2009 16th Asia-Pacific Software Engineering Conference

Incomplete Software Requirements and Assumptions Made by Software Engineers

Özlem Albayrak
Department. of Computer Technology and Information Systems, Bilkent University,
06800 Bilkent, Ankara, Turkey
ozlemal@bilkent.edu.tr

Hülya Kurtoğlu[1], Mert Bıçakçı[2]
Department. of Software Engineering,
STM Defence Technologies Engineering Inc.
06800 Bilkent, Ankara, Turkey
[1] hbozkurt@stm.com.tr, [2] mbicakci@stm.com.tr

Abstract—**Many software engineers make implicit assumptions when working with incomplete software requirements. To study assumptions made by software engineers while converting incomplete requirements to software design or to implementation phase deliverables, we conducted an experiment with 251 software engineers from eight companies. The results of this empirical study showed that how software engineers responded (using source code, pseudo code, or prototype) to an incomplete requirement significantly impacted the number of explicit assumptions they made. We studied relationships between the number of explicit assumptions and the engineers' experience and educational backgrounds. On average, non-computer-background engineers made more explicit assumptions than computer-background graduates. We found a significant relationship between the engineers' experience and the number of explicit assumptions made. We discuss the results and their implications.**

Keywords-incomplete software requirements;assumption

I. Introduction

In principle, a system's functional software requirements specification (SRS) should be both complete and consistent. However, in practice, for large and complex systems, it is impossible to achieve consistent and complete requirements [1]. Poor and incomplete SRS and inadequate requirements management are among the main reasons for project failure [2, 3, 4, 5, 6, 7, 8]. Poor requirements, even well managed, cause projects to fail [9]. Previous empirical studies conducted over a variety of software projects revealed that inadequate, inconsistent, incomplete, or ambiguous requirements are numerous and have a critical impact on the quality of the resulting software [10]. Empirical studies have shown that half of the errors identified at the development stage are due to inaccurate and incomplete requirements [11, 12] and early user involvement is related to better requirements quality [13]. According to some studies, incomplete requirements are the single largest cause of software project failure [5, 14].

In agile development, creating complete and consistent requirements documents is seen as infeasible or, at least not cost effective [15]. Agile requirements engineering is more dynamic and adaptive than following a formal procedure to produce a complete specification that accurately describes the system [16]. While not all software projects are suitable for agile development, SRS are critical for the success of most software projects [17]. An SRS is complete only if it includes all significant requirements, whether they relate to functionality, performance, design constraints, attributes, or external interfaces.

In real life, not all software requirements are complete and most software engineers proceed to develop software even when they face low-quality or incomplete software requirements. Previous research has studied ways to perform successful requirements engineering activities and has also studied their relationship with other processes [18, 19, 20]. The ultimate goal is to avoid generating incomplete, low-quality software requirements. If we cannot avoid such requirements, we should definitely avoid accepting them as complete, using implicit assumptions.

According to our literature survey, this study is the first one that attempts to quantify how software engineers treat incomplete requirements. It studies the relationship between software engineers' tendencies to make explicit assumptions and their preferences on how they proceed to design or implementation phases (using source code, pseudo code, or prototype). The study is composed of two parts: a quantitative experiment and qualitative post-interviews.

The remainder of this paper is organized as follows: Section 2 provides background information regarding incomplete requirements and implicit assumptions. Section 3 presents the experimental design, including the research questions, hypotheses, the subjects, and the variables. The results of the study are explained in Section 4. In Section 5, we discuss some possible threats to the validity of the study. We discuss results of the study in Section 6. Section 7 concludes the paper and presents proposals for future work.

II. Background

Requirements engineering (RE) is the process by which the requirements are determined [21]. RE process-improvement methods typically work with explicit process models with explicit document definitions [22]. The best way to develop a high-quality software system with minimal effort is to capture the requirements correctly the first time [17]. Without a well-written requirements specification, there is no way to validate that the system meets users' original needs [5]. Thus, it is highly recommended that an SRS be unambiguous and complete.

Previous studies in RE on how to avoid incomplete requirements suggest conceptual models for incomplete requirement descriptions, and frameworks merging incomplete and inconsistent views [23, 24, 25, 26]. Missing requirements discovery and taming ambiguity in natural language requirements were studied [27, 28]. Despite these efforts, software engineers are still faced with incomplete requirements. Requirements are not fully collected, in part due to the lack of a formal process or structure to support the analyst in eliciting all the available information [29].

In this study, we analyzed if the engineers' preferred way to deal with an incomplete software requirement is related to the number of assumptions they make explicitly.

In the literature, it is acknowledged that the requirements should be explicitly elicited, negotiated and documented, and then followed through in design and implementation [30]. We call the missing information between the incomplete software requirement and the complete software requirement the 'requirement gap.' A requirement gap can be filled by: information retrieved from the stakeholder, explicit or implicit assumptions made by the software engineers.

The first way is the best. Implicit assumptions should be avoided. When engineers make assumptions explicitly, they are aware of which gap they fill and how they fill the gap. As a result, they can share this explicit or recorded information with the stakeholders. In the case of implicit assumptions, engineers perceive the requirement as complete and continue the software development with their perceived requirements. When software engineers fill the gaps with information not shared and, hence, not confirmed by the user, they have made implicit assumptions, which may be the primary source of many user change requests, reworks, validity problems, and even project failures. Recent research studied various factors related to the assumptions made by the engineers [30,31,32].

We studied the impact of the engineers' working experience on the number of explicit assumptions made by the engineers. Up to 60% of individuals employed in the computer industry do not have computer-related education [33]. This figure motivated us to better understand the impact of educational background on the type of assumptions made in the case of incomplete requirements.

III. EXPERIMENT DESCRIPTION

A. Research Questions and Hypotheses

The primary research question that motivated this study was:

1. Do the ways software engineers respond to incomplete requirements impact the number of explicit assumptions they make while attempting to complete the gaps in the requirements?

The secondary research question was:

2. Do working experience and educational background of the software engineers impact the number of implicit assumptions made by the software engineers?

To investigate these research questions, a more-detailed set of three hypotheses was defined.

- $H1_a$: The number of explicit assumptions made by software engineers is affected by the engineers' preferred way to respond (code, pseudo code, or prototype).
- $H2_a$: The number of explicit assumptions made by software engineers is affected by the engineers' working experience.
- $H3_a$: The number of explicit assumptions made by software engineers is affected by the engineers' educational background (computer related or non-computer related).

B. Variables

There were three independent variables measured to determine their impact on the one dependent variable. Independent variables are the engineers':

- Preferred way to respond to an incomplete requirement,
- Working experience (number of years worked)
- Educational background (whether computer related or non-computer related).

The dependent variable is the number of implicit assumptions made by the software engineer while responding to the given requirement.

C. Design

1) Subjects: We formed a convenience sample composed of mostly CMMI Level 3 companies, one company was CMMI Level 5. We collected data from a total of 251 software engineers, 8 companies, and 39 projects. All subjects were from the same country, Turkey.

TABLE I. BACKGROUND AND EXPERIENCE DETAILS

Educational Background			**#**	**avg$_{exp}$**
Computer Related			143	
	CS	134		4.7
	CTIS	6		1.8
	CTP	2		2.5
	SwE	1		6
Others			75	
	Electrical & Electronics Eng.	56		5
	Mathematics	5		7
	Physics	4		8.5
	Electronic Communication	3		7
	Statistics	3		4.5
	Aerospace	1		8
	Economics	1		17
	Mechanics	1		10
	Nuclear	1		21
	Total		218	

CS: Computer Science, SwE: Software Engineering
CTIS: Computer Technology and Information Systems
CTP: Computer Technology and Programming
avg$_{exp}$: average working years' experience

Prior to our survey, we had conducted interviews with the software development directors of the companies. During the interviews, we described the purpose and procedure of the study to the directors. The directors then selected project managers of current software development projects. Finally, the participants were selected by these project managers. All projects were ongoing, and at different phases of the software development lifecycle. Each participant was currently involved in one project only. The participants did not receive any compensation for participation.

Out of 251 returned results, 33 had missing data in the background and/or experience fields. Hence, we removed such responses from our analysis regarding the secondary research question. A detailed breakdown of the subjects' backgrounds and average experience is shown in Table 1.

2) *Artifacts:* We used a generic and simple requirement written in natural language with different types of gaps seeded. (Figure 1). The artifact was initially used in a prior study in a university setting [2].

For the following software requirement, do one of the following three alternatives:

1. Draw prototype screens for at least two inputs you enter.
2. Write source code in any programming language you know (C/C#, Java...).
3. Write pseudo code.

For any positive number entered by the user, the program should display a list of even numbers less than the input.

PLEASE LIST ANY QUESTIONS/ASSUMPTIONS YOU HAVE FOR YOUR SOLUTION.

Figure 1. Question delivered in the study

TABLE II. GAP TYPES AND RELATED ASSUMPTIONS

Gap Type	Related Assumption/Question
Input type	Is the type of input integer, double, float?
Prompt	Which text messages are displayed to the user?
Order	Is the order of the list ascending or descending?
Format	What is the format of the output list?
Application type	Is it a console, windows, or Web application?
Error messages	Which errors are displayed, and how would errors be handled?
Stopping condition	What is the stopping condition while listing?
Validation	How is input validation realized?
Other	Any other assumptions/questions

The gaps are easy to identify in the study's delivered question. In real life, however, identification of gaps may be a more difficult problem. Hence, the used artifact may not be appropriate. We suggest to readers planning to replicate this study to develop and/or use better artifacts (See Section 5 for more detail).

There were different gap types seeded to the above requirement. Table 2 summarizes the gap types and related assumptions.

Their background, experience, university, degree, and current project codes were also asked of the participants.

3) *Procedure:* The artifact was delivered to the selected participants as a hardcopy document. The subjects were forbidden to ask questions during the study. While delivering the artifact, it was emphasized that writing any questions and/or assumptions was very important.

On average, it took the participants 15 minutes to complete the exercise. We collected the hardcopy answers from the software directors and analyzed the collected data. After the analysis we conducted post-interviews with the participants and later, in a group meeting environment, delivered presentations about the seeded gap types and discussed the study results.

The subjects who did not write anything on paper implicitly reflected their assumptions regarding design and implementation studies. We considered written assumptions and questions as explicit assumptions.

IV. RESULTS

Approximately half of the participants (51%) selected coding to respond to the question. 28% of the subjects used pseudo code, and 12% used prototyping. About 9% (20) of the participants used more than one way to answer the question. Only two subjects listed their assumptions without using any one of the ways suggested in the question.

The gap type explicitly stated by most of the participants was found to be the gap about the stopping condition, while the gap regarding the prompt was found as the gap with the least number of explicit assumptions written for it (Figure 2).

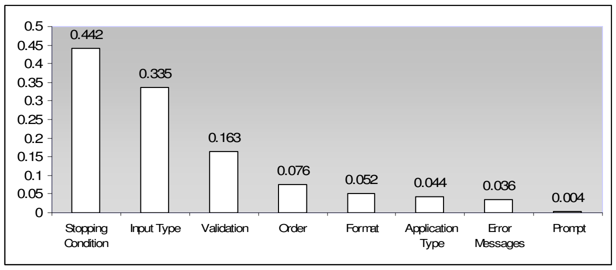

Figure 2. Average number of explicit assumptions versus gap type

We observed that many subjects did not progress to the software development phase without making explicit assumptions regarding stopping condition, input type and validation gaps.

We used SPSS 15.0 to perform the statistical analysis. For all statistical tests reported in this paper, we have used an alpha value of 0.05. The results show that both the responses of the participants and their working experience significantly impacted the number of explicit assumptions they made.

A. *Impact of Response (H1)*

Table 3 provides descriptive statistics about the subjects' answers regarding explicit assumptions.

To observe the relationship between the subjects' preferences and their tendency to complete gaps using implicit assumptions, we used one-way ANOVA. All of the assumptions of ANOVA were satisfied.

TABLE III. DESCRIPTIVE STATISTICS: RESPONSE

Response	Mean	Std. Deviation	N
code	1.47	1.646	128
pseudo code	1.69	1.527	71
prototype	1.27	1.639	30
all	1.00	1.247	10
code + prototype	1.00	1.528	7
code + pseudo code	.00	.000	2
none	4.50	3.536	2
Prototype + pseudo code	4.00	.	1
Total	1.50	1.628	251

TABLE IV. TEST OF THE ANOVA (RESPONSE TYPE)

Explicit Assumption

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	37.324	7	5.332	2.072	.047
Within Groups	625.425	243	2.574		
Total	662.749	251			

R Squared = 0.056 (Adjusted R Squared=0.029

The ANOVA was significant F (7. 243) = 2.072, p = 0.47, $\eta^2 = 0.056$ (Table 4). 5.60% of variance in the number of explicit assumption is explained by the preferred response type of the subjects. This result allows null hypothesis to be rejected in favor of $H1_a$. The preferred response of participated software engineers to the given incomplete requirement impacts the number of explicit assumptions made by the subjects.

Table 5 shows the average number of explicit assumptions per gap type with respect to response types mostly selected by the participants. All the engineers used implicit assumptions to fill the prompt gap.

TABLE V. AVERAGE NUMBER OF EXPLICIT ASSUMPTIONS WITH RESPECT TO RESPONSE TYPE

	Response Type			
Gap Type	***code***	***pseudo code***	***prototype***	***all***
Prompt	.000	.000	.000	.000
Validation	.172	.155	.133	*<u>.200</u>
Error Messages	.008	.056	<u>.100</u>	.000
Application Type	.039	.028	<u>.133</u>	.000
Format	.055	.056	<u>.067</u>	.000
Input Type	.359	<u>.380</u>	.200	.100
Order	.078	<u>.085</u>	.067	.000
Stopping Condition	.461	<u>.493</u>	.300	.300
Other	.297	<u>.437</u>	.267	.400
Max	0	4	3	1

*Underlined values in Table 6 are the maximum values of the average number of explicit assumptions corresponding to gap types. The Max row presents the number of underlined maximum scores per the participant's preferred way.

The ways subjects prefer to respond and the types of gaps were found to be related. The group that used prototyping had more explicit assumptions regarding format, error messages, and application type.

None of the participants who preferred writing source code reached the maximum score for any of the gap types. The participants who used pseudo code reached the maximum scores for gap types regarding input type, order, stopping conditions, and others. The participants who selected prototyping obtained the highest scores for non-functional requirements).

B. *Impact of Experience (H2)*

Table 6 provides descriptive statistics about the subjects' working experience in years and the number of explicit assumptions made.

TABLE VI. DESCRIPTIVE STATISTICS: EXPERIENCE

	Mean	Std. Deviation	N
Assumption	1.468	1.663	218
Experience	4.981	4.184	218

Linear regression analysis was conducted to evaluate the prediction of the number of explicit assumptions (num_{exp}) from the subjects' working experience. The regression equation for the number of explicit assumptions is

$$num_{exp} = 0.068*experience + 1.130 \quad (1)$$

2.9% of variance in num_{exp} is accounted for by its linear relationship with experience. The ANOVA was significant, F (1, 216) = 6.484, p<0.05 (Table 7).

TABLE VII. TEST OF THE ANOVA (EXPERIENCE)

Explicit Assumption

	Sum of Squares	df	Mean Square	F	Sig.
Regression	17.495	1	17.495	6.484	.012
Residual	582.780	216	2.698		
Total	600.275	217			

C. *Impact of Educational Background (H3)*

As shown in Table 2, we used two categories for the subjects' background: computer-related and others. There were 143 participants from computer-related and 75 from other backgrounds.

Figure 3 shows a box plot of educational background and number of explicit assumptions made by the participating software engineers. The box plot shows that the mean of explicit assumptions made by the participants with other backgrounds are more than of those with computer-related backgrounds.

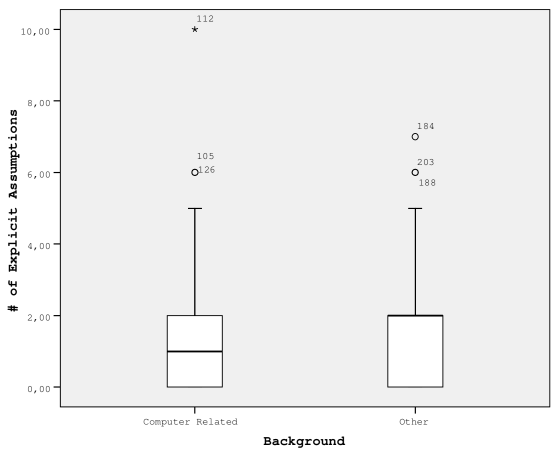

Figure 3. Educational background

A one-way analysis of variance was conducted to evaluate the relationship between educational background and number of explicit assumptions. The independent variable of background included two levels: computer-related and others. The dependent variable was the number of explicit assumptions. The ANOVA was not significant. F (1, 216) = 1.638, p = 0.202 (Table 8).

TABLE VIII. TEST OF THE ANOVA (RESPONSE TYPE)

Explicit Assumption

	Sum of Squares	**df**	**Mean Square**	**F**	**Sig.**
Between Groups	4.518	1	4.518	1.638	.202
Within Groups	595.758	216	2.758		
Total	600.275	217			

R Squared = 0.008 (Adjusted R Squared=0.003)

V. THREATS TO VALIDITY

As with any empirical study, there are various threats to validity that must be discussed. In this section we discuss the internal and external validity of our study. Internal validity is defined as the soundness of the conceptual relationships within a study.

The first threat is the threat of subject characteristics (or selection bias). We selected a convenience sample of eight companies. The subjects were selected by the project managers at these companies. Thus, we had no control over the selection of the subjects. The specific subjects who participated in the study could be the major reason for the observed results. This threat was alleviated to some degree by the fact that selected companies mostly had similar CMMI levels.

The second threat to the internal validity of this study is the threat of data-collector characteristics. At each company, different collectors collected data from the subjects. The characteristics of the data collectors might have affected results. In addition, the data collector may have unconsciously distorted the data in such a way as to make certain outcomes more likely, leading to data collector bias threat.

Approximately 14% of the subjects did not fill in the background and/or experience fields of the study. This may be considered to be a threat of loss of subjects.

Some subjects from two of the companies (E and H) had taken RE-related training two weeks before this study was conducted. We can consider this training as an unplanned event that may have affected the subjects' responses. Thus, history threat may be another threat of this study's internal validity.

External validity is defined as the degree to which results from the study can be generalized and provide insight.

The representativeness of the artifact is a threat to external validity. We used a very simple, textbook-sample-like artifact which had been previously used in a university setting [2]. We selected this generic artifact to make sure that all the subjects were equally familiar with the requirement. Since it was simple, it did not take much time for the subjects to complete. The artifact used in this study may not be reflective of an actual requirements document. A more realistic instrument could be considered for future studies.

The last threat is common to all empirical studies. It cannot be assumed that the results will always generalize beyond the setting in which the study was conducted. Thus, for more confidence in the results, the study should be replicated.

VI. DISCUSSION OF RESULTS

It is natural that the percentage of the engineers who adopted prototyping is low, because the given artifact was so simple. It is interesting that the engineers who responded using coding did not score the maximum of average number of explicit assumptions made in any one of the gap types.

During the post-interviews, the engineers stated that they make explicit assumptions when a gap occurs while tracing their algorithms. By means of post-interviews, we determined that the software engineers preferred implicit assumptions when the requirement gap has a minimal cost to update. For example, changing the message text that will appear as a prompt or updating the format of the output was found to be easy-to-do and less-costly updates. Instead of asking the users, the engineers reflected their implicit assumptions about the requirement when they proceeded with what they thought is or should be part of the requirement. In the case of rework, the engineers believed that it is not costly or time consuming to update their proposed solution. Of the interviewed engineers, 65% stated that assuming the prompt and format implicitly saves time.

The software engineers generally avoided making implicit assumptions regarding gaps that may change the algorithm and flow of the program, such as the stopping condition of a loop, input type, and validation rules. Interviewed engineers believed that changing the algorithm is an expensive update to do; thus, implicitly filling gaps related to such updates should be avoided.

Only 4% of the engineers made explicit assumptions to fill gaps related to format, application type, and error handling. After the study, most of the engineers agreed that these gaps may cause expensive updates when assumed implicitly.

More than 90% of the engineers filled the order gap implicitly. When interviewed, most of these engineers stated that they did not even realize that they were filling a gap. They assumed that their implicit assumption was the default behavior.

The subjects who mostly develop safety critical systems had the maximum number of explicit assumptions for the input type gap. The subjects currently transforming a console application to a web-based system scored the maximum number of explicit assumptions regarding the format gap type.

During the post-interviews, the engineers first stated that the type of input and the format of output could be solved at low cost. We demonstrated the intended question as a children's game, where children enter a number as a string, and the output is listed in different shapes. After the demo, almost 100% of the engineers admitted that for this question, input type and format might also be expensive to update. Thus, they all agreed that implicit assumptions should be avoided for filling any type of requirement gap and that the cost of filling the gap be determined based on the application's needs.

The subjects' former approach is similar to agile development. Agile practices usually omit the details and postpone the expenses for gathering them until the requirement needs to be fulfilled in the next iteration.

In this study, we found that those who used prototyping used better material (information from user and explicit assumptions) to fill in the gaps related to error messages, application type and format requirements. Thus, prototyping can be a good option for such requirement gaps. We also observed that engineers who preferred pseudo codes filled the input type, order, stopping condition gaps better than the others.

VII. Conclusion and Future Work

We studied software engineers' tendencies to make explicit assumptions and their preferred ways to complete incomplete requirements. We conducted a large scale experiment for quantitative analysis and structured meetings for qualitative analysis.

We've found that the way engineers respond to an incomplete requirement and the engineers' working experience significantly impact the number of explicit assumptions made by the engineers. Between the engineers' backgrounds and the number of explicit assumptions made, we did not observe a significant relationship.

To complete a given incomplete requirement for gaps related to error messages, application type, and formatting requirements, prototyping, and for other gaps pseudo codes were found to be a better way than using a programming language. Further studies may concentrate on types of gaps and their relationship to software engineers' preferences.

Future studies may focus on organizational, project, and customer-related variables. Further studies may study how software engineers complete incomplete requirements within different variables of software engineering.

We observed that training plays a crucial role; the companies that received formal RE training before the study scored the top two grades regarding the number of explicit assumptions.

References

[1] Sommerville, I., Software Engineering, Addison-Wesley, 2007.

[2] A. M. Salem, and M.O. Darter, "Requirement Analysis: A Practical Object-Oriented Approach", Journal of Computational Methods in Science and Engineering, IOS Press, vol. 6, Issue 1, 2006, pp.191-204.

[3] B.H.C. Cheng, and J.M. Atlee, "Research Directions in Requirements Engineering", Future of Software Engineering (FOSE '07), 23-25 May 2007, pp. 285-303.

[4] Zagajsek, K. Separovic, and Z. Car, "Requirements Management Process Model for Software Development Based on Legacy System Functionalities", 9th International Conference on Telecommunications, (ConTel 2007), 13-15 June 2007 pp.115-122

[5] H.F. Hoffman, and F. Lehner, "Requirements Engineering as a Success Factor in Software Projects", IEEE Software, vol. 18, no. 4, 2001, pp.58-66.

[6] H. Saiedian, and R. Dale, "Requirements Engineering: Making the Connection Between the Software Developer and Customer", Information and Software Technology, vol. 42, no. 6, 2000, pp.419-428.

[7] M. Agrawal, and K. Chari, "Software Effort, Quality, and Cycle Time:A Study of CMM Level 5 Projects", IEEE Transactions on Software Engineering, vol. 33, no. 3, March 2007, pp.145-156.

[8] M. I. Kamata, and T. Tamai, "How Does Requirements Quality Relate to Project Success or Failure?", Proceedings of the 15th International Requirements Engineering Conference, 2007 IEEE, pp. 69-78.

[9] R. Darimont, E. Delor, J. L. Roussel and A. Rifaut, "Requirements Engineering with Grail/Kaos: Tell the Requirements, All the Requirements, and Nothing Else but the Requirements", Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02), 2002, p.299.

[10] A. van Lamsweerde, and E. Letier, "From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering" RISSEF 2002, Springer-Verlag Berlin Heidelberg, 2004, pp. 325–340

[11] R. B. Rowen, "Software Project Management Under Incomplete and Ambiguous Specifications", IEEE Transactions on Engineering Management, vol. 37, no.1, Feb. 1990, pp.10–21.

[12] S. Lauesen, and O. Vinter, "Preventing Requirement Defects", Proceedings of the Sixth International Workshop on Requirements Engineering: Foundations of Software Quality, REFSQ 2000, Stockholm, Sweden, 2000.

[13] S. Kujala, M. Kauppinen, L. Lehtona, and T. Kojo, "The Role of User Involvement in Requirements Quality and Project Success", Proceedings of the 13th IEEE International Conference on Requirements Engineering, 2005, pp. 75-84.

[14] J. Noppen, P. Broek, and Akşit, M., "Software Development With Imperfect Information", Soft Computing, vol. 12, no. 1 / January, 2008, Springer Ferlag, pp. 3-28.

[15] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements Engineering and Agile Software Development", Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), 2003, pp.308-313

[16] L. Cao, and B. Ramesh, "Agile Requirements Engineering Practices: An Empirical Study", IEEE Software, 2008, pp.60-67.

[17] O. Dieste, N. Juristo, and F. Shull, "Understanding the Customer: What Do We Know About Requirements Elicitation?," IEEE Software, vol. 25, no. 2, 2008, pp. 11-13.

[18] A. Loconsole, "Empirical Studies on Requirement Management Measures", Proceedings of the 26th International Conference on Software Engineering (ICSE04), 23-28 May 2004, pp. 42-44.

[19] D. Damian, and J. Chisan, “An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management”, IEEE Transactions on Software Engineering, vol. 32, no. 7, July 2006, pp.433-453.

[20] J. Doerr, B. Paech, and M. Koehler, “Requirements Engineering Process Improvement Based on an Information Model”, *Proceedings of International Conference on Requirements Engineering (RE04)*, IEEE Computer Society Press, Los Alamitos, USA, 2004, pp. 70-79.

[21] E. Kuwana, J. D. Herbsleb, “Representing Knowledge in Requirements Engineering: An Empirical Study of What Software Engineers Need to Know”, Proceedings of IEEE International Symposium on Requirements Engineering, 1992, pp.273-276.

[22] F. Leung, N. Bolloju, “Analyzing the Quality of Domain Models Developed by Novice Systems Analysts”, Proceedings of HICSS’05 - doi.ieeecomputersociety.org 2005. pp.188-195.

[23] M. Hallmann, ”An Operational Requirement Description Model for Open Systems”, Proceedings of the 10th International Conference on Software Engineering, 11-15 April 1988, pp.286-295.

[24] A. M. Hickey, and A. Davis, “A Unified Model of Requirements Elicitation”, Journal of Management Information Systems, vol. 20 , no. 4 , 2004, pp.65-84.

[25] M. Sabetzadeh, S. Easterbrook, “An Algebraic Framework for Merging Incomplete and Inconsistent Views”, 13th International Conference on Requirements Engineering, RE 2005,2005, pp. 306-315.

[26] M. Sabetzadeh, S. Easterbrook, “View Merging in the Presence of Incompleteness and Inconsistency”, Requirements Eng, 2006, 11: pp. 174–19328 E. Kamsties, and B. Paech, “Taming Ambiguity in Natural Language Requirements”, Proceedings of the International Conference on System and Software Engineering and their Applications, December 5-8, Paris, 2000.

[27] S. W. Lee, D. C. Rine, “Missing Requirements and Relationship Discovery through Proxy Viewpoints Model”, Proceedings of the 2004 ACM symposium on Applied Computing, 2004, pp.1513 – 1518.

[28] J. Yoo, J. Catanio, R. Paul, and M. Bieber, “Relationship Analysis In Requirements Engineering”, Requirements Eng, 2004, vol. 9, pp. 238–247.

[29] A. Katasonov, and M. Sakkinen, “Requirements quality Control: A Unifying Framework”, Requirements Eng, 2006 11, pp. 42–57.

[30] O. Albayrak, M. Bicakci, and H. Bozkurt, “A Study to Observe Relations Between Software Engineers’ Responses to Incomplete Requirements and Requirements Volatility”, International Conference on Software Engineering Theory and Practice, (SETP 2009), Orlando, July 2009, pp.1-7.

[31] O. Albayrak, D. Albayrak, and T. Kilic, “Are Software Engineers’s Responses to Incomplete Requirements Related to Project Characteristics”, Proceedings of The 2nd International Conference on, the Applications of Digital Information and Web Technologies (ICADIWT 2009), London, 4-6 August 2009, pp.114-129.

[32] O. Albayrak, “Two Challenges of Teaching Systems Analysis and Design to Undergraduate Software Engineers”, in Systems Analysis and Design for Advanced Modeling Methods: Best Practices, A. Bajaj and S. Wrycza, Eds., IGI Global Publishing, 2009, pp.68-87.

[33] T. C. Lethbridge, J. D. Herrera, R.J. LeBlanc, and J. B. Thompson, “Improving Software Practice through Education: Challenges and Future Trends,” Proc. 29th International Conference Software Engineering, Future of Software Eng. 2007, Track, pp.12-28.