Value-Driven use Cases Triage for Embedded Systems: A Case Study of Cellular Phone

Neunghoe Kim[1]
Department of Computer
Korea University, Seoul, Republic of Korea

Vijayan Sugumaran[3]
Department of Decision and Information Sciences
Oakland University, Rochester, United States

Younkyu Lee[2]
Multimedia Processing Lab Samsung Advanced Institute of Technology, Suwon, Republic of Korea

Soojin Park[4, *]
Graduate School of Management of Technology
Sogang University, Seoul, Republic of Korea

***Abstract*—A well-defined and prioritized set of use cases enables the enhancement of an entire system by focusing on more important use cases identified in the previous iteration. These use cases are given more opportunities to be refined and tested. Until now, use case prioritization has been done from a user perspective, and through balanced measurement of actors/ objects usage. Lack of cost consideration for realization, however, renders it ineffective for economic purposes. Hence, this study incorporates the 'value' concept, based on cost benefit analysis, in use case prioritization for embedded systems. The use case satisfaction level is used as the surrogate for 'benefit', and the complexity of implementation for 'cost'. Based on the value, use cases are prioritized. As a proof-of-concept, we apply our value-based prioritization method to the development of a camera system in a cellular phone.**

Keywords*—Value-based software engineering; use case triage; embedded system; cost-benefit analysis

I. Introduction

Software development has evolved around users and there is more focus on developing use cases to cover the entire process such as inspection, requirement analysis, and testing. How to select use cases is one of the main issues in the planning process of an iteration-based software development project using the Unified Software Development Process [1]. The earlier a use case is placed in iteration, the more test opportunities it gets resulting in higher quality. Therefore, it is vital for improving the quality of the entire system to detect core use cases in the earlier stage and include them in the iteration plan.

Requirements prioritization has been an active area of research. Herrmann and Daneva [2] have conducted a systematic review of this literature and classified the existing requirements prioritization approaches based on several criteria and have identified fifteen well established methods. Based on their analysis, they point out several weaknesses among existing methods such as: a) not being able to estimate the benefits at the individual requirements level as opposed to the system level, b) lack of guidance for selecting the appropriate cost estimation technique given a specific context, and c) not taking into account the dependencies between various requirements [2]. Some of these limitations can be mitigated by conducting cost-benefit analysis at the use case level. We contend that this process enables us to better estimate the benefits, account for complexity as well as dependencies between requirements. Thus, use case prioritization provides a systematic approach for analyzing the benefits and cost of realization of requirements. In this research, we utilize the principles from cost-benefit analysis and determine the value of use cases by combining user preferences and complexity. Specifically, our proposed value-based use case prioritization method assigns higher priority to use cases that provide maximum satisfaction to users, while consuming minimum time and cost of realization.

The rest of the paper is structured, as follows. Section 2 discusses the prior studies related to the prioritization methods of use cases. The proposed approach and case study for value-oriented prioritization of use cases are described in Section 3. In Section 4, the evaluation conducted to verify the validity and efficiency of the proposed approach earlier is discussed. Section 5 concludes the paper, and outlines the future work.

II. Related Work

Conventional studies on use case prioritization assign priority based on objective measurement of actors and objects usage metrics as well as subjective stakeholder viewpoints [3]. However, they are not very effective for actual application since they lack economic consideration of the costs of realization. Karlsson and Ryan [4] discuss a cost-value approach for prioritizing requirements, however, they do not provide a systematic way of estimating cost. They also don't consider dependency relationships and software quality attributes.

Numeral Assignment Technique, Planning Game and the Analytic Hierarchy Process ("AHP") are three representative examples of the stakeholder preference-based approach. AHP incorporates pairwise comparison and is used for computation of relative values from stakeholders and cost of individual requirements. Technical complexity-based approaches are cost estimation methods such as the Lines of Code ("LOC"), Constructive Cost Model ("COCOMO"), Function Point Method, and Use Case Points Method. Models such as LOC and COCOMO are inappropriate for cost estimation of projects

*Corresponding Author

where the number of lines of code is hard to estimate. The problem with Function Point Method is that application to embedded systems is almost impossible due to its indifference to the internal operations of software. Finally, with respect to Use Case Points Method, the criteria are too unclear to determine the complexity and the weight of each use case for embedded systems. Hence, propose a new method for estimating use case complexity that reflects the characteristics of embedded systems.

By using the notion of value, we introduce a new method for prioritizing use cases for an embedded system, which considers both stakeholder preferences and technical complexity.

III. Value-Driven use Cases Triage Method and Case Study

A. Value of a use Case

The term "value" is defined in different ways according to the needs of different fields (e.g. marketing, business management). Typically, Cost-Benefit analysis compares benefits and costs of a project or a system. In our work, we define value as the ratio between the benefits and cost of software development, which is computed using the following equation [5]:

Benefit Cost Ratio (BCR) = Benefit/Cost (1)

In general, "cost" includes all costs ranging from capital, planning, installation, application development, to continuous maintenance, while "benefit" includes benefits from savings in labor and operation cost, and improved productivity [6].

Adapting the above equation to the software development context, the value of a use case can be determined as follows:

Value of Use Case = Level of Satisfaction of the Use Case

/Cost of Realization of the Use Case (2)

As shown in equation 2, the benefit of a single use case that specifies a particular functionality can be substituted with the satisfaction level that a user gets from the use case. Satisfaction level is measured using pairwise comparison of all the use cases, which is part of AHP. The cost factor is the cost of realization of a use case and depends on the complexity of that use case. It is measured using the extended complexity factors of the sequence diagram associated with the use case. According to equation 2, the more satisfied a user feels about a single use case, and at the same time, the less it costs for its realization, then higher the value of that use case.

B. Value-Driven use Cases Triage Method

The proposed method of prioritizing use cases involves cost-benefit analysis, which in turn computes the ratio between the benefit and the cost of developing that function. The proposed method consists of the following steps: 1) investigating relative satisfaction level, 2) identification of inter-component collaboration using state and sequence diagrams, 3) complexity calculation, and 4) use case value adjustment. Each of these steps is briefly described below and the computations done in each step is summarized in Table I.

The relative satisfaction level in Step 1 is obtained by means of AHP. Through pairwise comparison, we measure user satisfaction from realizing a function represented by a particular use case compared to other functions. The results form a comparison matrix, which is used to determine the relative satisfaction levels of use cases by averaging over the normalized columns, as proposed by Thomas Saaty [7].

The collaboration between components in a use case is examined in Step 2 using a state diagram and a sequence diagram. The Gray-Box based requirements specification method for embedded systems proposed in [8] is used to generate these diagrams. Among the objects constituting an embedded system such as controller, sensor, and actuator, we focus on the state diagram (top-level) for the embedded controller object. The information on how the state transition of the system interacts with internal components is represented in the sequence diagram.

In Step 3, the complexity of each use case is calculated based on the sequence diagram. The objects in the sequence diagram are classified as simple, average, and complex and weights assigned for each type. Similarly, messages are classified as synchronous or asynchronous, with weights given to each type. Since actors do not affect the complexity of software development, they are ignored. The weights are determined by an expert, since it is heavily dependent on the project and domain characteristics. The complexity of each use case is computed as follows using the equation discussed in [9] (shown in Table I). First, we count the total number of actors collaborating within the sequence diagram. Then, we count the number of objects in each category and multiply it by the corresponding weight for that category. Then, these weighted numbers are summed up. Similarly, we do the same type of computation for messages. The complexity of the sequence diagram is determined by adding up the scores for each of the three parts. The complexity of a use case is then determined by summing up the complexities of all the sequence diagrams that are associated with that use case. Overlapping in computation is avoided by not counting the actors, objects, and messages more than once if they appear in many sequence diagrams.

In Step 4, the value of each use case is determined by dividing the relative satisfaction level generated in Step 1 by its complexity computed in Step 3. Use case values are then adjusted by considering the dependency relationships among the use cases and their expected quality levels. Specifically, the value is adjusted if include and extend relationships exist among use cases, or when the sequential order of the use cases is established by the preconditions existing in them. The adjustment to reflect expected quality levels is to take into account the expectations on distinct quality attributes given for each use case. According to [10], the quality attributes relevant for embedded systems are: reliability, usability, performance, real timeliness, and purpose limitation. The expected quality level of each use case for each attribute is categorized as high, medium, or low. The value of each empirical weight is determined based on the project and domain characteristics. The use case value is multiplied by the weighted quality level scores to determine the adjusted value. These adjusted values determine the final order of development (priority) for each use case.

TABLE I. COMPUTATIONS USED IN VALUE-DRIVEN USE CASES TRIAGE

Step 1: **Investigating Relative Satisfaction Level (using AHP)**

$$\begin{bmatrix} Xij & \cdots & Xij \\ \vdots & \ddots & \vdots \\ Xij & \cdots & Xij \end{bmatrix} \bullet\bullet\bullet\bullet \begin{bmatrix} Xij & \cdots & Xij \\ \vdots & \ddots & \vdots \\ Xij & \cdots & Xij \end{bmatrix} \Rightarrow \begin{bmatrix} Xij & \cdots & Xij \\ \vdots & \ddots & \vdots \\ Xij & \cdots & Xij \end{bmatrix} \Rightarrow \begin{bmatrix} SL_1 \\ \vdots \\ SL_n \end{bmatrix} \quad \begin{pmatrix} \text{"Recording a Video"} \\ \text{Use Case} = 16 \end{pmatrix}$$

Use Case Comparison Matrices from All Users | Comparison Matrix Created using Geometric Mean of Individual Scores | Satisfaction Levels of Use Cases | Example

Step 2: **Identification of Inter-Component Collaboration**

The Gray-Box technique [8] is used to generate the State Diagram and the Sequence Diagram

Step 3: **Complexity Calculation (adapted from [9])**

- Complexity of a Sequence Diagram = Actor Complexity+Object Complexity+Message Complexity
  = ∑No. of actors+∑(No. of objects*object weight)+∑(No. of message*message weight)
- Complexity of Use Case = ∑(Complexity of Constituent Sequence Diagram)

Example: **Complexity Calculation from Case Study (for the "Recording a Video" Use Case)**

- Actor Complexity = User+Camsensor+MIC = 1+1+1 = 3
- Object Complexity = CamsensorIF*Complex+AudioControlIF*Average = (1*2)+(1*1.5) = 3.5
- Message Complexity = oper_set_state()*Asynchronous+get_sensordata()*Synchronous+ oper_record()*Asynchronous+oper_take_movie()*Asynchronous+set_autdio_path()*Asynchronous+get_audio_input_data()*Synchronous+check_free_space()*Asynchronous+save_recorded_data()*Asynchronous+get_temp_filename()*Asynchronous+save_file()*Asynchronous+stop_record()* Asynchronous+restartPreview()*Asynchronous = (1*1)+(1*2)+(1*1)+(1*1)+(1*1)+(1*2)+(1*1)+(1*1)+(1*1)+(1*1)+(1*1)+(1*1) = 14
- Complexity of the Sequence Diagram = 3+3.5+14 = 20.5
- Complexity of Use Case = 20.5 (This use case contained only one sequence diagram)

Step 4: **Use Case Value Adjustment**

- Adjusted Value of Base Use Case under <<extend>> = value of base use case*∑extend weight
- Adjusted Value of Included Use Case under <<include>> = value of included use case*∑include weight
- In Case of Precondition, Value of Prerequisite Use Case = value of prerequisite use case*∑precondition weight
- Adjustment of Quality Attributes = value of use case*∑(No. of quality attributes*weight of quality attributes category)

Example: **Use Case Value Adjustment ("Recording a Video" Use Case)**

- Value of Use Case = Relative Satisfaction Level/Complexity = 16/20.5 = 78 (We multiply this by 100 and round off to the nearest integer)
- Dependency (Not Related)
- Adjustment of Quality Attributes = 78*(reliability*Medium+usability*High+performance*High+real timeliness*High+purpose limitation*Low) = 78*(0.2+0.3+0.3+0.3+0.1+1) = 172 (1 is added to the sums of empirical weights to compensate for reduction of values owing to decimal values of weights)

C. Case Study: A Camera System in Cellular Phone

To demonstrate the feasibility of calculating use case values, we have conducted a case study using the camera system in a cellular phone. We implemented this system in a domestic 3G feature handset for a global electronics company. The project utilized 21 software developers and took 7 months to complete. First, use case modeling was carried out based on the requirements for the camera system. Ten use cases (Previewing, Taking a Snapshot, Recording a Video, Postviewing, Playing a Video, Album Management, Editing Photo&Video, Sending Photo&Video, Printing a Photo, Albumview) and 9 actors (Camsensor, MIC, Speaker, User, LCD, Wallpaper Manager, MMS Manager, Bluetooth Manager, Printer) were derived. The "Recording a Video" use case is used to demonstrate the computations in our approach.

1) STEP 1 Investigating relative satisfaction level: As part of the AHP methodology, the use case comparison matrices from 30 users were used to generate a single comparison matrix by taking the geometric mean of the individual comparison scores. This matrix was then normalized and the relative satisfaction levels for each of the use cases were

determined. The consistency ratio of the corresponding values in the comparison matrices from users was less than 0.1, indicating that these matrices are reliable [7]. To express the relative satisfaction levels as integer values, they were multiplied by 100 and rounded off to the nearest integer. The relative satisfaction levels for the use cases are: Taking a Snapshot = 24, Recording a Video = 16, Previewing = 11, Postviewing = 11, Playing a Video = 10, Album Management = 7, Albumview = 7, Sending Photo&Video = 6, Editing Photo&Video = 5, and Printing a Photo = 4.

2) STEP 2 identification of inter-component collaboration: A state diagram was created for the 'CameraController' component that controls the state and transition information of the camera system in the cellular phone. As an essential object controlling the state of the entire system, the 'CameraController' object functions as the owner of the state diagram [8]. In the state diagram, the camera system in a cellular phone should maintain states such as "idle," "initialized," "preview," "postview," "recording," "albumview," "editing," "sending," "printing," "playing, " "snapshot," and "stopped." As this object is used in computing the complexity of the use case, the "Recording a Video" use case triggers the state transition upon its activation, and the transition goes through the following flow: "preview → recording → preview."

Next, the sequence diagram is created, while assigning the events and the actions shown on the state diagram and marking them chronologically. In the sequence diagram of the "Recording a Video" use case, the CameraController in the "preview" state sends commands such as Oper_set_state() and Get_sensor_data() to Camsensor_IF object, upon receipt of the StartRecord event invoked by a user. Then, while carrying out its own oper_record(), the "preview" state transitions into the "recording" state. During this transition, Camsensor_IF also sends certain messages to Camsensors to fulfill the objective(s) of the message(s) it has received. Through this analytical process, it is made clear how the goal of each use case is accomplished by understanding what messages are sent and received by each component object constituting the entire system.

3) STEP 3 complexity calculation: The complexity of a use case is determined based on the information contained in the sequence diagram. The sequence diagram pertaining to the "Recording a Video" use case contains three actors, two objects and 12 messages. The object weights applied in this project are Simple = 1.0, Average = 1.5, and Complex = 2.0, while message weights are set as Synchronous = 2.0 and Asynchronous = 1.0. For example, CamsensorIF was classified as "complex,' while AudioControlIF was categorized as "average." In the case of messages, get_sensordata() and get_audio_input_data() were marked as "synchronous", while the others were deemed "asynchronous." Thus, the complexity of the sequence diagram for the "Recording a Video" use case is computed to be 20.5. It is to be noted that this use case contained only one sequence diagram. Therefore, the sequence diagram complexity also represents the complexity of the use case. The complexity values computed for the use cases in the case study are: Previewing = 28, Recording a Video = 20.5, Postviewing = 18.5, Album Management = 16.5, Sending Photo&Video = 15, Printing a Photo = 14, Playing a Video = 12.5, Taking a Snapshot = 12, Editing Photo&Video = 11, and Albumview = 10.

4) STEP 4 use case value adjustment: The use case values are obtained by dividing the relative satisfaction levels by the complexities. This ratio is expressed as whole number, by multiplying it by 100 and rounding off to the nearest integer. The values of the use cases computed in the case study are as follows: Taking a Snapshot = 200, Playing a Video = 80, Recording a Video = 78, Albumview = 70, Postviewing = 59, Editing Photo&Video = 45, Album Management = 42, Sending Photo&Video = 40, Previewing = 39, and Printing a Photo = 29. In the case of the "Playing a Video" use case, its relative satisfaction level is 10, or the 5th highest among the ten use cases, and its complexity is 12.5, or the 4th lowest among them. However, its value computes to 80, the second highest among the ten use cases.

Next, the use case values are adjusted based on inter use case dependencies and each case's expected quality level. The values are rounded off to the nearest integer. The value adjustment for the "Recording a Video" use case is shown in Table I. The adjusted values of the use cases in the case study are as follows: Taking a Snapshot = 420, Albumview = 189, Recording a Video = 172, Previewing = 128, Playing a Video = 104, Postviewing = 65, Album Management = 63, Editing Photo&Video = 50, Sending Photo&Video = 48, and Printing a Photo = 35. For the adjustment, a weight of 0.1 is assigned to dependency relationship, while three weights are assigned to the expected quality level (i.e. high = 0.3, medium = 0.2, low = 0.1). Playing a Video, Albumview, Postviewing, Editing Photo&Video, Sending Photo&Video, and Previewing use cases had their priority positions changed after the adjustment.

IV. Evaluation

In the case study described in section 3, we discussed how the use case values for a cellular phone camera system were derived. To demonstrate the effectiveness of the process, we have to answer the following three questions:

- Does the use case complexity computed through our approach match the complexity experienced in realizing the use case?
- How much do the stakeholders trust the results of our proposed method after applying it to their processes?
- Are the results from our approach more useful compared to the previous use case prioritizations that were being used?

We demonstrate the validity of our complexity calculation by showing the proportional relationship between our complexity values and the actual LOC values for corresponding use cases. In addition to this quantitative

evaluation, we demonstrate the trust shown by different stakeholders in our proposed approach by administering a survey to the marketing staff and development engineers of embedded software in the case study organization. Lastly, we show the usefulness of our proposed method through comparative analysis of the results from our method and the results from previous use case prioritizations generated by the development engineers.

A. Verification of the Complexity-Calculation Process

We counted the number of lines of executable source code upon completion of the development of the cellular phone camera system. As discussed earlier, the reason for measuring LOCs is to check whether or not the complexity-based use case priority, which had been generated prior to realization, matches the LOC size-based priority upon completion. If the two types of priority are in direct proportion to each other, our proposed method of estimating use case values is applicable to the actual development of embedded systems. To investigate the relationship between our use case complexities and the actually realized LOCs, we plotted the LOCs and the corresponding use case complexities, as shown in Fig. 1(a), (b). The use case complexity rank and the LOC rank corresponds to the rank ordering of use cases from the most complex (1) to the least complex (10) in the case study. As seen from Fig. 1(a), the complexity rank and the LOC rank for the use cases follow each other closely.

In this case study, the correlation coefficient between the complexities of the use cases computed based on our approach and the corresponding LOC was determined to be 0.96, meaning a strong relationship between them. Also, we ran a simple linear regression model with complexity as the independent variable and LOC as the dependent variable. The regression coefficients and the R^2 are shown in Fig. 1(b). The R^2 value is 0.9207, which is very significant and strongly suggests a linear relationship between the use case complexities computed through our approach and the resulting LOC. Considering these findings, it is fair to conclude that the use case complexity computation method proposed herein is a good indicator of the complexity of the actually realized code.

B. Acceptance of the Proposed Method

A survey was administered to 40 developers of embedded systems and 10 marketing staff members from the corporation that developed the camera system for the cellular phone. On average, the marketing staff members had five years of experience, and the developers had 7 years of experience.

The survey contained questions focusing on three main aspects: a) choosing use cases based on cost, b) trustworthiness of the results from our approach, and c) usefulness of our approach. With respect to the need for choosing use cases in consideration of development costs, 90% of the marketing staff and 95% of the developers indicated that cost should be considered in selecting use cases for implementation. With respect to our model's trustworthiness, 80% of the marketing staff and 85% of the developers responded positively. In terms of usefulness of our approach, 70% of the marketers and 40% of the developers acknowledged that the method would be useful in their organization. The lower percentage value of the developers may be due to unfamiliarity with modeling, personal habits, corporate culture and internal structural/organizational issues.

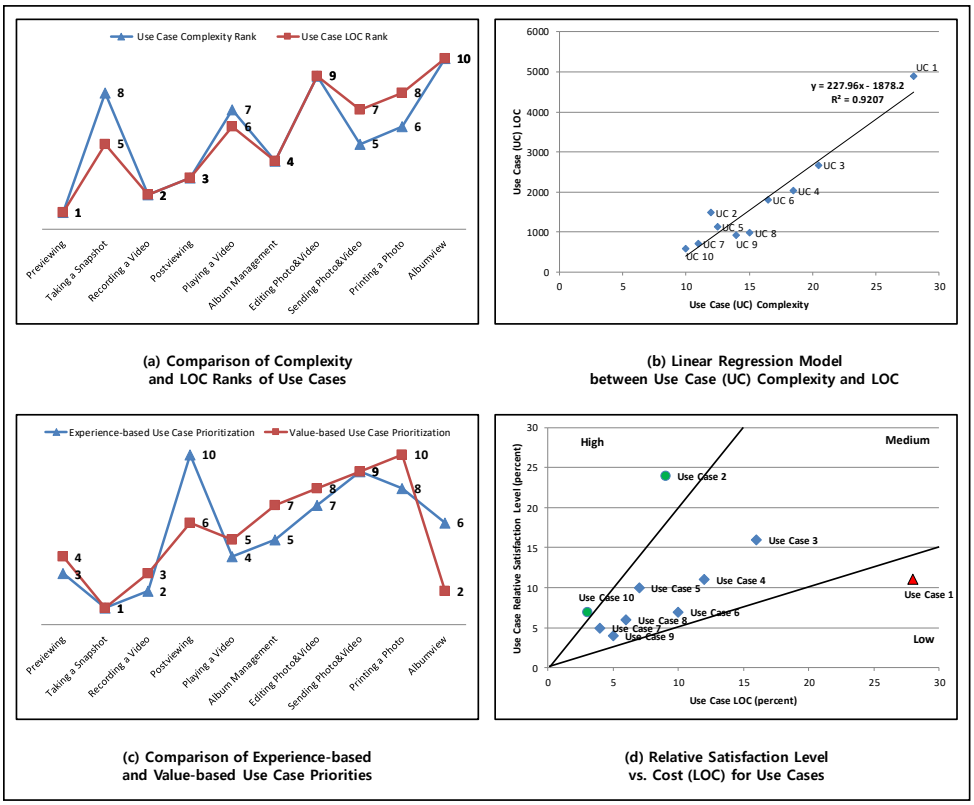

Fig. 1. Results from Case Study (Camera System in Cellular Phone).

C. Usefulness of the Proposed Method

For the ten use cases in the case study, the developers independently estimated the priority based on their prior experience. Developers tend assign high priority to use cases corresponding to basic functions even if they have high cost of realization. For the other functions, they assign lower priority even if they are complex. Fig. 1(c) shows the experience-based use case priorities and the value-based priorities developed using our approach. While the priorities are similar for a few use cases, there is considerable difference for some of them. To further analyze the differences, we have developed a cost-value diagram similar to the one discussed in [4]. The normalized relative satisfaction levels and the normalized LOC values for the use cases are plotted, as shown in Fig. 1(d). Based on the value of use cases (ratio of satisfaction level and LOC), we group the use cases into three categories (High, Medium, Low). For high value use cases, the ratio exceeds 2, for medium value, between 0.5 and 2, and for low value below 0.5, as used in [4]. As seen in Fig. 1(d), Use Case 2 (Taking a Snapshot) and 10 (Albumview) are high value use cases and they were correctly assigned a high priority value of 1 and 2 in our approach. However, the developers assigned a priority of 6 for Use Case 10, thus failing to identify this high value use case. Use Case 1 (Previewing) is a low value use case, as shown in Fig. 1(d). Our approach assigned a priority of 4, while the developers assigned a priority of 3. Thus, our approach is better able to assign more appropriate priorities compared to the experience based use case prioritization.

V. Conclusion

This study has proposed a value-based method for prioritizing use cases. This approach is used to improve quality by discerning "valuable" use cases and incorporating them in the iteration plan at an earlier stage. To demonstrate the validity and usefulness of the proposed approach, a case study and a survey were conducted.

The contributions of this study are as follows:

- In prioritizing use cases, the notion of value is defined based on the external requirement of "satisfaction level" and the internal requirement of "cost." Our prioritization process is based on value, which is a balanced metric. In this study, the cost of each use case refers to the effort required to realize that use case and it increases in direct proportion to the complexity of the use case.
- To determine the complexity of use cases tailored to the embedded system domain, it is computed based on the inter-component collaboration model.
- The validity of our model has been demonstrated by applying our complexity estimation model to an actual case and showing that the complexity estimates produced through our approach matched the actually realized LOCs.

Although we have demonstrated the feasibility of our approach, further work is needed to fully establish its efficacy. The evaluation results verify the validity of the complexity estimation of each use cases. However, further work is needed to verify the validity of users' satisfaction. As part of future work, a quantitative study will be conducted to investigate how much improvement can be achieved in the quality of the software product, when the relevant iteration planning is carried out in accordance with the prioritization results produced through our model.

References

[1] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley Professional, 1999.

[2] A. Herrmann and M. Daneva, "Requirements prioritization based on benefits and cost prediction: an agenda for future research," International Requirements Engineering Conference, pp. 125-134, September 2008.

[3] F. Moisiadis, "Prioritizing use cases and scenarios," International Conference on Technology of Object-Oriented Languages and Systems, pp. 108-119, November 2000.

[4] J. Karlsson, K. Ryan, "A cost-value approach for prioritizing requirements," IEEE Software, vol. 14, no. 5, pp. 67-74, September/October 1997.

[5] H. Erdogmus, "Cost-benefit analysis of software development techniques and practices," International Conference on Software Engineering, pp. 178-179, May 2007.

[6] T. Pisello, Return on Investment for Information Technology Providers, New Canaan Connecticut: Information Economics Press, 2001.

[7] T. L. Saaty, The Analytic Hierarchy Process, McGraw-Hill, 1980.

[8] S. Park, S. Park, "A gray-box based software requirements specification method for embedded systems," Journal of Korean Institute of Information Scientists and Engineers, vol. 38, no. 9, pp. 485-490, September 2011.

[9] A. Kanjilal, S. Sengupta, S. Bhattacharya, "Analysis of complexity of requirements: a metrics based approach," India Software Engineering Conference, pp. 131-132, February 2009.

[10] J. Lim, H. Yoon, "Extraction of quality attribute for designing the S/W architecture in weapon systems embedded software," Korean Fuzzy Logic and Intelligent Systems Society Autumn Conference, pp. 268-271, November 2006.