Software Design using Analogy and WordNet

Paulo Gomes, Francisco C. Pereira, Nuno Seco, Paulo Paiva, Paulo Carreiro, José L. Ferreira, Carlos Bento

CISUC – Centro de Informática e Sistemas da Universidade de Coimbra. Departamento de Engenharia Informática, Polo II, Universidade de Coimbra. 3030 Coimbra
{pgomes, camara}@dei.uc.pt, {nseco, paiva, carreiro}@student.dei.uc.pt, {zeluis, bento}@dei.uc.pt
http://rebuilder.dei.uc.pt

Abstract. Software design is a complex cognitive task. Given the lack of consistent and general methodologies, it often demands creative capacities that sometimes bring our insight closer to arts than to an engineering field. Much of the success obtained by software engineers in their designs, comes from their acquired know-how. One way to improve their work, is by providing CASE tools capable of assisting the software design task in two significant ways: storing past designs, and providing cognitive support for design space exploration. In this paper, we focus on the second aspect, presenting an approach to software design using analogy. We also describe how analogical reasoning can be combined with a general ontology – WordNet. This ontology provides object classification, and a conceptual network, in which we base our similarity mappings. An experimental study of the analogy-WordNet combination is also presented in this paper.

Keywords: Analogy, WordNet, Software Design, Case-Based Reasoning

Paper Track

Conference Topics: Reasoning Models, Natural Language Processing, Reuse of Knowledge, Case-Based Reasoning

1 Motivation and Goals

Software development usually comprises five different phases: analysis, design, coding, unit testing, and integration [1]. The analysis or specification phase identifies what the software must do. The design phase defines, at a conceptual level, how it is going to be done (defines the software behavior and structure). The goal of the coding phase is to implement the software modules. The aim of the unit test phase is to verify that all functions in each software module are according to the definitions stated in the design phase. The integration phase is to verify that the implemented software conforms properly with what was defined in the specification phase.

In our work we focus in the design phase due to various factors: it is an important phase, because most of the decisions made at this stage will constraint all the other phases; it is also a more complex task than the analysis phase, requiring more expertise and know-how from the developers; there are few computational tools that give cognitive support to designers; and knowledge at this phase is at a more abstract level and less formal, than the coding phase.

Analogy [2-5] is regarded as one important process in design [6]. It comprises the mapping between a source design and a target design. It can be efficiently used for transferring ideas across different domains. This also enables the exploration of areas of the design space that are not explored usually, allowing the generation of alternative designs. These designs can stimulate or offer the designer new ideas, and possibly better design solutions.

In this paper we present an approach to software design using analogy, which is integrated in a Computer Aided Software Engineering (CASE) tool named REBUILDER. This tool is based on Case-Based Reasoning [7, 8] and comprises a Knowledge Base with several types of knowledge, including WordNet, a general ontology. This paper presents the experimental results obtained with our approach in the study of the relation between WordNet and the analogy mechanism developed.

The next section describes related work on analogy and in special in the software development domain. Then we present our approach and section 4 describes the experimental results obtained. Section 5 finishes with a discussion.

2 Analogy

Analogical reasoning [2-5] is a widely used problem solving method. It consists in the transference of knowledge from one domain to a different domain. This transference is based on similarities between past problems and the new problem. The transferred knowledge is then used to generate solutions for the new problem. The main benefit of analogical reasoning is its capability to transfer knowledge from one domain to a different domain – cross-domain transfer of knowledge. It is this cross-domain transfer that enables the generation of new designs, and can be considered as an explorative process.

There are some research works that address the area of analogy applied to software reuse. From these we point out the work of Maiden and Sutcliffe in Ira [9]. Ira is a

CASE tool based on analogy that enables the reuse of software specifications. It provides user support for the task of system specification. Specification reuse involves three processes: categorization of a new problem, selection of candidate specifications, and adaptation of the selected specification to the new domain. Ira addresses these three issues by: obtaining the description of the new target problem from the software engineer; controlling the interaction with the user during selection and adaptation of an analogous specification; and reasoning with critical problem features to match new problems. While Ira is intended to the specification phase, REBUILDER works in the design phase.

Jeng and Cheng [10] use formal specifications to represent software components and use analogy to retrieve them. They base their approach on a software formal specification, which is hard to be dealt by humans. Software components can be requirements, design knowledge, code segments, or test plans. Thus their tool targets all the phases in software development, which makes the system hard to manage due to the different level of abstraction and specificity of each type of knowledge.

Spanoudakis [11] developed a computational model of similarity for analogical software reuse based on conceptual descriptions of software artifacts. Their approach is based on semantic similarity of software objects. ROSA [12] is also a CASE tool that reuses object-oriented specifications using analogy.

3 Our Approach to Analogy

Our approach to software design reuse with analogy is integrated in a CASE tool based on CBR, named REBUILDER. The main goal of REBUILDER is to provide intelligent support for software engineers during the design phase. This includes: retrieval of past designs based on similarity concepts; suggestion of new designs; verification of design constraints; evaluation of design properties; and learning new design knowledge. Analogy is one of the modules in the CBR engine (see Figure 1).

REBUILDER is intended to be used within a corporation environment, centralizing the corporation past designs in its knowledge base. There are two types of users interacting with the system. Software designers, using REBUILDER as an UML editor, and an administrator with the main task of keeping the knowledge base (KB) consistent and updated.

In order for a CASE tool to be used, the design language must be intuitive and human-centred. This is also true for software design where it is common the use of visual languages to represent designs. One worldwide software design language is the Unified Modelling Language [13], best known as UML. This language provides a representation for all the software development phases. By choosing UML as the representation language, we are providing the user with a design standard.

Figure 1 shows the architecture of REBUILDER. It comprises four main modules: UML editor, KB manager, KB, and CBR engine. The UML editor is the system front-end for the software designer, comprising the working space for design. The KB management is the interface between the KB and the system administrator. It provides access to various sub modules of knowledge, allowing the administrator to add, delete, or change knowledge from the KB, and fine-tuning it.

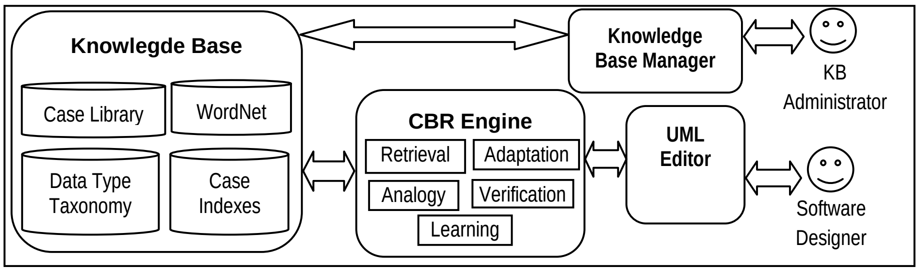


Figure 1 - The architecture of REBUILDER.

The KB comprises four modules: data type taxonomy, case library, case indexes, and WordNet. The data type taxonomy provides *is-a* relations between data types used in UML. This structure is used when the system has to compute the similarity between data types. The case library stores the design cases, each one representing a software design. They are stored in UML files created by the editor. Though UML has several types of diagrams, REBUILDER uses only class diagrams for reasoning, since it is the most used diagrams. The case indexes are used for case retrieval, allowing a more efficient retrieval. WordNet is a lexical reference system [14], used in REBUILDER as a general ontology that categorizes case objects.

The CBR engine performs all the inference work in REBUILDER. It comprises five sub modules: retrieval, analogy, adaptation, verification, and learning. The retrieval module searches the case library for designs or design objects similar to the query. The most similar ones are presented to the user, allowing the user to reuse these designs or parts of them. Retrieved designs can also suggest new ideas to the designer, helping him to explore the design space. The analogy module uses the retrieval module to select the most similar designs from the case library, and then maps them to the query design. This module is the subject of this paper, and it will be described in more detail in the next sections. The resulting mapping establishes the knowledge transfer from the old design to the query design. Analogy goes further than case retrieval, creating new designs. The adaptation module can be used to adapt a past design (or part of it) to the query design using design composition. The main usage of this module is in combination with retrieval. The verification module checks the current design for inconsistencies. The learning module acquires new knowledge from the user interaction, or from the system interaction. The next subsections describe the WordNet ontology, since it is needed to know what knowledge is in the WordNet, and then describe the analogy module.

3.1 WordNet

WordNet is used in REBUILDER as a common sense ontology. It uses a differential theory where concept meanings are represented by symbols that enable a theorist to distinguish among them. Symbols are words, and concept meanings are called synsets.

A synset is a concept represented by one or more words. If more than one word can be used to represent a synset, then they are called synonyms. But there is also another word phenomenon important for WordNet: the same word can have more than one different meaning (polysemy). For instance, the word mouse has two meanings, it can denote a rat, or it can express a computer mouse.

WordNet is built around the concept of synset. Basically it comprises a list of word synsets, and different semantic relations between synsets. The first part is a list of words, each one with a list of synsets that the word represents. The second part, is a set of semantic relations between synsets, like *is-a* relations (rat *is-a* mouse), *part-of* relations (door *part-of* house), and many other relations. In REBUILDER we use the word synset list and four semantic relation: *is-a*, *part-of*, *substance-of*, and *member-of*. Synsets are classified in four different types: nouns, verbs, adjectives, and adverbs.

REBUILDER uses synsets for categorization of software objects. Each object has a context synset which represents the object meaning. In order to find the correct synset, REBUILDER uses the object name, and the names of the objects related with it, which define the object context. The object's context synset can then be used for computing object similarity (using the WordNet semantic relations), or it can be used as a case index, allowing the rapid access to objects with the same classification.

3.2 The Analogy Module

Analogical reasoning is used in REBUILDER to suggest class diagrams to the designer, based on a query diagram. The analogy module has three phases: identifying diagrams candidate for analogy, mapping the candidate diagrams, and creation of new diagrams by knowledge transfer between the candidate diagram and the query.

Candidate selection is the first phase, and is the most important one. Selected candidates must be appropriate, otherwise the whole mapping phase can be at risk. Most of the analogies that are found in software design are functional analogies, that is, the analogy mapping is done using the functional similarity between objects. Since functional similar objects are categorized in the same branch (or near) in the WordNet *is-a* trees, and the retrieval algorithm uses these semantic relations to retrieve objects, the analogy module uses the retrieval module for this first phase. Thus, the analogy module benefits from a retrieval filtering based on functional similarity.

The second phase of analogy is the mapping of each candidate to the query diagram, yielding an object list correspondence. This phase relies on two alternative algorithms: one based on relation mapping, and the other on object mapping, but both return a list of mappings between objects.

The relation-based algorithm uses the UML relations to establish the object mappings. It starts the mapping selecting a query relation based on an UML heuristic (independence measure), which selects the relation that connects the two most important diagram objects. The independence measure is an heuristic used to assign to each diagram object a value based on UML knowledge that reflects an object's independence in relation to all the other diagram objects. Then it tries to find a matching relation on the candidate diagram. After it finds a match, it starts the mapping by the neighbour relations, spreading the mapping using the diagram relations. This algorithm maps objects in pairs corresponding to the relation's objects.

The object-based algorithm starts the mapping selecting the most independent query object, based on an UML independence heuristic. After finding the corresponding candidate object, it tries to map the neighbour objects of the query object, taking the object's relations as constraints. Both algorithms satisfy the structural constraints defined by the UML diagram relations. Most of the resulting mappings are only partial, so, a way of mapping ranking is needed, see section 3.3.

An important issue in the mapping stage is which objects to map. Most of the times, there are several candidate objects for mapping with the problem object. In order to solve this issue, we have developed a metric that is used to choose the mapping candidate. Because we have two mapping algorithms, one based on relations and another on objects, there are two metrics: one for objects, and other for relations. Sections 3.4 and 3.5 will present these metrics in more detail.

The final phase is the generation of new diagrams using the established mappings. For each mapping the analogy module creates a new diagram, which is a copy of the query diagram. Then using the mappings between the query objects and the candidate objects, the algorithm transfers knowledge from the candidate diagram to the new diagram. This transference has two steps: first there is an internal object transference, and then an external object transference.

In the internal object transference, the mapped query object gets all the attributes and methods from the candidate object that were not in the query object. This way, the query object is completed by the internal knowledge of the candidate object. The second step transfers neighbour objects and relations from the mapped candidate objects to the query objects from the new diagram. This transfers new objects and relations to the new diagram, expanding it.

3.3 Criteria for Mapping Ranking

The analogy algorithm uses four different criteria for ranking the mappings:

- Based on the number of mapped objects:

$$\frac{K}{PObjs} \tag{1}$$

- Based on the independence sum of mapped objects in the problem:

$$\frac{IndSum(MappedPObjs)}{IndSum(PObjs)} \tag{2}$$

- Based on the independence sum of mapped objects in the problem and the case:

$$1-\sum_{i=1}^{k}\left|\frac{PM_i}{\sum_{j=1}^{k}PM_j}-\frac{CM_i}{\sum_{j=1}^{k}CM_j}\right| \tag{3}$$

- Based on the number of mapped objects and independence sum:

$$w1*\frac{k}{PObjs}+w2*\left(1-\sum_{i=1}^{k}\left|\frac{PM_i}{\sum_{j=1}^{k}PM_j}-\frac{CM_i}{\sum_{j=1}^{k}CM_j}\right|\right) \tag{4}$$

Where *K* is the number of mapped objects. *PObjs* is the number of objects in the Problem. *IndSum(MappedPObjs)* is the Sum of the independence of mapped objects in the problem. *IndSum(PObjs)* is the Sum of the independence of objects in the problem. PM_i is the independence value of mapped problem object *i*. CM_i is the independence value of mapped case object *i*. *w1* = 0.5 ; *w2* = 0.5. What ranking to use is up to the user to select, the default is the first one.

3.4 Similarity Metric for Mapping Objects

The similarity metric for mapping two objects (A and B) is based on three factors. The first is the distance between A's synset and B's synset in the WordNet ontology (D_1). For the second factor, the Most Specific Common Abstraction (MSCA) between A and B synsets must be found. The MSCA is basically the most specific synset which is an abstraction of both object's synsets. Considering the distance between A's synset and MSCA (D(A,MSCA)), and the distance between B's synset and MSCA (D(B,MSCA)), then the second factor is the relation between these two distances (D_2). This factor tries to account the level of abstraction of concepts. The last factor is the relative depth of MSCA in the WordNet ontology (D_3), which tries to reflect the objects' level of abstraction. The concrete formulas are:

- Similarity metric between A and B:

$$\begin{array}{c} -1 \Leftarrow \text{does not exist MSCA} \\ w_1 * D_1 + w_2 * D_2 + w_3 * D_3 \Leftarrow \text{exists MSCA} \end{array} \tag{5}$$

Where w_1, w_2 and w_3 are weights associated with each factor, the values are selected based on empirical work and are: 0.55, 0.3, and 0.15.

$$D_1 = 1-\frac{D(A,MSCA)+D(B,MSCA)}{2*DepthMax} \tag{6}$$

Where *DepthMax* is the maximum depth of the *is-a* tree of WordNet. It's current value is 14.

$$D_2 = 1 \Leftarrow A = B$$ (7)

$$D_2 = 1 - \frac{|D(A,MSCA) - D(B,MSCA)|}{\sqrt{D(A,MSCA)^2 + D(B,MSCA)^2}} \Leftarrow A \neq B$$

$$D_3 = \frac{Depth(MSCA)}{DepthMax}$$ (8)

Where Depth*(MSCA)* is the depth of MSCA in the *is-a* tree of WordNet.

3.5 Similarity Metric for Mapping Relations

The previous metric is mainly used in the object-based mapping algorithm, where the best candidate object must be selected for mapping with a problem object. For the relation-based mapping algorithm, the selection metric is not for ranking objects, but for ranking relations. This metric is based on the object similarity metric, but takes into account the relation, and the relation's objects. The metric for ranking mapping relations is:

Suppose that the two relations are R_1 (A-B) and R_2 (C-D), and: MAB is the MSCA between A and B, MAC is the MSCA between A and C, MBD is the MSCA between B and D, and MCD is the MSCA between C and D.

If (MAB exists) and (MAC exists) and (MBD exists) and (MCD exists) Then the metric value is:

$$w_1 * ASim(A,C) + w_2 * ASim(B,D) + w_3 * RASim(R_1, R_2)$$ (**9**)

Where *ASim(A,C)* is the similarity metric for mapping objects A and C (described in section 3.4). Weights w_1, w_2 and w_3 are: 0.25, 0.25 and 0.5. *RASim(R_1,R_2)* is given by:

$$RASim(R_1, R_2) = \begin{bmatrix} w_1 * Length(R_1, R_2) \\ + w_2 * Angle(R_1, R_2) \\ + w3 * Depth(R_1, R_2) \end{bmatrix}$$ (**10**)

$$Length(R_1, R_2) = 1 - \frac{\left| \begin{matrix} \sqrt{D(A,MAB)^2 + D(B,MAB)^2} - \\ \sqrt{D(C,MCD)^2 + D(D,MCD)^2} \end{matrix} \right|}{\begin{bmatrix} \sqrt{D(A,MAB)^2 + D(B,MAB)^2} + \\ \sqrt{D(C,MCD)^2 + D(D,MCD)^2} \end{bmatrix}}$$ (**11**)

$$Angle(R_1, R_2) = 1 - \left| \frac{D(A, MAB)}{\sqrt{D(A, MAB)^2 + D(B, MAB)^2}} - \frac{D(C, MCD)}{\sqrt{D(C, MCD)^2 + D(D, MCD)^2}} \right| \quad (12)$$

$$Depth(R_1, R_2) = 1 - \frac{|P(A) + P(B) - P(C) - P(D)|}{2 * DepthMax} \quad (13)$$

Where w_1, w_2 and w_3 are weights with values: 0.25, 0.25 and 0.5. *Length(R_1,R_2)* is a factor that reflects the distance between objects in the relations. *Angle(R_1,R_2)* reflects the angle between the objects and the respective MSCAs. *Depth(R_1,R_2)* reflects the depth in the WordNet, or in other words the abstraction level of objects.

If (MAC exists) and (MBD exists) and (MCD not exists) and (MAB not exists) then the metric value is:

$$\frac{ASim(A, C) + ASim(B, D)}{2} \quad (14)$$

Else the metric value is –1, meaning that the relations have no similarity.

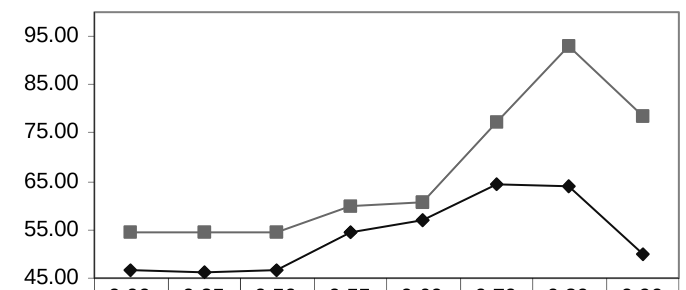
4 Experimental Results

The KB used comprises a case library with 60 cases. Each case comprises a package, with 5 to 20 objects (total number of objects in the knowledge base is 586). Each object has up to 20 attributes, and up to 20 methods.

The factor that constraints the analogy mapping is the value used as threshold for mapping objects and relations. This threshold is the minimum value allowed for a mapping to happen (values near zero indicate that everything can be mapped). We also wanted test the two developed mapping algorithms: relation-based and object-based.

Each of the 60 cases were used as a problem, and we used 16 threshold/algorithm combination, yielding 960 runs. In each of these runs, the five best solutions for each mapping ranking criteria (see section 3.3), coming from different cases, were gathered. This yields 20 solutions by run. The data gathered for each solution was: number of mappings, distance of mapped objects (in the WordNet), depth (in WordNet) of the MSCA between mapped objects, and percentage of correct mappings by solution. This last figure was obtained by human evaluation of what is considered a correct mapping. From these gathered data we derived the number of correct mappings by solution.

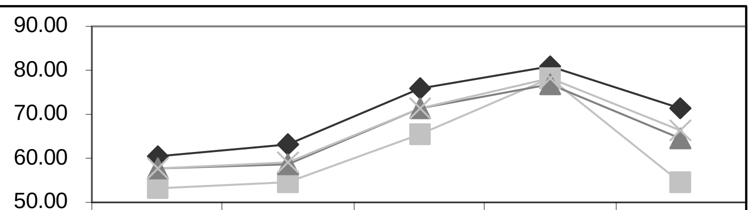
The results obtained for the average percentage of correct mappings is presented in Figure 2. From these results it is clear that the object-based mapping yields more accurate mappings and that the best threshold value is 0.8 for object-based mapping and 0.7 for relation-based mapping.



	0.00	0.25	0.50	0.55	0.60	0.70	0.80	0.90
◆ Relation-Based Mapping	46.54	46.12	46.71	54.37	56.91	64.55	64.00	49.93
■ Object-Based Mapping	54.35	54.39	54.66	60.04	60.61	77.35	93.11	78.63

Figure 2 - Average %of correct mappings.

In Figure 3 the results for the average percentage of correct mappings by mapping ranking criteria. The Mapping criteria corresponds to formula (1), Independence to formula (2), Mixed to (3) and Weighted to (4). From these results it can be seen that the first criteria is the best one in all situations. Then criteria Weighted is slightly better than Mixed, and the last one is the independence criteria.



	0.55	0.6	0.7	0.8	0.9
◆ Mapping	60.46	63.02	75.77	81.06	71.56
■ Independence	53.19	54.58	65.47	78.21	54.68
▲ Mixed	57.52	58.54	71.30	76.94	64.71
× Weighted	57.65	58.91	71.25	78.02	66.17

Figure 3 - Average % of correct mappings by ranking criteria, and by threshold.

5 Discussion

This paper presents an approach to analogy for software design. We integrate this approach with a CASE tool based on a CBR framework, which enables the exploration the design space through cases. We describe the results of an experimental study that relate the correct percentage of mappings with some mapping properties.

One of the main advantages of using analogy in a CASE tool for software design, is the opportunity of exploring design space regions not usually accessed by the designer. This can have two effects: boosting the designer's creativity, and generating novel designs, depending if the system does not provide a complete solution or it can generate the solution by itself. The integration of a common ontology like WordNet, has provided the analogy mechanism with a far more broad range of action, enabling it to establish cross-domain transfer of ideas.

There are some limitations in our approach. One of them is the need for good computational resources. This is due to the complexity of the reasoning processes involved, and also to the size and complexity of WordNet. Other limitation, is the generation of bizarre solutions, which is a normal effect of an exploration mechanism. This can be compensated with the insertion of the verification module in the analogical reasoning process.

Acknowledgments

This work was supported by POSI - Programa Operacional Sociedade de Informação of Portuguese Fundação para a Ciência e Tecnologia and European Union FEDER, under contract POSI/33399/SRI/2000, and by program PRAXIS XXI.

References

1. Boehm, B., A Spiral Model of Software Development and Enhancement. 1988: IEEE Press.
2. Gentner, D., Structure Mapping: A Theoretical Framework for Analogy. Cognitive Science, 1983. **7**(2): p. 155-170.
3. Hall, R.P., Computational approaches to analogical reasoning; A comparative analysis. Artificial Intelligence, 1989. **39**(1): p. 39-120.
4. Holyoak, K.J. and P. Thagard, Analogical Mapping by Constraint Satisfaction. Conitive Science, 1989. **13**: p. 295-355.
5. Thagard, P., et al., Analog Retrieval by Constraint Satisfaction. Artificial Intelligent, 1990. **46**: p. 259-310.
6. Gero, J., Computational Models of Creative Design Processes., in Artificial Intelligence and Creativity, T. Dartnall, Editor. 1994, Kluwer Academic Publishers.
7. Kolodner, J., Case-Based Reasoning. 1993: Morgan Kaufman.
8. Maher, M.L., M. Balachandran, and D. Zhang, Case-Based Reasoning in Design. 1995: Lawrence Erlbaum Associates.
9. Maiden, N. and A. Sutcliffe, Exploiting Reusable Specifications Through Analogy. Communications of the ACM, 1992. **35**(4): p. 55-64.
10. Jeng, J.-J. and B. Cheng. Using Analogy and Formal Methods for Software Reuse. in IEEE 5th International Conference on Tools with AI. 1993.
11. Spanoudakis, G. and P. Constantopoulos. Similarity for Analogical Software Reuse: A Computational Model. in 11th European Conference on Artificial Intelligence. 1994. Amesterdam, The Netherlands: John Wiley & Sons.
12. Tessem, B., et al., ROSA = Reuse of Object-oriented Specifications through Analogy: A Project Framework, 1994, Department of Information Science, University of Bergen. p. 23.
13. Rumbaugh, J., I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual. 1998, Reading, MA: Addison-Wesley.
14. Miller, G., et al., Introduction to WordNet: an on-line lexical database. International Journal of Lexicography, 1990. **3**(4): p. 235 - 244.