Construct a Tool for Aspect-Oriented Analysis and Design Based on Theme Approach

Nada N. Saleem **Mohammad H. Abdulwahab**
Nada_N_S@uomosul.edu.iq
College of Computer Sciences and Mathematics
University of Mosul

Received on: 19/10/2011 **Accepted on: 14/12/2011**

ABSTRACT

In this research, a (RADC-AO) tool was constructed and implemented for the requirements analysis, design and stub-code generation according to aspect-oriented (AO) concepts based on theme approach. RADC-AO automatically identifies crosscutting concerns in natural language requirements text by using natural language processing (NLP), analyze requirements and apply a set of operations on themes got in the analysis process, design classes and aspects, draw class diagram, and generates stub-code.

RADC-AO tested by input complete informal text requirements for payroll system (that contains security, logging, authorization, in addition to its core functionality which includes employees information entering, loans information entering, and payment calculation), RADC-AO successes in the test and gives good results.

Keyword: aspect-oriented (AO), RADC-AO tool, natural language processing (NLP)

بناء الأداة (RADC-AO) الجانبية التوجه في التحليل والتصميم يالاعتماد على منهج الموضوع

ندى نعمت سليم **محمد هيثم عبدالوهاب**
كلية علوم الحاسوب والرياضيات/ جامعة الموصل

تاريخ استلام البحث: 2011/10/19 **تاريخ قبول البحث: 2011/12/14**

الملخص

في هذا البحث تم بناء وتنفيذ الأداة (RADC-AO) لتحليل المتطلبات الخاصة بالبرمجيات وتصميم وتوليد الهيكل العام لشفرة البرمجة طبقاً لأفكار البرمجة جانبية التوجه والاعتماد على منهج الموضوع (theme). الأداة (RADC-AO) تحدد الهموم (concerns) المتشابكة في المتطلبات المتمثلة بهيئة نص باللغة الطبيعية بشكل آلي باستخدام معالجة اللغة الطبيعية، تحليل المتطلبات وتطبيق مجموعة من العمليات على المواضيع (themes) المستخرجة من تحليل المتطلبات، تصميم الـ (classes) والـ (aspects)، رسم مخطط الـ (classes)، ومن ثم توليد الهيكل العام لشفرة البرمجة.

تم فحص الأداة (RADC-AO) بإدخال نص كامل لمتطلبات نظام حساب الرواتب، يحتوي هذا النظام على (أمنية، توثيق، تفويض، بالإضافة إلى الوظائف الأساسية للنظام التي تضم إدخال معلومات الموظفين وإدارة هذه المعلومات، إدخال المعلومات الخاصة بالقروض وإدارة هذه المعلومات، حساب الرواتب). نجحت الأداة (RADC-AO) في الاختبار وأعطت نتائج جيدة.

الكلمات المفتاحية: البرمجة جانبية التوجه, الأداة (RADC-AO), معالجة اللغة الطبيعية

1. Introduction

Natural language processing (NLP) is the attempt to extract a fuller meaning from free text. This can be put roughly as figuring out who did what to whom, when, where, how and why[1]. As (NLP) technology matures, it is increasingly being used to support other computer applications[2]. There are many uses for the (NLP), the most important among them are (text preparation, information retrieval, automatic translation from one natural language to another, natural language interfaces to databases and other systems, and so on)[3].

Requirements are specifications of what should be implemented[4]. A requirement can be specified in many styles, may be specified in plain text, as diagrams, or as tables[5]. One common format for specifying requirements is simply to list each individual requirement without describing any hierarchy or other explicit relationships among the requirements[6].

A concern is anything that is of interest to a stakeholder, whether an end user, project sponsor, or developer. For example, a concern can be a functional requirement, a nonfunctional requirement, or a design constraint on the system[7].

Crosscutting concerns are behaviors that span multiple, often unrelated, implementation modules. In addition to, crosscutting concerns cannot be neatly separately from each other[8].

Crosscutting concerns are problem when modules in a system may interact simultaneously with several requirements, which mean that concerns are tightly intermixed, *code tangling* occurs. Since crosscutting concerns spread over many modules, related implementations also spread over those modules. The concerns are poorly localized, and this is called *code scattering*[8].

Aspect-oriented programming (AOP) is a novel topic in the software engineering and languages communities[9], which addresses the construction of software artifacts that traditional software engineering constructs fail to modularize[10]. AOP solves the tangling and scattering problems that other programming paradigms can face[11] and [12].

There are a number of approaches for aspect-oriented software development (AOSD) like Viewpoint-Based Aspect-Oriented Approaches, Goal Based Aspect Oriented Approaches, and Theme approach. The feature that characterized the theme approach is that this approach is new while other approaches are emerged as extensions of some non-AO approaches[13].

The theme is the foundation for the theme approach. A theme is a collection of structures and behaviors that represent one feature. Theme approach is for aspect oriented analysis and design; therefore it contains two levels analysis, and design. The Theme approach is divided into two segments: Theme/Doc and Theme/UML. These both operate on and refer to the same themes, but depict them at different phases of the lifecycle. Theme/Doc provides views and functional support for identification and depiction at the analysis phase, whereas Theme/UML allows standard UML modeling of relevant structure and behavior for each theme at the design phase[14].

There are many researchers working in the identification of crosscutting concerns in the requirements based on theme approach and other approaches, following a brief explanation about their works:

- 3CI tool: created by Busyairah, Zarinah[15][16].
- A novel use of text mining as a technology to assist in the discovery and verification of early aspects in requirements documents: created by Yan Wu, Mansour Zand, Harvey Siy, Victor Winter[17].
- Mining Aspects in Requirements: created by Américo Sampaio, Neil Loughran, Awais Rashid and Paul Rayson[18].

(NLP) was used in their researches to identify the crosscutting concerns in the text requirements, focuses on the crosscutting concerns identification, but neglecting to focus on analysis or design.

2.1 The Proposed RADC-AO Tool

The RADC-AO is proposed and constructed for aspect-oriented analysis, design and stub-code generation which is used as a mean to help software engineer to input the problem requirements in an informal text requirements then automatically output stub-code according to (AO) concepts. Figure 1 shows RADC-AO capabilities.

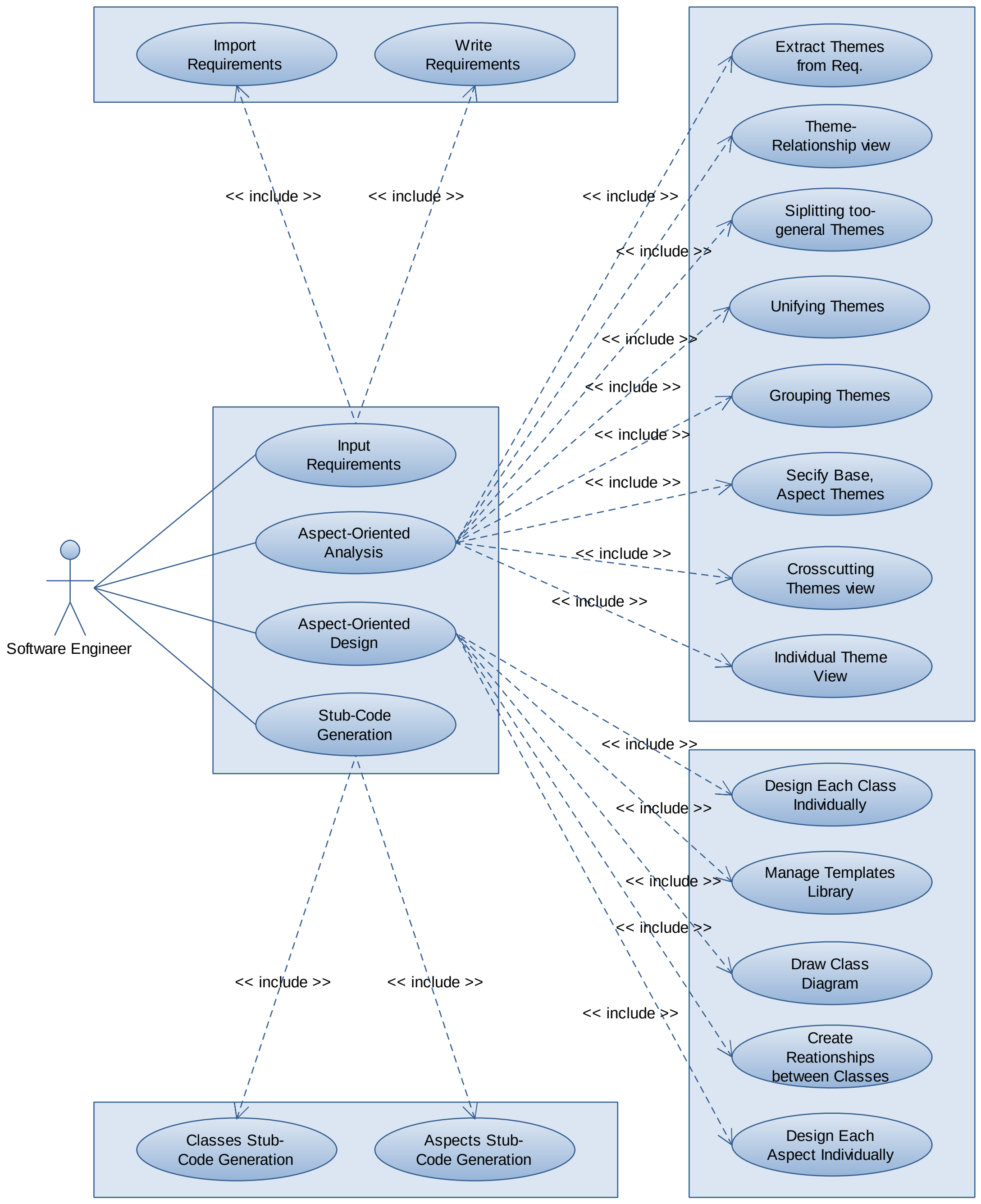


Figure 1: RADC-AO use cases.

2.2 RADC-AO description

To describe RADC-AO tool, a brief description of its capabilities must be explained, these capabilities are:

2.2.1 Inputting Requirements for RADC-AO

RADC-AO proposed tool dealing with informal requirements text. Therefore, it can read the requirements from text file or from (RADC-AO) tool requirements text field area.

2.2.2 Aspect-Oriented Analysis in RADC-AO

Aspect-oriented Analysis in RADC-AO containing the following steps:

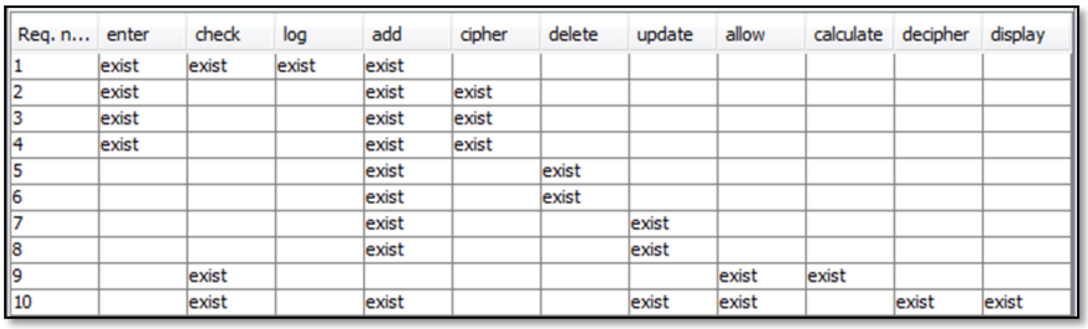
❖ Extracting Themes from Requirements Text

The first step in the aspect-oriented analysis is extracting themes from requirements, as we previously mentioned that RADC-AO is dealing with informal textual requirements which lead to dealing with textual natural language. Theme is well known as an action verb, from this point, there is a need for natural language processing to determine all verbs from the textual requirements. There are two ways to determine verbs in requirements: the first is using Part-of-Speech (POS) tagger which does not provide an indication about the relationship or dependency between (POS) especially between the verb and its subject, the verb and its direct or indirect object. The second way is using a parser which indicates the dependency between each (POS) and other related tokens. RADC-AO uses a parser to extract each verb and its subject and object if exist from requirements and save the information in two dimensional string.

After parsing process which determines all verbs in requirements, we need to return each verb to its base. This process is very important because we may face verb like 'save' and another verb like 'saving' which belong to the same action, but they appear different.

All linguistic processes are dealing with large database and exploit large amount of CPU and RAM, therefore and from the beginning we use a stop word list which contains all unwanted verbs, or trivial verbs and delete these trivial verbs from the string mentioned above if exist. This step is premature and considered as one operation that operates on themes named (delete unwanted themes).

After all the previous processes are completed, the matrix named theme-requirement relationship matrix is created which is depicted in Figure 2, the matrix containing each requirements and its themes, so it shows crosscutting themes when theme exists in more than one requirement.



Req. n...	enter	check	log	add	cipher	delete	update	allow	calculate	decipher	display
1	exist	exist	exist	exist							
2	exist			exist	exist						
3	exist			exist	exist						
4	exist			exist	exist						
5				exist		exist					
6				exist		exist					
7				exist			exist				
8				exist			exist				
9		exist						exist	exist		
10		exist		exist			exist	exist		exist	exist

Figure 2: Theme-Requirement Relationship Matrix.

❖ Theme Relationship View Graph

RADC-AO uses theme-requirement relationship matrix to draw theme relationship view by drawing each requirement, drawing each theme without redundancy, and linking between each requirement and its themes.

❖ Operating on Themes

Operating on themes contains several operations which are:

- **Splitting too General Themes**

To split a theme, we must be sure that theme was too general, as well known theme representing a behavior or an action or a verb, and mostly this a verb has an object, like 'add number' the verb is 'add' and the object is 'number' and all this information RADC-AO retains it. We must differentiate this 'add number' from 'add name', the first means a mathematical operation and the second may save the information name to a database, the differentiation in this case depends on the object. Therefore, we create a database for categorizing words (objects) into categories like 'number', 'information', 'security', and so on, where we face a theme that belongs to more than one requirement and every requirement has its own object category. Here, we must split this theme to these categories, and where we find theme its object not belonging to any category or it's unknown to RADC-AO database, RADC-AO force the software engineer to categorize this object manually by hand and save this object and its category.

- **Grouping Synonym Themes (Unifying Themes)**

Grouping synonym themes means there are themes that are synonym, but they are different in their lettering, like 'save', 'store', or 'cipher', 'encode', therefore we want to take each theme and extract all its synonyms from a database and search in the remainder themes if there is an accordance with them we merge them.

- **Grouping Themes that seem Encompassed by the Same Theme**

Grouping themes that seems encompassed by the same theme means that we may face themes that have some common structure, like 'add name', 'delete name', 'move next', 'move previous', all these themes are database operations and have common structures, or 'cipher', 'decipher', they are opposite themes, but they have very common structure, therefore we must group these themes as one parent theme that has the common structure 'generalization' and the original themes have the distinct structure 'specialization'. To do that RADC-AO contain a database for themes categories depending on the structure "verb + object category = theme category" it means that if we face theme 'add' and its object is 'name' and 'name' which belongs to information category as described in splitting too general themes paragraph, it means that this theme is dealing with database because we previously specified a rule that provides ('add' + 'information' = 'database'), another theme like 'delete' and its object is 'name' means that the delete theme also dealing with database, therefore RADC-AO will merge them and the output is one theme called 'database'. Also, when RADC-AO faces a verb and its object is unknown to it. It forces the software engineer to determine the new verb's rule and save the new rule to the database.

❖ Specifying Base and Aspect Themes

Aspect theme will be designed according to theme approach not according to object-oriented as base theme, therefore we must specify aspects, aspect is a theme that exists in more than one requirement or concern, the number of times that specific theme will be triggered to consider it as an aspect called Aspect Triggered Threshold (ATT), (ATT) is not specific, someone can consider it two times, someone else can consider it three times, whatever (ATT) can the software engineer specify it by inputbox appearing to him/her.

The number of times that each theme will be triggered in a given requirements called (NTT), RADC-AO tool is responsible to calculate (NTT) for all themes and then comparing it with (ATT) to decide each theme as an aspect or a base.

❖ **Crosscutting Theme View**

Crosscutting theme view is done by drawing all themes and their interaction between them (which theme trigger which theme) and then drawing all requirements and relating each requirement with the most dominant theme of it. Dominant theme is determined by the software engineer by listing all themes existing in the requirement to him/her, and he/she chooses the dominant theme from the list.

❖ **Individual Theme View**

There are two types of the individual theme view in RADC-AO:

- Base theme individual view: which draws each base theme and relates it with all requirements that exist in them, and relates the base theme with the subjects and objects relating to it.
- Aspect theme individual view: which draws each aspect theme and relates it with all requirements that exist in them, and relates the aspect theme with the subjects and objects relating to it.

2.2.3 Aspect-Oriented Design in RADC-AO

RADC-AO retain all analysis information, it uses this information for the design process. In the design process, there are two types of design, one for base themes design (base theme will be a normal class) and another for aspect themes design (aspect themes will be aspects).

2.2.3.1 Designing Classes:

Designing classes containing the following steps:

❖ **Designing each Class Individually**

Base theme will be a normal class. In base theme design RADC-AO take each base theme and extract its category as class name and consider all verbs belong to it as operations for that class and the objects if exist considered as attributes for that class. In addition, in the design level software engineer can delete class or create new class and add operations and attributes as he/she wishes, or renames class. Software engineer in design level in RADC-AO is supported by a library that contains many templates of reusable classes design, this library enables the software engineer to import classes or operations or attributes from it. This library enables software engineer to manage it by modifying the existing design templates or delete from it or add designs to it.

❖ **Drawing Class Diagram**

After completion of base theme design, RADC-AO tool will show all the classes previously designed according to UML class diagram standards, the resulted graph is shown in workspace that enables software engineer to dynamically treating with class diagram. After drawing a class diagram software engineer is able to create relationships between classes that are drawn, these relationships are (Inheritance, Aggregation, Association, Dependency, and Composition).

2.2.3.2 Designing Aspects:

RADC-AO can design aspects in two stages:

Stage 1 – Design Aspect Operations and Attributes:

In aspect theme designing RADC-AO takes each aspect theme and extracts it's category as aspect name and consider all verbs belong to it as operations for that aspect, and the objects if exist are considered as attributes for that aspect. In addition, in the design level software engineer can delete add or delete operation and attributes as he/she wishes.

Stage 2 – Design Aspect's Joinpoint, Advice kind:

RADC-AO can automatically derive the joinpoints and advice kind from all previously information that is collected during the analysis stage as follows:

Joinpoint: derived from the knowledge of what is the theme which is before the current theme (if the current theme not the first theme in the sequence of themes in requirements) or what is the theme after the current theme (if the current theme is the first theme in the sequence of themes in requirements) – it means that what is the theme that trigger the current aspect – this knowledge is derived from the sequence of themes in requirements, after knowing what is the theme, RADC-AO know to which class or aspect it belongs from the design information for aspects and classes. By the previous knowledge, RADC-AO can define what is the class or aspect ? and what is the operation that trigger the current operation ?.

Advice kind: if the current theme is not the first theme in the sequence of themes in the requirements graph, then the advice kind is 'after', if the current theme is the first theme in the sequence of themes in requirements then the advice kind is 'before'.

After RADC-AO determines the joinpoint and advice kind, software engineer easily can change the design of joinpoint or advice kind.

2.2.4 Stub-Code Generation

RADC-AO uses the design information got from design process to produce stub-code for the classes and aspects belonged to system that is analyzed and designed by RADC-AO.

All the previous steps included in RADC-AO tool can be abstracted in a flowchart, Figure 3 represents the flowchart of the RADC-AO steps.

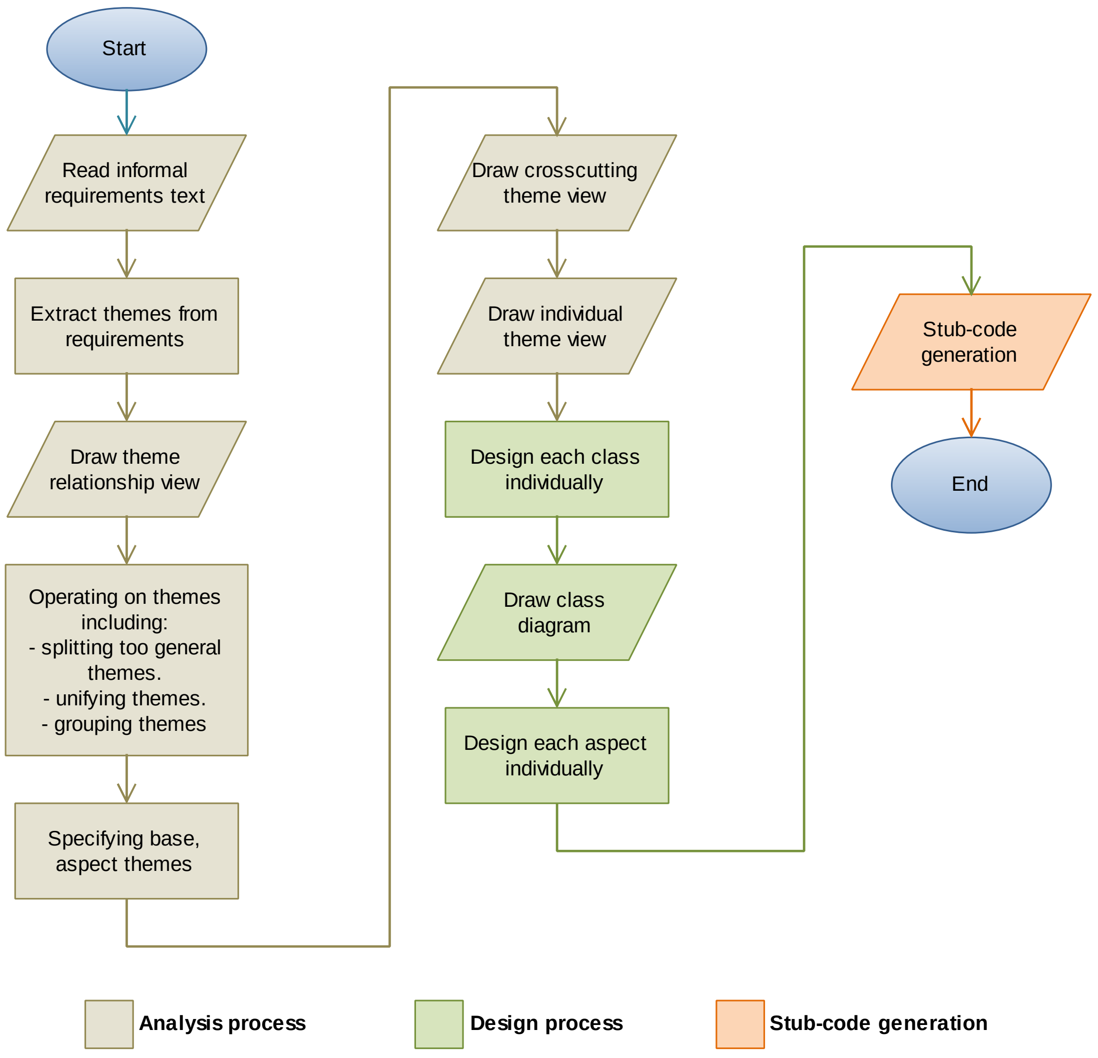

Start
Read informal requirements text
Extract themes from requirements
Draw theme relationship view
Operating on themes including:
- splitting too general themes.
- unifying themes.
- grouping themes
Specifying base, aspect themes
Draw crosscutting theme view
Draw individual theme view
Design each class individually
Draw class diagram
Design each aspect individually
Stub-code generation
End
Analysis process
Design process
Stub-code generation

Figure 3: RADC-AO tool flowchart.

3. RADC-AO Testing and Results:

RADC-AO was tested by using it to construct a payroll system. This payroll system includes the next functions:

1. Enter information about employees in the organization.
2. Update employee's information.
3. Delete employee's information.
4. Enter employee's loan information.
5. Update employee's loan information.
6. Delete employee's loan information.
7. Enter login information for each payroll system's users.
8. Calculate the payment for each employee.
9. The payroll system must automatically save each used function in the payroll system, who uses this function, what is the date and time of the use.
10. Reviewing the information collected in function 9.

When the payroll system runs, it must ask the user for his username and passwords to check them with previously saved usernames and password for all system's users. The payroll system allows two types of employees to use it (administrator and normal employee). Administrator can use all functions in the system, where normal employee can use (1, 2, 3, 4, 5, 6) functions only.

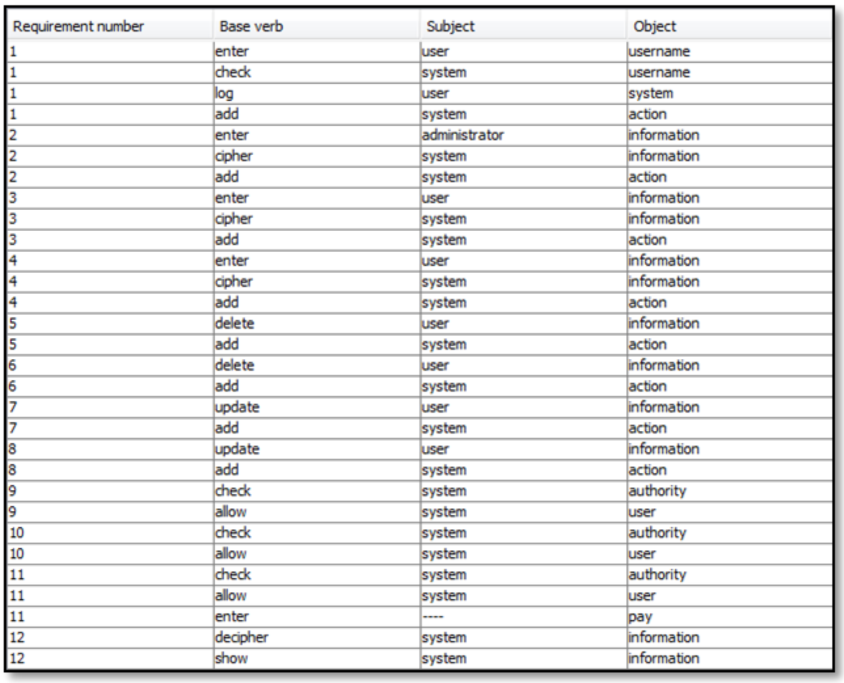
When user enters information it must be coded before it will be saved, and when system shows information to the user, it must be encoded before it will be showed.

The payroll system has a set of textual requirements that are entered to the RADC-AO. The followings are the set of requirements:

1. User must enter username, password. system must check username, password. only authorized user can log the system, system add this action to history.
2. Administrator can enter employee's information and the system must cipher this information before save the information. system add this action to history.
3. User can enter employee's pay information and the system must cipher this information before save the information. system add this action to history.
4. User can enter employee's loan information and the system must cipher this information before save the information. system add this action to history.
5. User can delete employee's pay information, system add this action to history.
6. User can delete employee's loan information, system add this action to history.
7. User can update employee's pay information, system add this action to history.
8. User can update employee's loan information, system add this action to history.
9. System must check authority before system allow user to entering to history.
10. System must check authority before system allow user to entering to team information
11. System must check authority before system allow user to entering calculate pay.
12. System must decipher all ciphered information then system shows information.

In order to complete the analysis, design and stub-code generation to the payroll system, sequential steps must be applied, these steps are explained in following paragraphs and figures:

1. Entering the payroll requirements.
2. Extracting themes from requirements. Figure 4 shows themes (base verb), its subject and object if exist in each requirement.



Requirement number	Base verb	Subject	Object
1	enter	user	username
1	check	system	username
1	log	user	system
1	add	system	action
2	enter	administrator	information
2	cipher	system	information
2	add	system	action
3	enter	user	information
3	cipher	system	information
3	add	system	action
4	enter	user	information
4	cipher	system	information
4	add	system	action
5	delete	user	information
5	add	system	action
6	delete	user	information
6	add	system	action
7	update	user	information
7	add	system	action
8	update	user	information
8	add	system	action
9	check	system	authority
9	allow	system	user
10	check	system	authority
10	allow	system	user
11	check	system	authority
11	allow	system	user
11	enter	----	pay
12	decipher	system	information
12	show	system	information

Figure 4: The Result of Extracting Themes Step.

3. Drawing theme relationship view, Figure 5 shows the theme relationship view for the payroll system, where blue nodes represent themes, red nodes represent requirements.

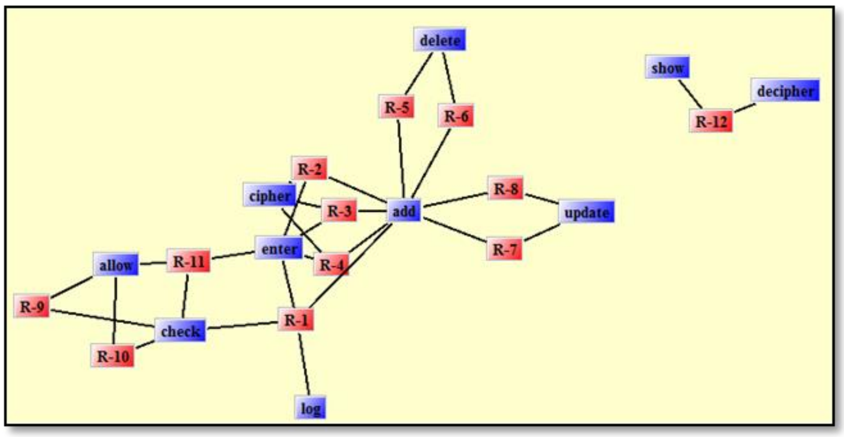


Figure 5: Theme Relationship View for the Payroll System.

4. Splitting, unifying and grouping operations on themes, Figure 6 shows the resulted relationship view after splitting, unifying and grouping operations applied on the payroll system.

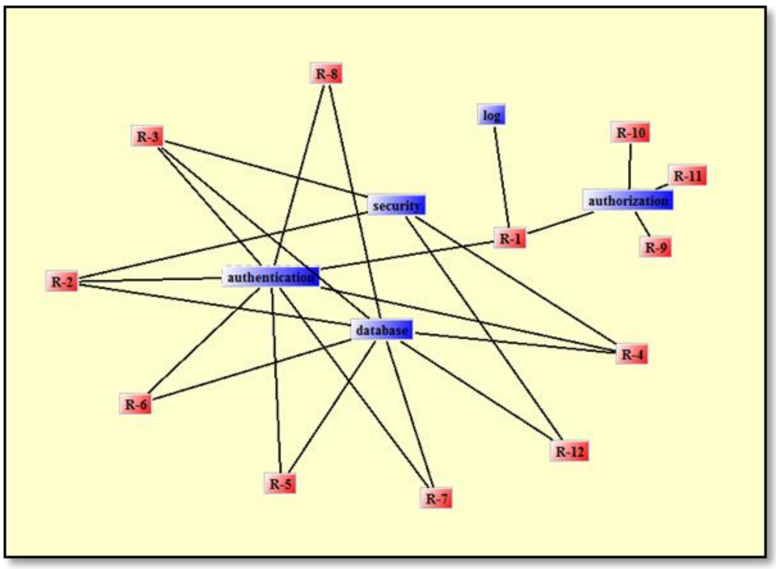


Figure 6: Theme relationship view after splitting, unifying, grouping themes view for the payroll system.

As depicted in Figure 6, the number of themes is decreased because (add, enter, update, delete) themes merged in database theme, and so on.

5. Specifying base and aspect themes by enter the (ATT) value in an input message.

6. Crosscutting theme view, Figure 7 shows a part of the crosscutting theme view.

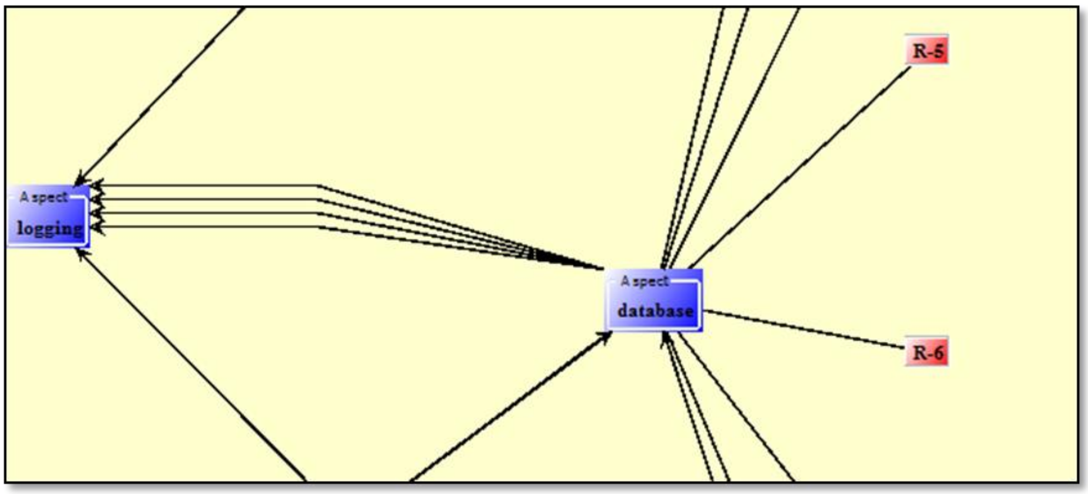


Figure 7: A Part of Crosscutting Theme View.

As depicted in Figure 7, each requirement is represented in the red node and linked to the most dominant theme for it, themes in one requirement are lined by an arrow starts with the triggering theme and ends with the triggered theme.

7. Drawing individual base themes view, Figure 8 shows a part of base themes individual view.

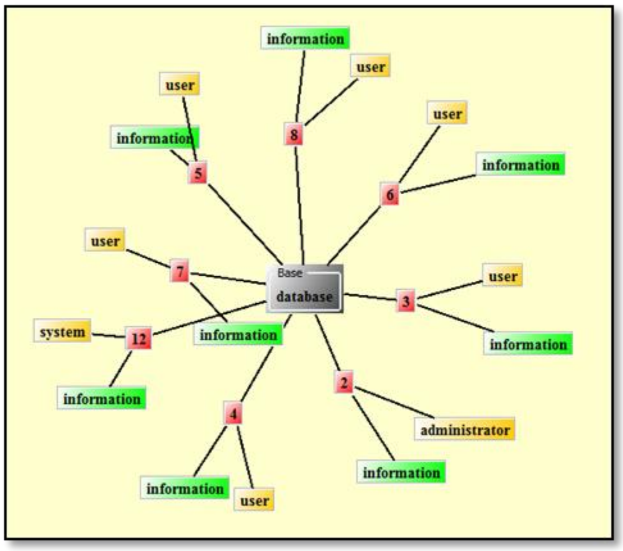


Figure 8: Part of the base themes individual view of the payroll system.

In Figure 8, the gray node represents the base theme, red nodes represent requirements that contain the base theme, yellow nodes represent the subject of that base theme and the green nodes represent the objects of that base theme.

8. Drawing aspect themes individual view, Figure 9 shows a part of aspect themes individual view.

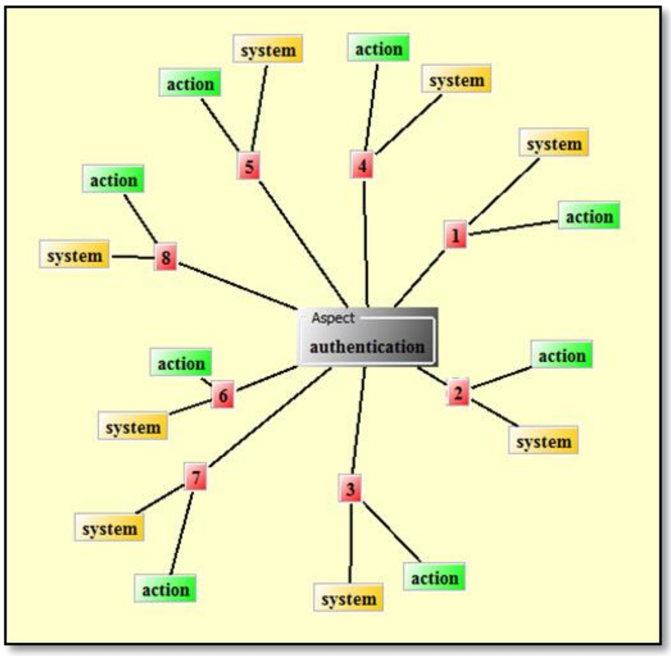


Figure 9: Part of the Aspect Themes Individual View of the Payroll System.

In Figure 9, the gray node represents the aspect theme, red nodes represent requirements that contain the aspect theme, yellow nodes represent the subject of that base theme and the green nodes represent the objects of that aspect theme.

9. Designing each class (its origin base theme) individually, figure 10 shows how each class can be designed individually, how to create new classes by new class button, rename classes, delete classes, create new attributes, delete attributes, create new operations, delete operation, managing templates library.

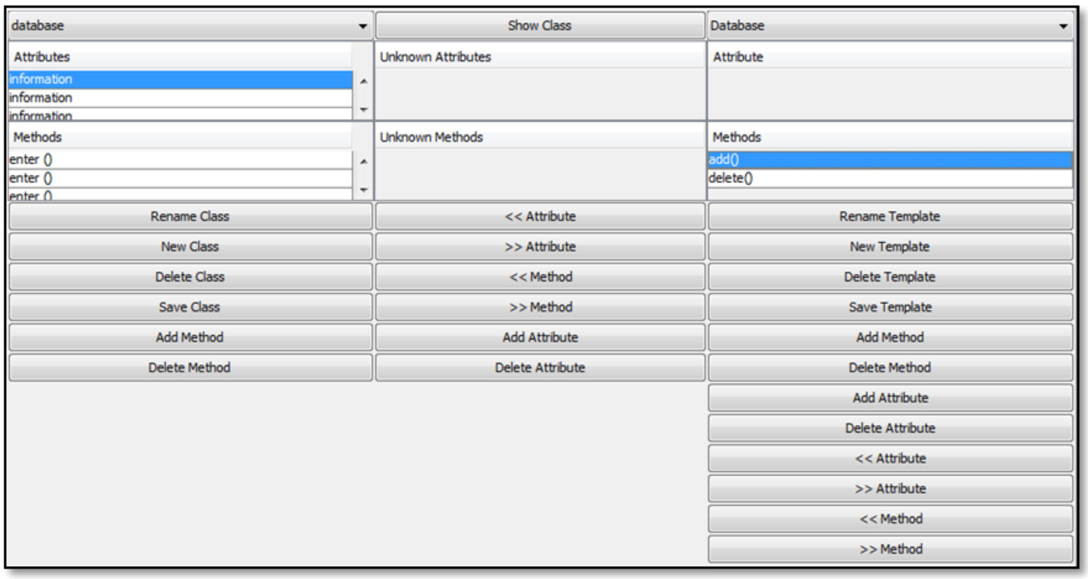


Figure 10: Design each Class Individually for the Payroll System.

10. Drawing class diagram for classes that are designed in step 9. Figure 11 shows the class diagram for the payroll system.

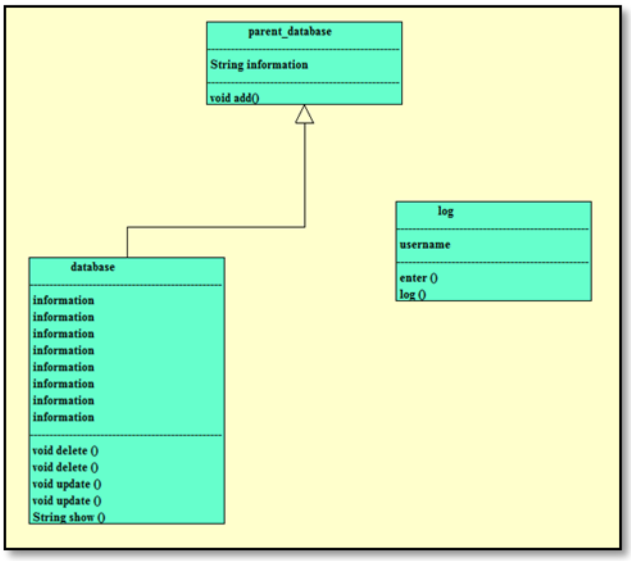
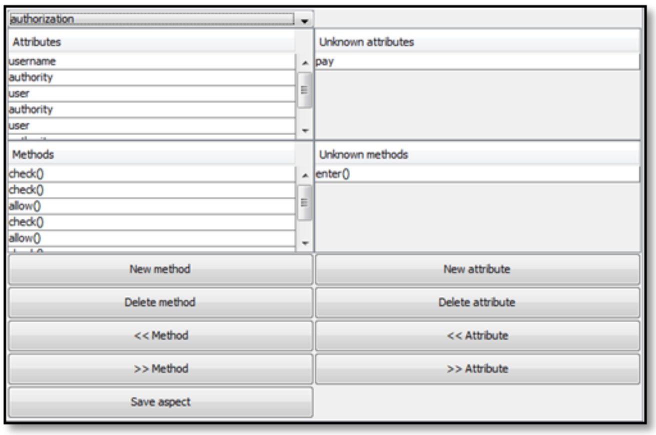


Figure 11: Payroll system class diagram.

11. Designing each Aspect Individually. Figure 12 shows how this step done.

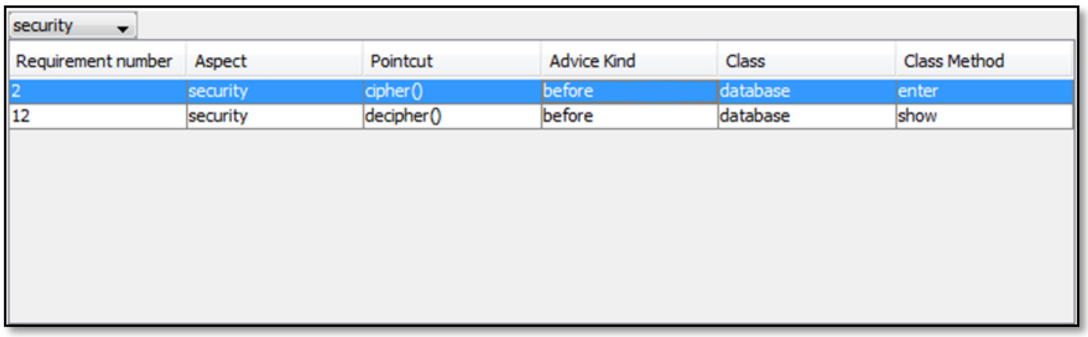

authorization

Attributes	Unknown attributes
username	pay
authority	
user	
authority	
user	

Methods	Unknown methods
check()	enter()
check()	
allow()	
check()	
allow()	

Figure 12: Design each Aspect Individually for the Payroll System.

12. Designing each aspect's adevices kind, Figure 13 shows how RADC-AO extracts the design of each aspect's adevices kind and poincuts, how software engineer can modify the extracted previous design.


security

Requirement number	Aspect	Pointcut	Advice Kind	Class	Class Method
2	security	cipher()	before	database	enter
12	security	decipher()	before	database	show

Figure 13: Joinpoints and Advices Kind Design for the Payroll System.

13. Stub-code generation for classes and aspects, next is an example of generated stub-code for log class and security aspect :

```
class log
{
String username;
void enter ()
{
}
log ()
{
}
}
 //END OF CLASS
aspect security_ASP
{
String information;
int information;
int information;
String information;
cipher(): call (database.enter(..));
after : enter
{
}
String decipher(): call (database.show(..));
before : show
{
}
}
//END OF ASPECT
```

From the previous results RADC-AO successes in the test as the followings:

- Identifying the crosscutting concerns in the requirements text by a set of (NLP) steps.
- Successfully implementing all operations on themes (splitting, unifying, and grouping).
- It is automatically extracting classes and aspect design information like operations and attributes, and enables the software engineer to dynamically complete the design like operation type, attribute type and relationship between classes.
- Successes in facilitating the generation of class diagram.
- Facilitating the programming by generate stub-code.

4. Conclusions

Through the building and testing of the RADC-AO, we conclude the following:

- It is possible and very successful to automate the disturbing and complex levels of software development process.
- It is possible and very successful to integrate the (NLP) paradigm into software engineering stages especially those that require linguistic processing like requirements.

- It is possible to produce linguistic rules that can help in software analysis and design process.
- It is possible to depend on the results produced from automated tools for their accuracy.

5. Future Work

Through the importance of software engineering in the software development it must be automated and computerized. From this point, the most important recommendations are:

- Expanding RADC-AO services domain through developing it to abstract requirements as use case diagram.
- Creating code templates library for common used codes to produce a complete code for common situations.
- Improving the analysis and design processes by making RADC-AO iterative and supply it with software metrics.
- Developing it to extract the relationship between classes and aspects automatically.
- Automating the determining dominant theme of each requirement.

REFERENCES

[1] Anne Kao, Stephen R. Poteet, (2007), "Natural Language Processing and Text Minning", chapter one, Springer, United States of America.

[2] Thomas C. Rindflesch, (1996), "Natural Language Processing", Cambridge Journals, Annual Review of Applied Linguistics, Volume 16, page 70-85.

[3] Igor Bolshakov, Alexander Gelbukh, (2004), "Computational Linguistics", chapter 3, 1st Edition, Mexico.

[4] Karl E. Wiegers, (2003), "Software Requirements", chapter one, 2nd Edition, Microsoft press, Washington.

[5] Soren Lauesen, (2002), "Software requirements Styles and techniques", chapter 1, 1st Edition, Addison Wiley, Great Britain.

[6] Robin F. Goldsmith, (2004), "Discovering real business requirements for software products success", chapter 11, 1st Edition.

[7] Ivar Jacobson, Pan-Wei Ng, (2004), "Aspect-Oriented Software Development with Use Cases", Addison Wesley Professional, United States of America.

[8] Niklas Påhlsson, (2002), "Aspect-Oriented Programming: An Introduction to Aspect-Oriented Programming and AspectJ", Topic Report for Software Engineering, Department of Technology University of Kalmar, SWEDEN.

[9] John Viega, Jeffrey Voas, (2000), "Can Aspect-Oriented Programming Lead to More Reliable Software ?", IEEE Journal, Volume 17, Issue 6, pages: 19-21.

[10] Stefan Hanenberg, Sebastian Kleinschmager, Manuel Josupeit-Walter, (2009), "Does Aspect-Oriented Programming Increase the Development Speed for Crosscutting Code ? An Empirical Study", 3rd International Symposium on Empirical Software Engineering and Measurement.

[11] Filho, F., Rubira, C., Garcia, A., (2005), "A Quantitative Study on the Aspectization of Exception Handling", Workshop on Exception Handling in OO Systems (held with ECOOP), Glasgow, Scotland.

[12] Sasa Subotic, Judith Bishop, Stefan Gruner, (2006), "Aspect-Oriented Programming for a Distributed Framework", South African Computer Journal, Suid Afrikaanse Rekenaar Tydskrif 37, pp.81-89.

[13] Ruzanna Chitchyan, Awais Rashid, Pete Sawyer, Alessandro Garcia, Mónica Pinto Alarcon, Jethro Bakker, Bedir Tekinerdogan, Siobhán Clarke, Andrew Jackson, (2005), "Survey of Analysis and Design Approaches", chapter 3, AOSD Europe.

[14] Elisa Baniassad, Siobhan Clarke, (2004), "Theme: An Approach for Aspect-Oriented Analysis and Design", Proceedings of the 26th International Conference on Software Engineering.

[15] Ali, B. S., Kasirum, Z. M., (2008), "3CI : A Tool for Crosscutting Concern Identification", International Conference on Computational Intelligence for Modeling Control & Automation.

[16] Busyairah Syd Ali, Zarinah Mohd. Kasirun, (2008), "Crosscutting Concern Identification at Requirements Level", Malaysian Journal of Computer Science.

[17] Yan Wu, Mansour Zand, Harvey Siy, Victor Winter, (2006), "Systematic text-mining approach for deriving aspects and patterns from domain knowledge", ICSE 2006 Workshop on Early Aspects, Shanghai, China.

[18] Américo Sampaio, Neil Loughran, Awais Rashid and Paul Rayson, (2005), "Minning Aspects in Requirements", Computing Department, Lancaster University, Lancaster, UK.