A Probability Algorithm for Requirement Selection In Component-Based Software Development

Ruba Alzyoudi
Master Student
Department of Software Engineering
Hashemite University
P.O. Box 330136 Zarqa (13115)
Jordan
ruba.alzyoudi@hu.edu.jo

Khaled Almakadmeh
Assistant Professor
Department of Software Engineering
Hashemite University
P.O. Box 330136 Zarqa (13115)
Jordan
khaled.almakadmeh@hu.edu.jo

Hutaf Natoureah
Lecturer
Department of Computer Science
Hashemite University
P.O. Box 330136 Zarqa (13115)
Jordan
hutaf@hu.edu.jo

ABSTRACT

Nowadays highly competition business environment, customer increased expectations, and highly advances in computer technologies and software leads many organizations to adopt Component Based Software Development (CBSD) approach in developing their systems. As CBSD apply the idea of Components On The Shelf (COTS) that looks for creating, using, and reusing previously used component, CBSD expected to result on faster software development which entails shortest time to market and products of higher quality.

In CBSD, there still complexity regarding selecting the appropriate requirements for the components, and further deciding which component to be delivered first to the customer. From the fact that many approaches are presented in the literature to solve this problem but still there is some angles should be covered, in this paper we presented an algorithm to facilitate the process the prioritizing functional requirements in the incremental software development model depending on the dependency relationship between requirements.

Keywords

Component based development, functional requirements, dependency, algorithm**,** incremental CBSD

1. INTRODUCTION

Recently, a demand for large-scale, complex and cost-effective systems increased, these requirements face several challenges concerning productivity cost and time, and sometimes unmanageable software quality.

Many researches contribute in finding new, efficient, easily managed, and cost effective paradigm for software development. CBSD is an approach to software development that relies on the reuse of existing software components to reduce the development costs and production cycle, while increasing the final product's quality.

A software component is defined as a unit of composition with contractually specified interfaces and explicit context dependencies [1]. Software components must be identified and evaluated in order to determine if they provide required functionality for systems being developed [2]. Domain Engineering (DE) is a process in which the reusable component is developed and organized and in which the architecture meeting the requirements of this domain is designed [3].

Component-based software engineering (CBSE) is a process that emphasizes the design and construction of computer-based systems using reusable software "components." based on the idea to develop software systems by selecting appropriate off-the-shelf components and then to assemble them with a well-defined software architecture [5].

Despite the difficulties in producing generic, scalable, adaptable, and reusable components, CBSD opened the door for companies to achieve high competitively by fast delivering of software in and incremental-based way. Where software components are design and reused in many other software and incremental way of delivering the system to the customer will make this process effective while reducing the time-to-market and increasing the productivity by developing the software product by using already coded and tested modules.

Building components and deciding on increments delivering priorities are two major challenges facing the CBSD approach, since many considerations should be taken into account such as a software component is defined as a unit of composition with contractually specified interfaces and explicit context dependencies [1]. Every day we make many decisions like whether to take this bus or the next one, even with we have just a couple of choices, decisions can be difficult to make. But what will happen when having tens, hundreds or even thousands of functions to use or alternatives, decision-making becomes much more difficult. One of the keys to making the right decision is to prioritize between different alternatives. It is often not obvious which choice is better, because several aspects must be taken into consideration We need to develop the functionality that is most desired by the customers, as well as least risky, least costly, and so forth. Prioritization helps to cope with these complex decision problems.

IPAC '15, November 23-25, 2015, Batna, Algeria

DOI: http://dx.doi.org/10.1145/2816839.2816903

In this research, we discuss three factors which may affect the selection process but we focus on the dependency factor because The dependencies between individual requirements have the most important influence on selecting the functional requirements which effected on many software engineering activities e.g., project planning, architecture design, and change impact analysis. Further, we suggest an algorithm to enhance the prioritizing functional requirements selection process while using and incremental component CBSD approach depending on the dependency relationship between requirements. In order to test the presented algorithm, an example of software requirements system is used.

The rest of the paper is organized as follows: Section 2 presents a review of the literature; Section 3 presents the research methodology. Finally, section 4 presents conclusion of findings and discussion.

2. Review of the literature

Huge research done by software engineering researches on issues related to software development methodologies and several of them concerned with incremental CBSD.

Off-The-Shelf Option (OTSO) is an approach presented by Kontio. This approach depends on six major phases: "searching" for components that mostly satisfies system requirements and constraints, then "screening" to pass the best components to the evaluation phase. In the "evaluation" phase, the component is judged against the functional requirements, correctness, and software architecture and business concerns: analysis, deployment, and assessment phases are then followed.

Kontio[4] described the characteristics of some selected state of art CBSD models that are practiced in software industries: they proposed a complete model for Component Based Software Development for reuse. The main phases are feasibility study; system requirement and analysis; system design, component identification and adaption, component integration engineering, system testing and system release and deployment [6].

Morisio & Tsoukis[7] proposed to address the quality requirement during the evaluation process to formalize the component selection process. They proposed IusWare (IUStitia softWARis) approach which is based on Multi-criteria Decision Aid (MCDA). The state of art and transfer of component technology from engineering concept to software concept is presented they have followed CBD-Arch-DE process which is considered as a better approach among the both CBSD and Domain Engineering [3].

In [2], researchers considered how the Common Criteria (CC), an internationally recognized standard for security requirements definition and security assessment of IT systems can be applied towards the development of component-based systems. The process includes six steps: system high level design, component requirements definition, component search, component evaluation, component selection, and component integration and operation.

A discussion about the selection techniques for components is presented by [8]: they described the component selection techniques that help to select the components which can satisfy the requirements. The cluster based component selection process, for example, consists of 3 stages: dependency analysis between Concrete Level Goals (CLG), goal-oriented specification, and cluster analysis, goal oriented.

After defining the meaning of the term “priority”, the purpose and benefits of requirements prioritization are listed by Donald Firesmith [12]. This is followed by a concise discussion of the challenges and risks that a requirements team must face when prioritizing requirements. Then, various techniques for prioritizing requirements are identified, and finally a set of recommendations (including a recommended prioritization process) are made.

Patrik and Anneliese [16] provided an overview of techniques for prioritization of requirements for software products. Methods are given about how to combine individual prioritizations based on overall objectives and constraints and how to approach a prioritization situation.

Koziolek [14] suggest an approach to generate feedback from quantitative architecture evaluation to requirements engineering, in particular to requirements prioritization. Koziolek proposed to use automated design space exploration techniques to generate information about available trade-offs.and described application scenarios.

The dependency model proposed by Pohl [18] was based on a survey of over thirty publications in the area of requirements engineering. The other is a requirement (inter-)dependency model proposed by Dahlstedt and Persson [19]. The dependency types evaluated in this study come from these two well-known dependency models this study provides a concise overview of the seven inter-dependency types suggested in the D-model.

A family of test case prioritization techniques is presented by Parthiban et. al. [15] using the dependency information from a test suite to test suite that priority. Zhang Zhang [13] suggested an approach for specifying functional requirements dependency. Zhang generalized a classification of functional requirements dependency and proposed a process meta-model to specify the semantic information of functional requirements dependency and deploy it on a wiki platform named Semantic REWiki. In this paper, we use a set of functional requirements which are taken from an online shopping system applied by Zhang Zhang [13].

3. Research Methodology

3.1 Research Method

In order to generate a component selection measurement, we start by analyzing the factors that may affect or affected by our components functionalities selection. Since deploying the high priority increment first will affect the overall throughput of the software. these factors are :Stakeholders factor, Risk factor, and finally compatibility and dependability factors Figure 1 below describes the relationship between these parties.

In this paper, we focus on the dependency factor since dependencies between individual requirements have the most important influence on selecting the functional requirements so we aimed to present an algorithm to facilitate the process the prioritizing functional requirements in the incremental software

development model depending on the dependency relationship between requirements.

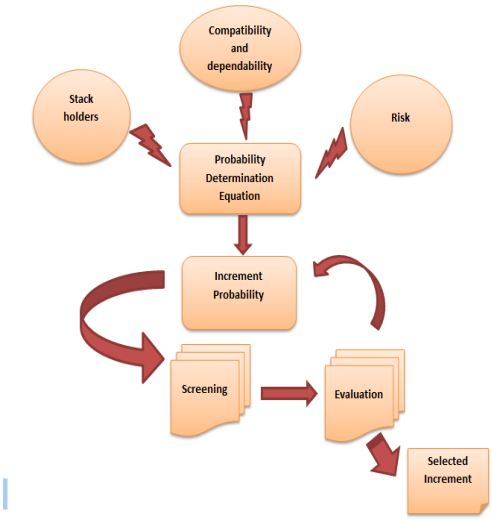

Figure1. Increment functions selection procedure

3.2 Stakeholder's factor

Stakeholders are the most important asset in determining the core functions of the system and thus they are the assistant in increments selection process.

Stakeholders includes several parties, major stakeholders in process of CBSD are: Component developers, application assemblers who are responsible for locating suitable components and assemble them in integrated application systems that satisfy customer requirements and Customers [9].

Several methods used in order to prioritize requirements according to stakeholders' opinion. The ten most important; this was done with a simple 1 to 10 ranking method, in which one (1) being “not important” and ten (10) “very important”. Based on the elicitation meetings and the perceived ideas of what was important to the different stakeholders; a number was set for each requirement. Other requirements could be prioritized according to the “Five-Way Priority Scheme” or other methods for prioritization, such as the "hundred-dollar test" and the "Yes/No" vote. Selection of the most appropriate method will depend on a weighing scheme for the disadvantages and advantages of these methods against each other.

3.3 Risk factor

While CBSD helps overcome inadequacies in traditional development, it also poses risks to the profitability and even long-term survival of each of its stakeholders. From uncertainties in leveraging existing legacy code to the inability to find needed components, they confront challenges in constructing component solutions that address their evolving enterprise requirements. Therefore, before embarking on component-based development projects, each stakeholder must assess its risks and devise sound strategies to address them [9].

3.4 Compatibility and Dependency

Compatibility factor describes the how the components can operate satisfactory together on the same system. consistency between the numbers and types of method arguments and on appropriate use of a method return type by determining those interfaces which can satisfy all possible sequences of requested operations.
Compatibility assessment can help determine whether a pre-existing software entity can be reused in a particular environment [10]. Dependency on the other hand, Component dependency analysis is crucial to effective maintenance, evolution, testing, debugging, and management of component based systems.

Considering a system S to be built, the system has a set of defined functional requirements R1, R2…Rn where n is the number of functional requirements.

If component based development model is applied it will be a challenge to find the best increment specifications and the optimal order at which these increments will be implemented.

Assuming that the software production process Spp is based on a set of well defined user requirements and by analyzing dependencies between requirements; our approach aims to prioritize requirements as components in order to enhance the process of delivering the system to the customer. Functions delivered by a system have many relationships between them. For example, as defined by Pohl [18], all types of dependencies between functions goes under five main types: the condition dependency, content dependency, document, evolutionary and abstraction dependency. Dahlstedt [19] defined three types of dependency: structure, constrained, and cost/value dependency.

Zhang Zhang [13] suggested a dependency classification for functional requirements; generally the FRDs can be divided into three categories: Structural Dependency, Constraint dependency and Operational Dependency. From these models, we defined three main relations from which we can order and prioritize the requirements in an efficient and valuable manner, these functions are : support(Ri,Rj), Contradict(Ri,Rj), and Before(Ri,Rj).

- Support(Ri,Rj) relation means that the achievement of requirement Ri is required in order to achieve the requirement Rj, since Ri give a type to support to Rj such as (input/output) or (based_on) or (pre-condition).

- Contradict(Ri,Rj): means that the requirement Ri has some conflict with requirement Rj, thus they could not applied at the same time.

- Before(Ri,Rj): means that the implementation of Rj as soon as possible after Ri will result in an advantage such as reduction of the total implementation cost or adding some value to whole software development process.

3.5 The Requirements Prioritizing Algorithm

Depending on previous three relations, we present an algorithm to prioritize requirements and generate the increments in incremental software development. The algorithm as follows:

Step 1:
Dfine the type of relation between each pair of functioanl requirments, such that:

- the relation Support(Ri,Rj) equlas 1 if Rj could not be implemented unless Ri is already done because there is a type of input/output or inheriance relationships for example.
- the relation Before(Ri,Rj) will be evaluated to 1, if Rj not dependent on Ri but if we implement it as soon as aposible after Ri this will add some value or decrease some costs.
- the relation Confict (Ri,Rj) will be evaluated to 1 if the two requiremnts Ri and Rj could not exist at the same time.

Step 2:
Create a two dimensional matrix where each row represent a functional requirement and the rows represent the same series of requirements. Fill the Reltionship Matrix by the values (S, B, or C) to indicate the type of relation between each pair of requirements, or a dash symbol (-) to indicate that the two requirements doen not have any direct relation.

Step 3:
Generate A directed Graph from the Support relationships:
- generate a strat node
- for each requirements in the columns that have no support relationship in its entire rows, generate a new child node from the start node (level 1)
- for each node in level 1,search for any support relationship in its row and create a directed arc from this node to the node which it supports

Step 4:
Apply the Before relationship:
- assign each node a sequence number following depth first search path.
- aor each functional requirement, if its row has a Before relationship with other requirement, be sure that the serial number of this requirement is less than the serial number for the other one.
- if the serial numbers are not correct in satisfying the before relationship, then swap the place of the two branches containing these requirements.
- re-order the nodes serial numbers to reflect the change done.

Step 5:
Apply the Conflict relationship:
- cut the graph into pieces (increments) such that no two conflict requirements exist in the same piece of the graph.
- remove any redundant nodes because the requirement should be implanted once over all the system.
- each piece will represent an increment in the software development process
- assgin the increments serial numbers according to their order in graph (depth-first)

3.6 Example of Software System

In order to test the presented algorithm, a proper example of software system is required. We use a set of functional requirements which is taken from an online shopping system applied by Zhang [13] in his paper. Table 1 lists the FRs identified from the online shopping system, which can be used as examples to present the FRD.

Table 1. Functional Requirements for the online shopping system

ID	Name	Description
REQ 2.1	Display Product	The system displays all the products to the customer
REQ 2.2	Select Product	The system allows the customer to select a product from the product list
REQ 2.3	Define amount	The system queries the customer for the amount of selected product
REQ 2.4	Define color	The system queries the customer for the color of selected product
REQ 2.5	Add Product	The system puts the product which the customer selected to his shopping cart
REQ 3.1	Update amount	The system queries the customer to update amount of selected product
REQ 3.2	Update color	The system queries the customer to update color of selected product
REQ 4.1	Display order	The system displays the detail information of the order to the customer
REQ 5.1	Confirm order	The system prompt the client to confirm the acceptance of the order
REQ 6.1	Send mail	The system sends an information mail to the customer's email address
REQ 7.1	Display notice	The system displays a successfully notices to the customer in order to inform the customer that the order has been accepted

Table 2. The filled matrix

	Req 2.1	Req 2.2	Req 2.3	Req 2.4	Req 2.5	Req 3.1	Req 3.2	Req 4.1	Req 5.1	Req 6.1	Req 7.1
Req 2.1	----	S	B	B	B	B	----	C	----	----	----
Req 2.2	----	----	S	S	S	S	S	C	----	----	----
Req 2.3	----	----	----	----	B	S	----	S	B	----	----
Req 2.4	----	----	----	----	S	S	B	C	----	----	----
Req 2.5	----	----	----	----	----	B	B	C	----	----	----
Req 3.1	----	----	----	----	----	----	----	S	B	----	----
Req 3.2	----	----	----	----	----	S	----	B	----	----	----
Req 4.1	C	C	C	C	C	----	----	----	B	B	B
Req 5.1	C	C	C	C	C	C	C	----	----	S	B
Req 6.1	C	C	C	C	C	C	C	C	----	----	B
Req 7.1	C	C	C	C	C	C	C	C	----	----	----

To apply the algorithm, we need to generate a two dimensional matrix where columns and rows represents the same set of functional requirements. And then fill the matrix by symbols (S, C, or B) to indicate the type of relationship between each pair of requirements. (S refers to Support relationship, B refers to the Before relationship, and C refers to the Contradict relationship). Table 2 shows the filled matrix.

From the matrix above, for example we see that define amount requirement (Req2.3) will support display order requirement (Req4.1) since we need the result of Req2.3 to start the Req4.1. While Req2.3 assumed to come before Add product requirement (Req2.5) since adding product has not input/output or inheritance relation with Req2.3, but it is expected to require adding some product after computing the amount for the previous added one. Requirement Define Color (Req2.4) has a contradict relation with the requirement Display Order (Req4.1) since it is not logical to do the same operations at the same time, because displaying the order stopped any update operation. Next, we need to generate a directed graph from the support relationships in table 2. Figure 2 below shows the support direct graph; note that Req2.1 for example has an outcome directed arc to Req2.2 because Req2.1 supports Req2.2 (as analyzed in the relationship matrix).

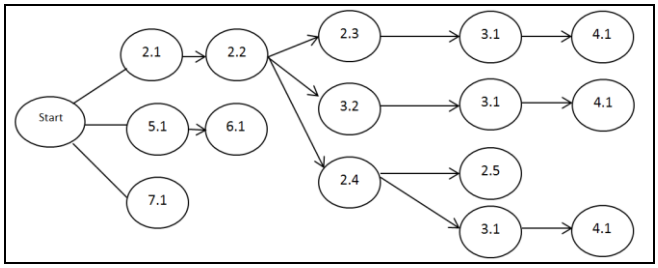

Figure 2: support relationship directed graph

To apply the Before relationship, we start by giving serial numbers to each node starting from the start node and flowing depth first search.(note: we start from the right because we add the nodes to the graph starting from the right branch) See Figure 3.

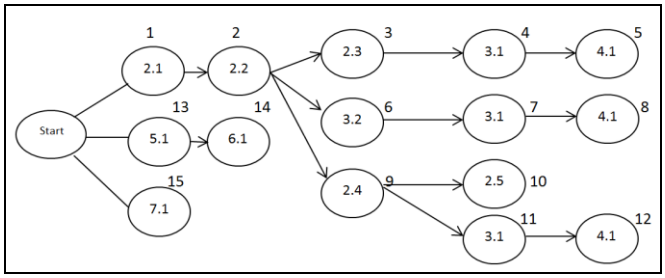

Figure 3: Assigning serial numbers to graph nodes

Looking at the matrix we note that, for example, Req2.4 expected to be implemented before Req3.2 while Req2.4 has serial number 9 and Req3.2 has serial number 6!! Thus we shall swap the two branches as shown in figure 4.

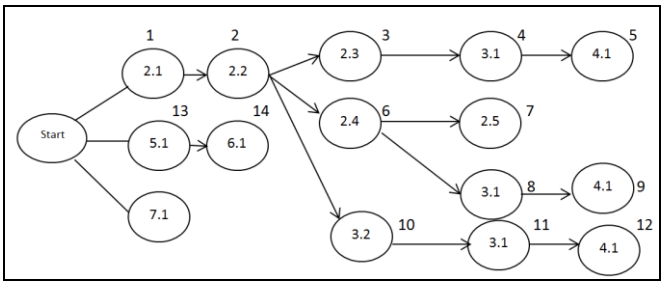

Figure 4: The graph after swapping two branches

We will note also that Req2.5 need to become before Req5.1, another swapping happened in Figure 5 to represent the relation

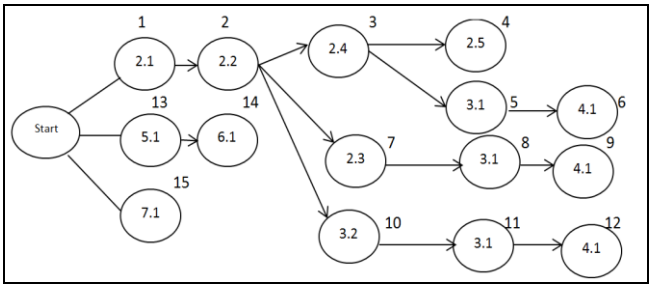

Figure 5: The graph after applying the Before relationship

Now, the last step is to cut the graph into small pieces to represent the increments. The cut operation depends on the contradict relationship such that no tow contradicted requirements could exist in the same increment. Because 2.1 and 2.2has conflict with 4.1, and 5.1 conflict with 2.3 and 3.2, three increments defined as shown in figure 6.

Note: a small branch is deleted since it is already exist in increment2 and could not be implemented until implementing Req2.3 and Req3.2.

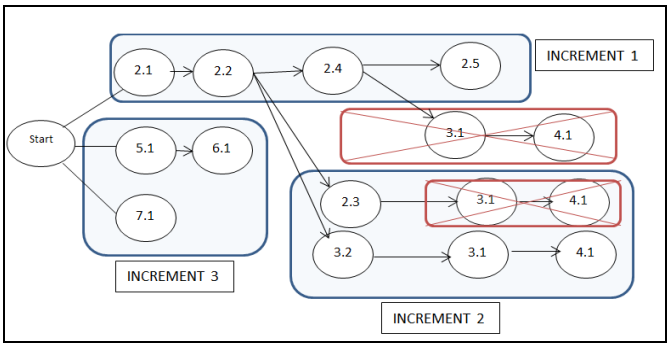

Figure 6: The Defined Increments

As a result from this algorithm, we conclude that this system could be implemented in an incremental model as three increments:

- Increment 1: Display Products, Select Products, Define Color, Add Product
- Increment 2: Define Amount, Update Color, Update Amount, Display Order
- Increment 3: Confirm Order, Send Mail, Display Notice.

4. Conclusion

This paper presented an analysis of factors that affect the increment selection process in the incremental CBSD. Our study considers three main factors affect increment selection, namely; stakeholder's opinion, the expected risk percentage of reuse, and increments dependency. This paper presented in particular algorithm to analyze increments dependency to facilitate the process of prioritizing requirements and increments selection in an incremental software development process. We used an example of online shopping system requirements as proof of concept. Future work will be devoted to test the proposed algorithm with several case studies.

5. References

[1] Jeetendra Pande, "On Some Critical Issues in Component Selection in Component based Software Development", International Journal of Computer Applications (0975 – 8887) Volume 46– No.4, May 2012.

[2] Wes J. Lloyd, "A Common Criteria Based Approach for COTS Component Selection", Published by ETH Zurich, Chair of Software Engineering JOT, 2005.

[3] N.Md Jubair Basha, Salman Abdul Moiz," Component Based Software Development: A State of Art", IEEE-International Conference on Advances In Engineering, Science And Management, March 30, 31, 2012.

[4] J. Kontio, "OTSO: A Systematic Process for Reusable Software Component Selection" University of Maryland, Maryland, CSTR- 3478, December 1995

[5] DEBAYAN BOSE,"Component Based Development", Indian Statistical Institute, 2010.

[6] Asif Irshad Khan, Noor-ul-Qayyum, Usman Ali Khan, "An Improved Model for Component Based Software Development", Software Engineering 2012, 2(4): 138-146,

[7] M. Morisio and A. Tsoukis, "IusWare: a methodology for the evaluation and selection of software products," IEE Software Engineering vol. 144 (3), June 1997

[8] Nitish Madaan and Jagdeep Kaur, "A Survey on Selection Techniques of Component Based Software", International Journal of Information & Computation Technology. ISSN 0974-2239 Volume 4, pp. 1245-1250, Number 13 2014

[9] Padmal Vitharana, "Risks And Challenges Of Component-Based Software Development", Communications Of The ACM August 2003/Vol. 46, No. 8.

[10] Donald C. Craig, "Compatibility of Software Components Modeling and Verification", phd thesis, Department of Computer Science, Memorial University of Newfoundland, May 2007.

[11] Cleidson de Souza, Erik Trainer, Stephen Quirk, David Redmiles, "Exploiting the Relationship between Software Dependencies and Coordination through Visualization", Conference'08, ACM, 2008.

[12] Donald Firesmith, "Prioritizing Requirements"Journal Of Object Technology Published by ETH Zurich, Chair of Software Engineering, 2004 Vol.3, No.8, September-October 2004, http://www.jot.fm.

[13] Zhang Zhang, "Specifying Functional Requirements Dependency in the REWiki", M.Sc. thesis, University of Tampere School of Information Sciences, Computer Science, 53 pages, June 2013.

[14] Anne Koziolek, " Research Preview: Prioritizing Quality Requirements based on Software Architecture Evaluation Feedback", Department of Informatics, University of Zurich, Switzerland, Publisher Springer-Verlag Berlin Heidelberg, 2012.

[15] T.Dinesh Parthiban, Mr.R.Kamalraj, Dr.S.Karthik," Establishing a Test Case Prioritization Technique Using Dependency Estimation of Functional Requirement", International Conference on Engineering Technology and Science-(ICETS'14), Vol 3,10,11, February 2014

[16] Patrik Berander, Anneliese Andrews, "Requirements Prioritization" , in Engineering and Managing Software Requirements, edited by A. Aurum and C. Wohlin, Springer Verlag, 2005

[17] He Zhang_,a, Juan Lic, et. al.," Investigating Dependencies in Software Requirements for Change Propagation Analysis" Information and Software Technology June 17, 2013

[18] K. Pohl, "Process-Centered Requirements Engineering", Advanced Software Development Series, Research Studies Press, 1996.

[19] A. G. Dahlstedt , A. Persson, " Engineering and Managing Software Requirements, chapter Requirements Interdependencies: State of the Art and Future Challenges" Springer, 2005.