used to determine the use case similarity.

4: Event flow similarity

The two vectors of event flows are compared to determine the difference between them. The similarity between vectors could be determined using the following formula [10], [9]:

$$similarity(Q,D) = \frac{\sum_{c=1}^{n} q_c \cdot d_c}{\sqrt{\sum_{c=1}^{n} (d_c)^2 \cdot \sum_{c=1}^{n} (q_c)^2}}$$

Where:

Q = Vector of the query event flow

D = Vector of the event flow from the domain model

q_c = Value of dimension c of vector Q

d_c = Value of dimension c of vector D

n = total number of dimensions

Not every cluster in the event flow vector contributes equally to the similarity between two event flows. A number of factors influence the importance of events and clusters in event flows:

- *The nature of an event*. External events are more important than internal events. The external events define what functionality the system offers and the internal events define how the system achieves this functionality.
- *Generality of events*. Events that are used often in various event flows contain less information about the semantics of an event flow than the events that are used only sporadically.

The importance of a cluster can be expressed as an importance factor. In our system, we consider the frequency of event use as the most important of the factors, and we calculate it as follows:

$$i_c = 1 - r_c / r_t$$

Where:

i_c = Importance factor of cluster c

r_c = Number of references to cluster c by all event flows in a domain model

r_t = Number of references to all clusters by all event flows in a domain model

The updated formula to calculate the difference between two event flows is:

$$similarity(Q,D,I) = \frac{\sum_{c=1}^{n} q_c \cdot d_c \cdot i_c}{\sqrt{\sum_{c=1}^{n}(i_c \cdot d_c{}^2) \cdot \sum_{c=1}^{n}(i_c \cdot q_c{}^2)}}$$

Where:

Q	= Vector of the query event flow
D	= Vector of the event flow from the domain model
I	= Importance vector
q_c	= Value of dimension c of vector Q
d_c	= Value of dimension c of vector D
i_c	= Value of dimension c of vector I
n	= total number of dimensions

The calculation of difference between event vectors 'Withdraw money from ATM' and 'Deposit money through ATM' (Cf. Figure 8 and Figure 9) results in a similarity of 95% (where the importance vector I is taken from our current domain model).

5: Use case similarity

There are many possible approaches to the calculation of similarity between use cases. One such possibility is to match event flows of a use case with the most similar event flows of another use case. In order to achieve this, all event flows can be compared with each other and the best pairs of event flows can be used to determine similarity between use cases. Another possibility is to discard event flows all together and to consider use cases as a collection of their events. Determining similarity can then be calculated in exactly the same fashion as the similarity between event flows.

The main advantage of using event flows to determine use case similarity is that the structure of a use case is considered as important. Precise information can be given about the match of each event flow. It can occur, for instance, that two use cases have an overlapping set of event flows and some totally different event flows. If a use case is considered as a collection of event flows in order to determine similarity between use cases then it is possible to detect the identical event flows. If a use case is considered as a collection of events then an average similarity will be the result because the high similarity of the overlapping event flows will be disguised by the totally different event flows. Therefore, the structure of use cases will be used in order to determine similarity between use cases.

We consider only the event flows that are used to describe the system main functionality. This means that the exceptional event flows are not taken into consideration at all. They handle erroneous situations, and are thus directly derived from the primary and alternate event flows. Therefore the emphasis will be on primary and alternate event flows.

The following formula is used to determine the lower bound of the similarity between use cases:

$$similarity(U,V) = \frac{\sum_{t=1}^{x} similarity(Q_{ut}, D_{vt}, I)}{x}$$

Where:

U	= Query use case
V	= Use case from domain model
Q_{ut}	= Vector of the event flow t from query U
D_{vt}	= Vector of the event flow t from V
I	= Importance vector
x	= min(|U|,|V|)

The pairs of Q_{ut} and D_{vt} are chosen in such a way that neither Q_{ut} or D_{vt} could be used in more than one pair. The pairs are constructed iteratively, by matching event flows in the decreasing order of their similarity. This approach does not necessarily result in the optimum solution but it is a good indicator of use case similarity and provides the heuristic that avoids the calculation of similarities for all combinations of event flow pairs.

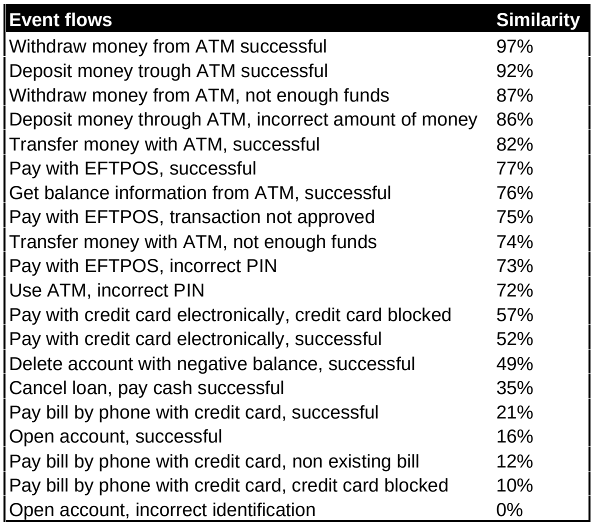
There are several ways to present suitable use cases to the software developer. In order to provide the software developer good insight in the existing use cases of the domain model, two presentations will be used:

- Use cases will be ranked by overall similarity. This indicates that the highest-ranked use case is most similar to the new use case, with respect to all component event flows.
- Use cases ranked on the number of event flows that are very similar. The similarity between use cases only indicates the average similarity of all the event flow pairs. If the similarity between two use cases is average then it can mean that all the event flow pairs have an average similarity or that some event flow pairs have low and some high similarity. The event flow pairs with high similarity are interesting because they give good opportunities for reuse and, hence, this information is presented to the software developer.

6: Example

We have constructed a small prototype to test our model of use case similarity and reuse. Twenty event flows were built using approximately 50 events. The events have been clustered manually. Next, the vectors of existing event flows have been compared with the vector of a new event flow 'Withdraw money from cash dispenser'. The results are shown in Figure 10. So far, we can see that the results are promising.

We experienced that the way of clustering events is the major factor that influences the quality of the results. Incorrectly formed clusters cause bad results. Therefore, this task should be done with great care.



Event flows	Similarity
Withdraw money from ATM successful	97%
Deposit money trough ATM successful	92%
Withdraw money from ATM, not enough funds	87%
Deposit money through ATM, incorrect amount of money	86%
Transfer money with ATM, successful	82%
Pay with EFTPOS, successful	77%
Get balance information from ATM, successful	76%
Pay with EFTPOS, transaction not approved	75%
Transfer money with ATM, not enough funds	74%
Pay with EFTPOS, incorrect PIN	73%
Use ATM, incorrect PIN	72%
Pay with credit card electronically, credit card blocked	57%
Pay with credit card electronically, successful	52%
Delete account with negative balance, successful	49%
Cancel loan, pay cash successful	35%
Pay bill by phone with credit card, successful	21%
Open account, successful	16%
Pay bill by phone with credit card, non existing bill	12%
Pay bill by phone with credit card, credit card blocked	10%
Open account, incorrect identification	0%

Figure 10: Similarity analysis results

7: Summary and future work

This article described a method of representing and comparing use cases in the UML domain model. Such methods can be applied as the search mechanism of the specification and design reuse engine. We believe that reuse at the earliest stage of software development will bring the greatest savings in development cost and time. Our method has been tested on a small model, but considering the method's similarity to those employed in information retrieval, we believe it is scalable to the larger domain models. The next stage in our work will be to evaluate the system using a considerably larger domain model.

Meanwhile, we are investigating the semantic relationships between various UML artefacts in order to predict the impact of partial reuse of use cases components. When a similar use case is found we want to supply the reuser with relevant information about the consequences of changing a use case. The reuser will be presented with the information about which artefacts downstream the development life-cycle need to be changed and which can be reused without any alteration.

We are also conducting experiments to find out whether undergraduate students enrolled in systems analysis and design course are able to work and study more effectively (accurately, completely and efficiently) with our prototype system than without it. As the number of students under study is quite large, the conclusive results are not available at this stage.

References

[1] G. Arango and R. Prieto-Diaz, “Domain Analysis Concepts and Research Directions,” in *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 9-32.

[2] T. Biggerstaff and C. Richter, “Reusability Framework, Assessment and Directions,” *IEEE Software*, vol. 41, 1987.

[3] J. L. Cybulski, “Introduction to Software Reuse,” . University of Melbourne, Australia, 1996.

[4] T. Isakowitz and R. J. Kauffman, “Supporting Search for Reusable Software Objects,” *IEEE Transactions on Software Engineering*, vol. 22, pp. 407-423, 1996.

[5] I. Jacobson, M. Ericsson, and A. Jacobson, *The Object Advantage - Business Process Reengineering with Object Technology*. Menlo Park, CA: Addison-Wesley, 1994.

[6] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller, “Five Papers on WordsNet,” Princeton University CSL Report 43, 1990.

[7] J. M. Moore and S. C. Bailin, “Domain Analysis: Framework for Reuse,” in *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 179-203.

[8] Rational-Software-Corporation, “Object-Oriented Analysis and Design Using UML Version 3.0,” 1997.

[9] C. J. Rijsbergen, *Information Retrieval*. London, UK: Butterworth, 1979.

[10] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Readings, Massachusetts: Addison-Wesley Pub. Co., 1989.