Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 8198039, 16 pages
http://dx.doi.org/10.1155/2016/8198039

Research Article

Toward an Agile Approach to Managing the Effect of Requirements on Software Architecture during Global Software Development

Abdulaziz Alsahli, Hameed Khan, and Sultan Alyahya

Department of Information Systems, College of Computer and Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia

Correspondence should be addressed to Abdulaziz Alsahli; alsahli9@ymail.com

Received 31 March 2016; Revised 14 July 2016; Accepted 23 July 2016

Academic Editor: Carmine Gravino

Requirement change management (RCM) is a critical activity during software development because poor RCM results in occurrence of defects, thereby resulting in software failure. To achieve RCM, efficient impact analysis is mandatory. A common repository is a good approach to maintain changed requirements, reusing and reducing effort. Thus, a better approach is needed to tailor knowledge for better change management of requirements and architecture during global software development (GSD).The objective of this research is to introduce an innovative approach for handling requirements and architecture changes simultaneously during global software development. The approach makes use of Case-Based Reasoning (CBR) and agile practices. Agile practices make our approach iterative, whereas CBR stores requirements and makes them reusable. Twin Peaks is our base model, meaning that requirements and architecture are handled simultaneously. For this research, grounded theory has been applied; similarly, interviews from domain experts were conducted. Interview and literature transcripts formed the basis of data collection in grounded theory. Physical saturation of theory has been achieved through a published case study and developed tool. Expert reviews and statistical analysis have been used for evaluation. The proposed approach resulted in effective change management of requirements and architecture simultaneously during global software development.

1. Introduction

Requirement engineering (RE) is defined as specifying and representing the customers' requirements, thereby giving them a proper structure with which to meet them. RE is the foremost step of the software development lifecycle. All other phases are highly dependent on RE. A change in one phase affects the quality of other phases. If requirements are not handled in early phases, they may affect the quality of software and also result in failure of the system [1].

Efficient RE critically depends upon effective communication and coordination of requirement development and management. Producing perfect requirement specifications becomes difficult in GSD because of geographical, temporal, and social distances between development teams. GSD interprets communication gaps between different people involved in software projects [2]. With communication gaps, requirement management becomes a major challenge in global software development [3]. Requirement management facilitates and manages changing requirements throughout the software development phases. Requirement management also handles relationships among requirements, traceability, and dependency between requirements and the software architecture [3]. RCM is a subtask of requirement management, and the relationship is depicted in Figure 1. Requirement change management includes several activities such as reporting the change, analyzing the possible impact of the change, and finally implementing the change in the system.

When a change request is initiated by any stakeholder, software developers analyze the change and its ripple effect on the requirement document. Changes may occur at any phase of the software development lifecycle. The major issue is analyzing the change and its impact. Thus, impact analysis of change management is the most important part of RCM.

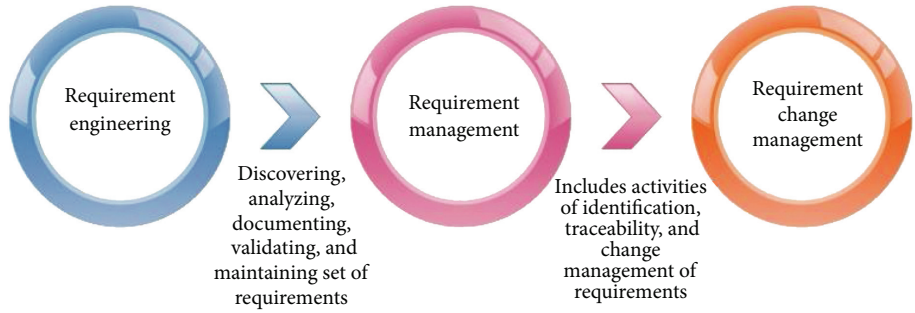

Figure 1: Context of requirement change management.

Most users are unclear about functional requirements and business requirements. As the environment and technology change, requirements and their perspective and level of understanding also change. Similarly, certain new requirements emerge with the changing requirements. In doing so, requirement engineers should possess knowledge regarding relevant requirements and domain knowledge. Because requirements tend to change, it becomes difficult to manage relevant requirement knowledge and past experiences. Thus, software and requirement engineers cannot keep pace with the changing requirement knowledge. Organizations that maintain a proper knowledge base have a higher impact factor than those that do not.

Case-based reasoning (CBR) helps in storing and reusing past cases based on similar situations and their results in a case-based repository. The repository forms a knowledge base for the system by storing relevant experiences and knowledge assessments. The vital factor of CBR is that it allows for retrieval and reuse of previously stored cases to solve similar situations [4, 5]. CBR involves retrieving cases from the case-based repository (knowledge repository), reusing retrieved cases, revising the past experiences, and adapting the retrieved case. After the new case has been merged with the previous retrieved cases, retaining is most beneficial step in which retrieved cases must be retained together with the new case as a new solution in CBR [5]. Similarly, synchronization of requirements and architecture is a reported challenge of GSD [1, 6, 7]. Because architectural design is dependent on the requirement document, the architecture also changes with the changing requirements, so it is important to keep pace with the requirements and changes to the architecture [8–11]. Domain knowledge and requirements are stored in the case-based repository, but there exists no proper approach or tool for handling a changing architecture due to changing requirements. With the changing requirements, many new conflicts arise, and traceability issues are also faced. Past experiences also must be kept in mind while reusing those cases. Commonly, utilizing past experiences, utilizing domain knowledge, and storing records of both of them are not properly achieved. Thus, software developers cannot use past knowledge, which is the most vital asset in implementing the new changes. There is a need for a better approach that overcomes the issues related to requirement and architecture interplay, gives a better solution for managing the changing requirements, and solves the issues of change management during GSD. Our main goal is to propose a hybrid approach that will handle change management for both the requirements and architecture during global software development. We will prove our hypothesis through statistical analysis and a conducted case study. The paper is structured as follows: Section 2 explores and describes related work regarding our area of research. Section 3 describes our applied research methodology, which is followed by a description of the proposed approach in Section 4. The results gathered and compiled from both the expert review and a case study are summarized and discussed in Section 5. In the end, the conclusion of our research and limitations are discussed in Section 6.

2. Related Work

This section describes related work regarding requirement change management and synchronization of requirements and architecture. The section also explores related approaches that exist in the literature and their limitations and motivation.

2.1. Requirement Change Management. Requirement change management includes several activities such as reporting the change, analyzing possible impacts of the change, and finally implementing the change in the system [13]. Requirements continually change as a response to competitors in the market, changes in technology, changing laws, and user preferences. Poor change management is the foremost barrier to the success of software systems, possibly resulting in system failure. Changes to requirements during software development place the schedule, quality, and cost at risk [14]. As a project enters later phases, handling of changes becomes difficult, so costs of changes also increase [1]. It is essential to obtain prior knowledge to achieve better change management. In addition, the change might result in altered functionality of system components, so change impact analysis is essential in handling changes [3]. Although impact analysis is vital, it is also risky. It could have both positive and negative impacts on software functionality, quality, sales, customers' attitudes, schedule, and so forth. The change control board is responsible for addressing activities of change management, which include identification of the change, analysis of the change (including impact analysis), approval of the change, and implementation of the desired change [15].

López et al. [16] explained risks and challenges associated with change management during global software development. The authors explained that RCM is difficult because changes may incur delays, so they suggested a safeguard to overcome the challenge. Each change made in the requirement document shall be announced to the project manager or the analyst so that the change can be discussed and accepted by the whole team. The change control board describes the channels for accepting a change. Its pros and cons are discussed, and voting takes place. Patil and Ade [17] proposed a system that provides a secure cloud for requirement information in GSD, thus simplifying RCM.

2.1.1. Case-Based Reasoning. Case-based reasoning can serve as an efficient approach for handling requirement change management. Jani [18] proposed a hybrid agile model for the evolution of agile practices by incorporating the artificial intelligence technique of case-based reasoning. The model resulted in enhancing agile practices through a case-based repository, through which changes were easily managed. Similarly, Tidemann et al. [19] proposed a new approach for architecture handling. They applied their proposed approach to a Norwegian fish farm business. Because the information was stored conventionally, they introduced knowledge management to the earlier stages through case-based reasoning. Their approach not only was limited to the fish farm business but is also applicable to other research industries. Those studies that exist in the literature regarding the use of case-based reasoning helped us incorporate a case-based knowledge repository in our proposed approach.

2.2. Synchronization of Requirements and Architecture. To keep the requirements and architecture synchronous, Nuseibeh [12] proposed a model that accepts changing requirements as in the spiral lifecycle approach. It is an alternative to the frozen requirements of a waterfall, and it developed a Twin Peaks model to evolve requirements and architecture concurrently, adaptive to changes in requirements. However, the current Twin Peaks model lacks practical details about how the synchronization is achieved, especially during GSD. Pimentel et al. [10] used the I* framework to address the coevolution of requirements and architecture as forward evolution considering requirement changes, its effects on components of the architecture, whether it is necessary to reconfigure an architecture to suit those changes in the requirements, and backward evolution, which addresses changes in architecture, for example, whether degradation of components might prevent the fulfillment of requirements. However, his work is challenged by the lack of tools supporting the I* framework, poor readability, and bad scalability. Use case maps (UCMs) [20] relate requirements and architecture by linking functional requirements represented by use cases to architectural components, employing use case maps in which use cases are linked to each other and to their related architectural components in a map-like view. The technique is used early during software development and scales as requirements evolve. However, it suffers from the large human intervention that is required, and its semantics are unclear. Hall et al. [21] developed problem frames to explore the problem domain for applications developed in less-known application domains by conceptualizing the problems in the requirements. They further extended problem frames to concentrate on not only the problem domain of requirements but also the solution domain of architecture. Their work is limited by the fact that it is suitable only for rare or new domains. To the best of our knowledge, none of the above or similar techniques has been previously applied to overcome the associated challenges of requirement change during global software development.

2.2.1. Twin Peaks Model. Nuseibeh [12] proposed Twin Peaks as a spiral model to be an alternative to the sequential development of requirements and architecture. The author explained Twin Peaks for handling requirements and architecture simultaneously. Their approach proved to help develop a software project incrementally and speedily. Separation of the requirement phase from the architectural phase often is unachievable. Also choice of requirements easily impacts the architecture developed by software architects. The proposed approach was tailored to the intertwined activities of both requirement engineering and architectural development. The approach made use of different activities and processes related to software development phases, thus resulting in enhanced architectural design and stakeholder satisfaction. Their proposed approach handled a major risk associated with software development; that is, requirements continue to emerge after feedback has been gathered regarding the developed prototypes and models; the Twin Peaks model incrementally handles those emerging requirements. Adopting proposed Twin Peaks model helps in iterative, synchronous, and rapid development of requirements and architecture. Twin Peaks can be used to manage rapid changes. Their approach also helps in synchronizing core requirements with the architecture. Their proposed approach is shown in Figure 2. Although the approach was quite efficient, it did not explain the types of architectural patterns to be addressed. There are other implementations of Twin Peaks existing in the literature and also in industry; for example, Mirakhorli and Cleland-Huang [22] applied Twin Peaks in the space industry and tailored them to handle critical requirements and architecture.

3. Research Methodology

The incorporated research method was a qualitative grounded theory. The main reason for choosing a grounded theory was to investigate the complex situation and real-world problems regarding the interplay of requirements and architecture during GSD. Qualitative data collection such as interviews helped in understanding the concerns of the contexts of global software development in detail. Grounded theory (GT) is the systematic generation of a theory that overcomes the issues related to the context. It is based on the data collected from the literature and interviews [23–25]. GT was developed by Barney and Anselm [26], who presented it in the published book *The Discovery of Grounded Theory* [27]. Glaser explained grounded theory as follows:

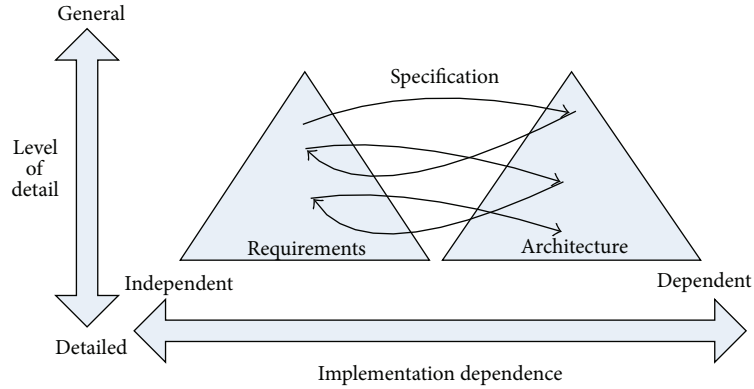

Figure 2: Twin Peaks model proposed by [12].

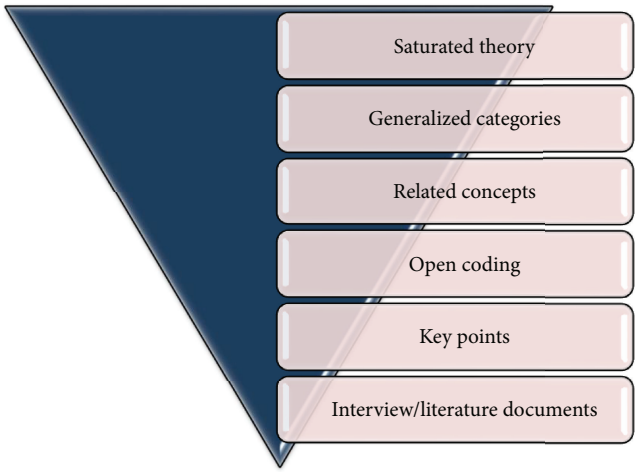

Figure 3: Abstraction levels of grounded theory.

> *The grounded theory approach is a general methodology of analysis linked with data collection that uses a systematically applied set of methods to generate an inductive theory about a substantive area. [28]*

Grounded theory aims at proving a generated hypothesis through constant comparison of identified codes and concepts at each incremental level of abstraction. Several factors led to the selection of grounded theory. First, grounded theory is suitable for use in the investigation of complex situations [29] and thus was best suited. Second, rather than testing an existing theory, grounded theory is used for the generation of a theory, which allows a researcher to move from data to theory systematically [24]. Third, both grounded theory and agile software development rely on the social behavior of people, so a strong correlation exists between them. Figure 3 shows the level of abstraction of grounded theory.

In the following section, we elaborate the theory generation process using grounded theory. We also describe some data collection methods, codes, and identified concepts.

3.1. Data Collection. Research was initiated by collecting data from both literature and interviews. Different studies were searched through extensive online databases. All selected literature incorporated change management issues and applicability of agile software development during global software development. In addition, issues were collected from interviews of different personnel working in global software development. Most teams were located in Pakistan, India, and Saudi Arabia. A total of twenty participants contributed to this research. Different experts from the domains of change management, global software development, and agile development in addition to experts in case-based reasoning were consulted. The experts were also consulted about synchronization of requirements and architecture.

3.2. Data Analysis. The data collected from both the interviews and literatures were further analyzed by finding independent and dependent variables. Regression models were applied to all identified issues of requirement and architecture management during global software development. The ripple effect caused by requirement change was also estimated by agile methodologies applied to architecture and requirement management during global software development. The constant comparison method was used for further analysis of the data and key points.

3.3. Open Coding. Jones and Alony [25] described open coding as the foremost step for extracting key points from interview/literature transcripts. Open coding is defined as breaking gathered data into codes and concepts, which further combine to form a progressive grounded theory. Key points save time by gathering in codes that can be easily generated. An example of a key point, code, and related concepts is shown below:

> Interview quotation: "*Although requirements are gathered, sometimes it becomes difficult to embed changes owing to creeping requirements because of geographical dispersion.*"

Key point (KP1): mismatch requirements and architecture traceability.

Code: architecture change management.

Figure 4 shows how related concepts were extracted from the interview and literature transcripts.

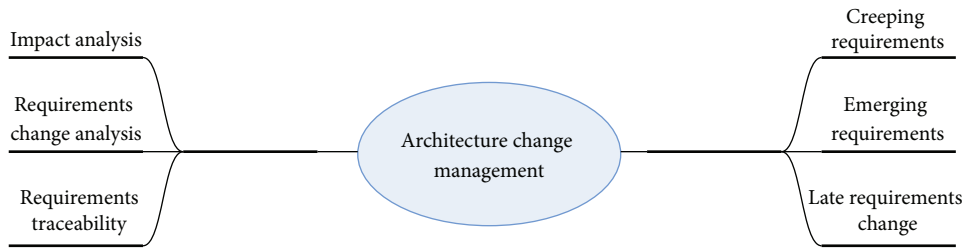

Figure 4: Related concepts for the extracted code from both the interview and the literature transcripts.

Table 1: Progressive grounded theory mapped to six C's coding family.

Six C's coding family	Theory of synchronized development of requirements and architecture during GSD
Condition	Requirements keep changing; therefore architecture changes during GSD.
Covariance	Agile approach changes with changes in causes of the theory of synchronized development of requirements and architecture.
Consequences	Mismatch requirements and architecture. As architecture does not match clients' needs, this makes design unacceptable.
Context	Requirements and architecture change management during GSD.
Causes	Stakeholders change core requirements and it is not translated into architectural design during GSD.
Contingencies	Requirements handling, architecture handling, and effective collaboration.

3.4. Generating Progressive Theory. Several studies in the literature explained how grounded theory can be generated and validated [23–25, 27, 28]. To prove our progressive grounded theory and achieve real-world saturation, case-based reasoning and scrum were used for requirement and architecture management, whereas Twin Peaks models were applied for handling the synchronization of requirements and architecture. Thus, based on identified codes, concepts, and categories, progressive grounded theory was proposed.

According to our proposed theory, the use of scrum practices for software development and case-based reasoning for requirement and architecture management, when merged with the Twin Peaks model, results in efficient requirement and architecture management and efficient synchronization during global software development. In the end, a hybrid agile approach was proposed that was further evaluated based on tool and a case study. The proposed theory was mapped with the six C's coding family proposed by Glaser [30]. All six C's were mapped with the approach. Table 1 elaborates our proposed grounded theory mapped to the six C's coding family.

4. Hybrid Agile Approach

To prove our hypothesis presented in the form of grounded theory, a hybrid agile approach was needed for its real-world saturation. However, before proceeding to its logical view, we proposed its conceptual view.

4.1. Information Model. The conceptual view has been presented in the form of an information model. Regarding requirement change management and requirement and architecture traceability, the information model is well suited. It incorporates different contexts with their associated relationships and suitable information for the system. The information model has been explained in Figure 5.

The information model shows that a case-based repository has been merged within the system to handle changes in the requirement. When the project is initiated, requirements are gathered and explained in a requirements document. Each requirement from the requirements document is converted to specific cases. Each case represents each requirement stored in the case-based repository. Whenever changes occur, the change control system is initiated. Changes to be made in a specific requirement are checked in the repository. If it exists in the repository, it is reused; otherwise, it is added to the repository. After the case is maintained, a case report is generated. After the information model, a logical view is shown in the form of a hybrid agile approach. Figure 6 explains the proposed hybrid agile approach.

4.2. Approach Description. The approach is distributed into two phases, each of which is implemented at each site. Requirement management is performed at location 1, whereas architecture management is conducted at location 2. The model has been proposed based on the Twin Peaks model. The model progresses from generalized to detailed and from independent to dependent. Site 1 is the requirement phase. First, preliminary requirements with their associated goals are identified through user stories. After user stories have been compiled, each one is elaborated. Each feature is selected and described with reference to their associated

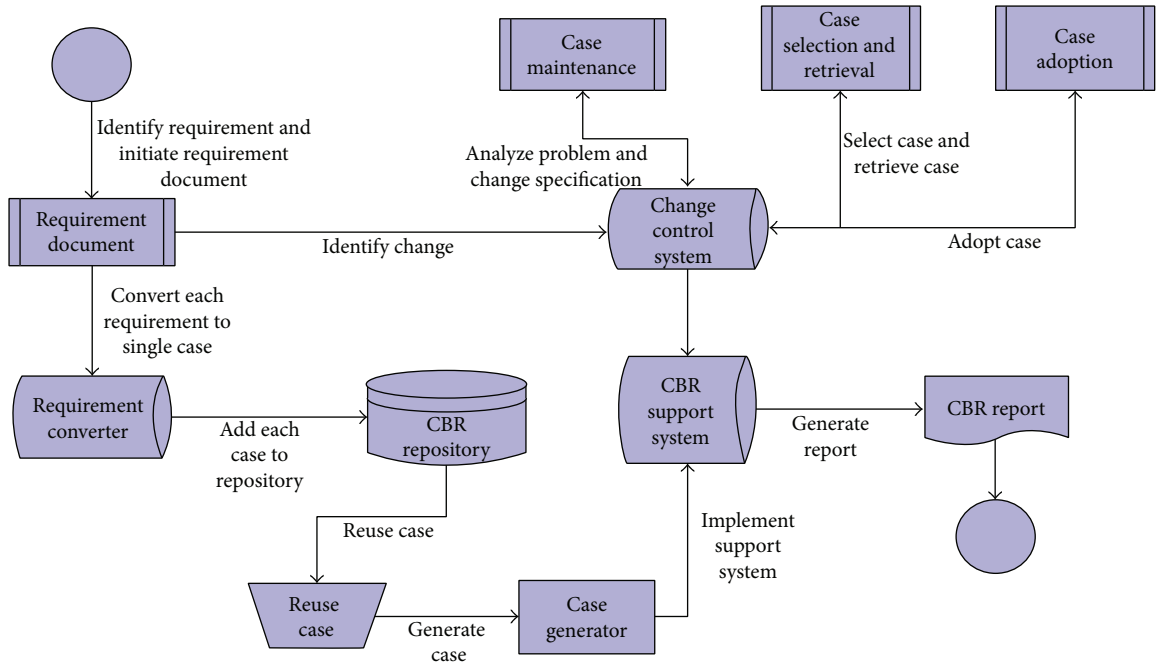

Figure 5: Proposed information model.

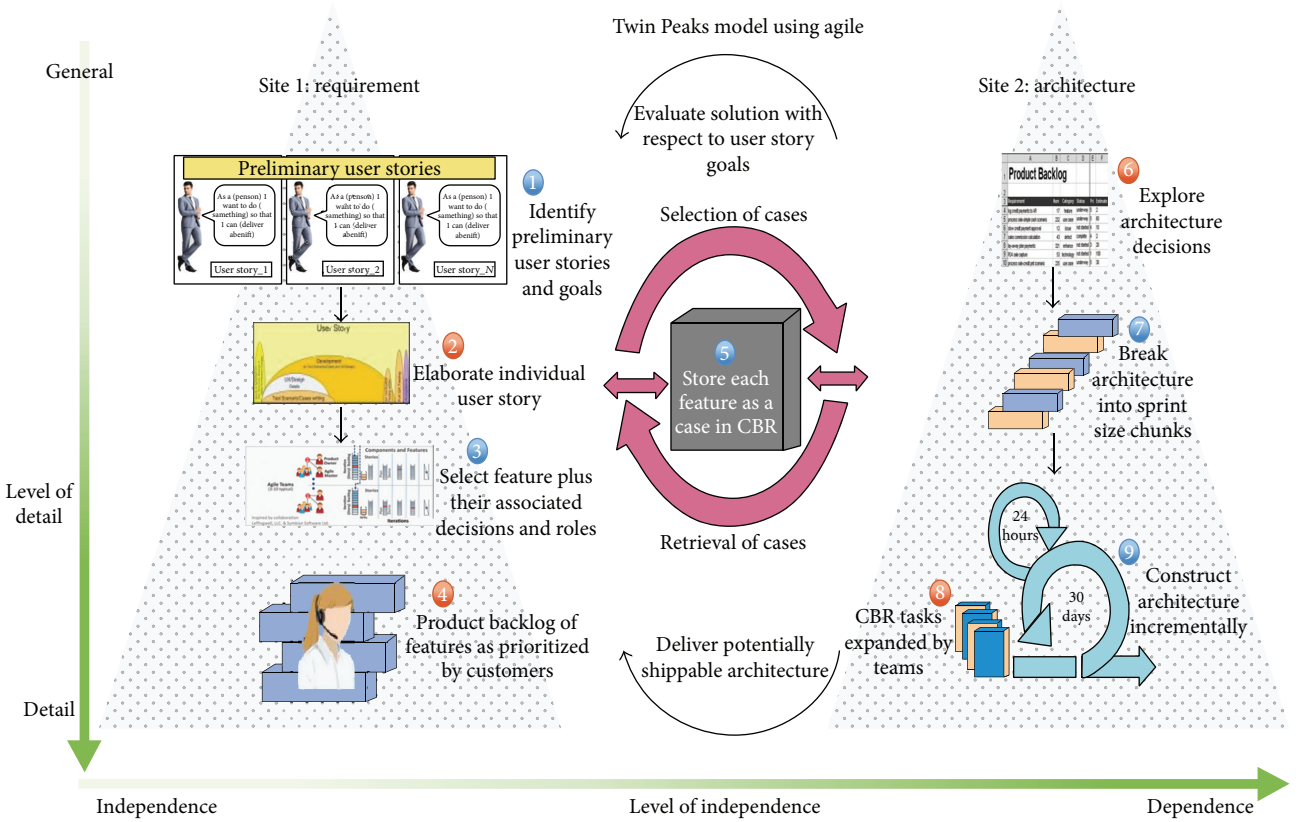

Figure 6: Proposed hybrid agile approach.

Table 2: Context of selected domain experts.

Expert #	Agile position	Agile method	Org size	Location	Domain	Team size	Duration (months)	Iteration (weeks)
E1	Agile coach	Scrum	M	India	Agile training	8	48	3
E2–E9	Software architect	Scrum & traditional	S	Pakistan	Software development	4	12	1
E10	Project manager	Scrum & RUP	L	India	Software rational development	14–17	9	4
E11–E14	Change evaluator	Scrum & XP	M	Saudi Arabia	Distributed development	8–10	4-5	3
E15–E17	Change verifier	Scrum & XP	M	Saudi Arabia	Agile trainer	8–10	14-15	4
E18–E20	Change request receiver	Scrum & XP	M	Saudi Arabia	Telecom	10	8-9	3

decisions and goals. Requirements are prioritized and stored in a product backlog. The customer is responsible for prioritizing requirements in the product backlog. On the middleware, a case-based repository is stored online. From site 1, requirements are converted into cases that are stored in the CBR. At site 2, the architecture phase is managed. Architectural decisions are explored for each case. The software architect is responsible for carrying out architectural decisions. Because Twin Peaks is iterative, incremental, and speedy, each architectural decision is broken down into small and iterative releases. Each release results from each broken-down architectural decision into small sprints. Cases are expanded by team members, and the architecture is developed. Each solution is evaluated with reference to user story goals. Once the architecture is developed, a shippable architecture is transferred to the client organization. Both distributed teams communicate through an online web-based tool.

5. Results and Discussion

In this section, the results are evaluated and discussed.

5.1. Statistical Analysis. Evaluation of the proposed approach was conducted based on interviews with domain experts. The domain experts belonged to different domains, and their level of experience and detail is explained in Table 2. A total of 20 domain experts were selected and participated in the questionnaire.

To evaluate our proposed theory mapped with the proposed approach, seven null hypotheses were proposed. The raw data collected from interviews and questionnaires have been proposed statistically to prove our null hypothesis. Advantages of statistical analysis include description of big data with tables and charts. Seven questions were generated for the null hypothesis, which were tailored to the main issues of requirement and architecture change management and the extent to which our proposed approach addresses those issues. A total of 30 companies were contacted, from which 20 responses were received. Out of 20 responses, 5 responses belonged to conventional collocated organizations. Our analysis showed that agile CBR, when merged with the Twin Peaks model, results in better synchronization and change management of requirements and architecture during GSD. The seven generated null hypotheses are elaborated below with their associated variables.

H_0^A: requirements tend to provide better coverage when stored in the form of cases in the case-based repository.

H_0^B: Twin Peaks in proposed approach provides better requirement architecture synchronization.

H_0^C: when using agile development, breaking the architecture into small sprints provides better change management.

H_0^D: the case-based repository increases the complexity of retrieval of cases which hinder requirement change management.

H_0^E: the relationship between dependent and independent variable is easily handled through the proposed approach.

H_0^F: using agile practices alone will overcome the problem of requirements and architecture synchronization.

H_0^G: the requirements and architecture management vary from the conventional to the distributed case-based repository approach.

And alternate hypotheses are as follows.

$H_0^{A\prime}$: requirements do not provide better coverage when stored in the form of cases in the case-based repository.

$H_0^{B\prime}$: Twin Peaks in proposed approach does not provide better requirement architecture synchronization.

$H_0^{C\prime}$: when using agile development, breaking the architecture into small sprints does not provide better change management.

$H_0^{D\prime}$: the case-based repository decreases the complexity of retrieval of cases which provide better requirement change management.

Table 3: Questionnaire details against each hypothesis.

Variable	Question	Rating scale	Numeric mapping
A	Do the requirements tend to provide better coverage when stored in the form of cases in the case-based repository?	1	Failed coverage
		⋮	⋮
		5	Full coverage
B	Does Twin Peaks in the proposed approach provide better requirement and architecture synchronization?	1	Complete failure
		⋮	⋮
		5	Full synchronization
C	While using agile, does breaking the architecture into small sprints provide better change management?	1	Not at all
		⋮	⋮
		5	Excellent management
D	Does the use of a case-based repository increase the complexity of case retrieval, thus hindering requirement change management?	1	Don't know
		2	Up to 25%
		3	26–50%
		4	51–75%
		5	76–100%
E	Is the relationship between dependent and independent variables covered properly?	1	Don't know
		2	Up to 25%
		3	26–50%
		4	51–75%
		5	76–100%
F	Does agile practice during GSD lack synchronization between changing requirements and architecture?	1	Don't know
		2	Up to 25%
		3	26–50%
		4	51–75%
		5	76–100%
G	Is there any difference between requirement and architecture management from the conventional to the distributed case-based repository approach?	1	Less than 20%
		2	Less than 40%
		3	Less than 60%
		4	Less than 80%
		5	Less than 100%

$H_0^{E\prime}$: the relationship between dependent and independent variable is not handled through the proposed approach.

$H_0^{F\prime}$: using agile practices alone will not overcome the problem of requirements and architecture synchronization.

$H_0^{G\prime}$: the requirements and architecture management is the same in the conventional and the distributed case-based repository approach.

Each hypothesis was tested against the defined questions. The questions are defined with their rating scale in Table 3. The variables used for each hypothesis were included in the questions.

Domain experts gave their responses according to the rating scale. Table 3 shows the details of the questionnaire against each hypothesis.

Statistical analysis was performed on the responses gathered. Subsequently, a Mann-Whitney U test was also performed to prove our null hypothesis [31]. Probability represented by p was also calculated. To perform the U test and determine the probability, a significance level denoted by $\alpha = 0.05$ was set and the critical value of U at $p \leq 0.05$ is 18. If the value of probability was above the significance distribution and the value of U is above the value of critical value of U, then our hypothesis could be accepted; otherwise, the hypothesis would be rejected. Thus, the alternative hypothesis was proposed in case of rejection. Table 4 shows the complete statistical analysis. In Table 4 T defines the organization type, min stands for minimum and max for maximum value, med refers to median, M. rank refers to mean rank, and Std. dev refers to standard deviation. U test and p (probability) were also calculated. In the table, a comparison has been made between the distributed agile (DA) and conventional (C) responses. The values of U and p were interpreted to make a comparison between the two datasets. Of the 7 hypotheses, 2 were rejected because the value of probability lies below the significance level and the

Table 4: Statistical results.

P.	T	#	Mean	Min/Max	Med	M. rank	Std. dev	U	p
H_0^A: coverage	DA	15	3	1/5	3	10.5	1.316	35	0.43
	C	5	2.8	2/4	3	11	0.748		
H_0^B: Twin Peaks	DA	15	2.8	1/5	3	10.5	1.423	29	0.24
	C	5	1.8	1/3	2	11	0.748		
H_0^C: agile motivation	DA	15	3.2	1/5	3	10.5	1.275	29.5	0.25
	C	5	2.8	2/4	3	11	0.748		
H_0^D: CBR complication	DA	15	3.3	1/5	4	10.5	1.135	11.5	0.01
	C	5	1.8	1/3	2	11	0.748		
H_0^E: real-time communication	DA	15	2.8	1/5	3	10.5	1.223	29.5	0.25
	C	5	2.4	2/3	2	11	0.489		
H_0^F: agile practices	DA	15	3.8	1/5	4	10.5	1.107	15	0.02
	C	5	2.4	2/3	2	11	0.489		
H_0^G: requirement & architecture	DA	15	3.6	1/5	4	10.5	1.306	35.5	0.44
	C	5	3.6	2/4	4	11	1.019		

value of U is below the critical value of U; thus, the alternate hypothesis was proposed. The remaining 5 were accepted. From the obtained results, it was clear that the use of agile CBR over Twin Peaks helped in synchronizing requirement change in software architecture.

5.2. Tool Development. To develop a tool to evaluate the proposed approach, architecture was also developed. Figure 7 describes the architectural view of the tool. We have mapped our architecture on the three-tier architecture. The first tier is the presentation side, which comprises the requirement catalog for the addition and selection of requirements and the change initiator menu. The presentation tier allows the client of the system to initiate and handle change and case addition to the repository. The presentation layer also allows the client to represent their case reports and requirement document formed based on the requirements. The second layer is the application layer, which comprises the CBR support system. It also comprises a converter and a generator. The requirement converter converts each requirement into a case and a case generator that generates similar cases to the change request. The final layer is the data layer. All data storage and repositories are handled in this layer.

To validate our approach, we mapped a case study of a remote tower monitoring system with our proposed approach. Similarly, a tool was developed for the requirement and architecture change management system (RACMS). Figures 8–16 show the different functionalities of the developed tool RACMS.

5.3. Case Study: Int Netlink. We have overlapped our approach with Int Netlink's remote tower monitoring system case study. Int Netlink is a Dubai-based company that provides tower installation and maintenance services to telecom companies. They were contracted by a reputable telecom company to install and monitor telecom towers in Kabul, Afghanistan.

To monitor the towers effectively, they wanted to develop a software solution that can help their staff remotely monitor each site. The main goals in developing this application were to

(i) monitor remote sites;

(ii) collect data for future use.

Each remote site was required to have a separate dashboard interface to provide in-depth details in the form of gauges and alert reports of the individual sites to the associated staff.

The main requirement was to monitor the following attributes of each remote site:

(i) input voltage;

(ii) output voltage;

(iii) diesel generator voltage;

(iv) diesel generator fuel level;

(v) temperature.

The hardware installed in each remote site had the capability of sending a raw data string at set intervals of time via the TCP/IP protocol to the given IP address of the remote server, where the software application would collect data, parse it, store it in the database, and render it in the dashboard.

The software application uses socket technology to listen to and receive the data from any of the remote sites. Each remote site was configured with the server's IP address and port to which the raw data was sent.

The development team organized the requirements in the following manner:

(i) *Requirement 1*: program socket interface that will listen to any string from the remote sites.

(ii) *Requirement 2*: accept data from the socket and parse it based on the given cheat sheet.

(iii) *Requirement 3*: categorize the parsed data into various given attributes and store them into the database tables of each attribute.

(iv) *Requirement 4*: calculate alerts separately and store them into the database alert table.

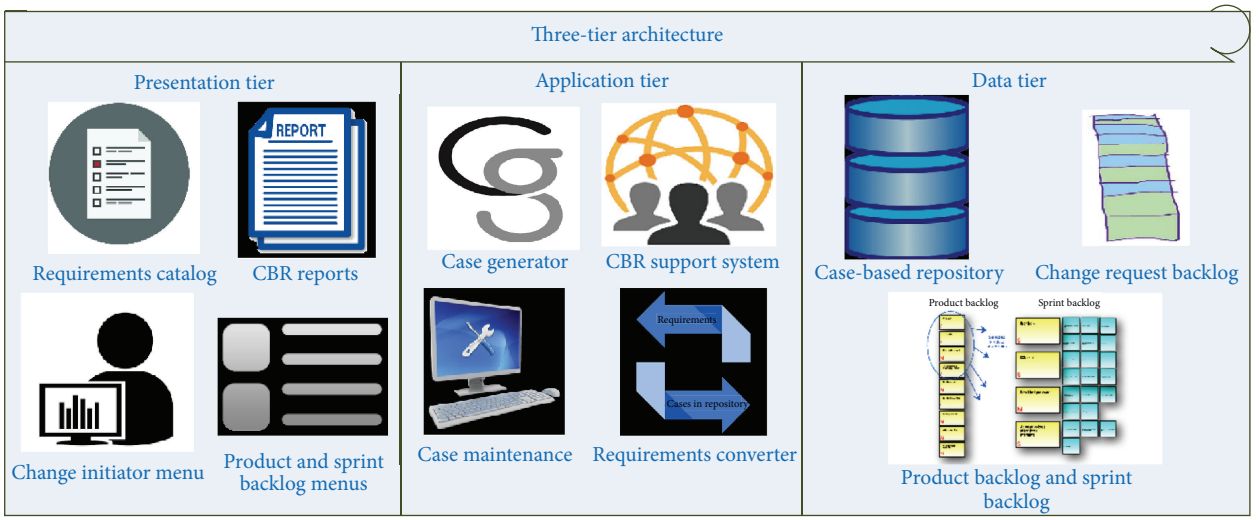

Figure 7: Architectural view.

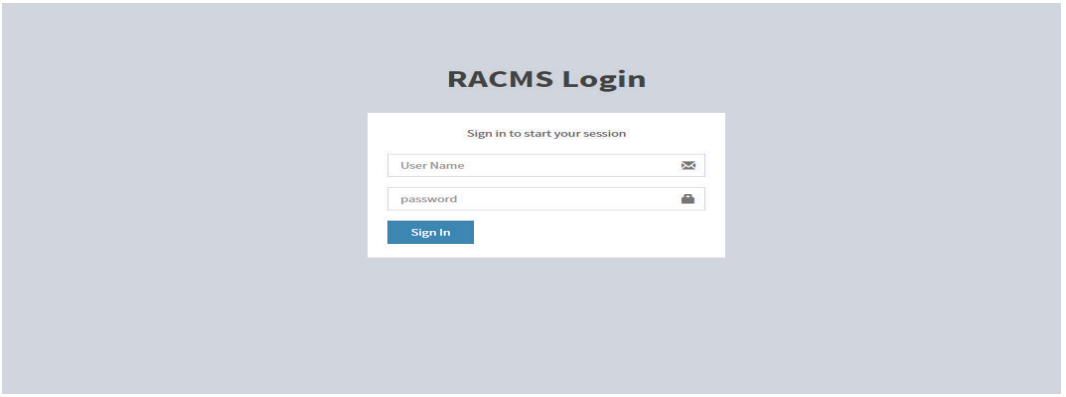

Figure 8: Login interface of the RACMS tool.

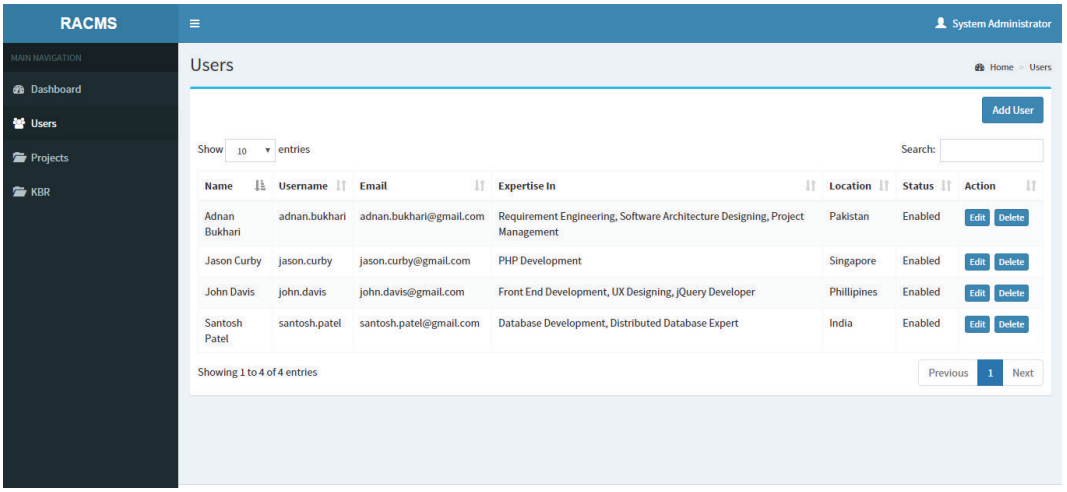

Figure 9: Management of the users of the tool.

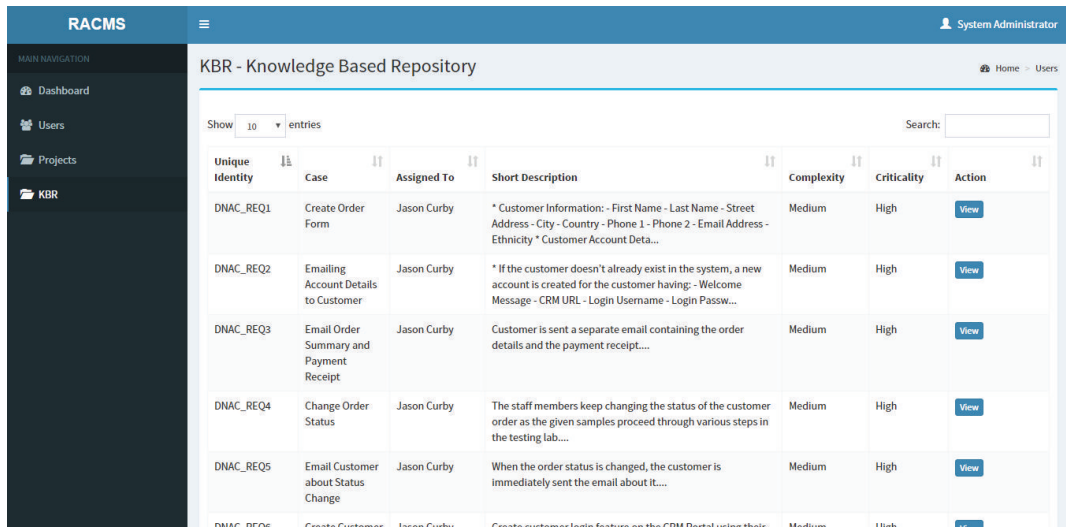

Figure 10: Knowledge-based repository.

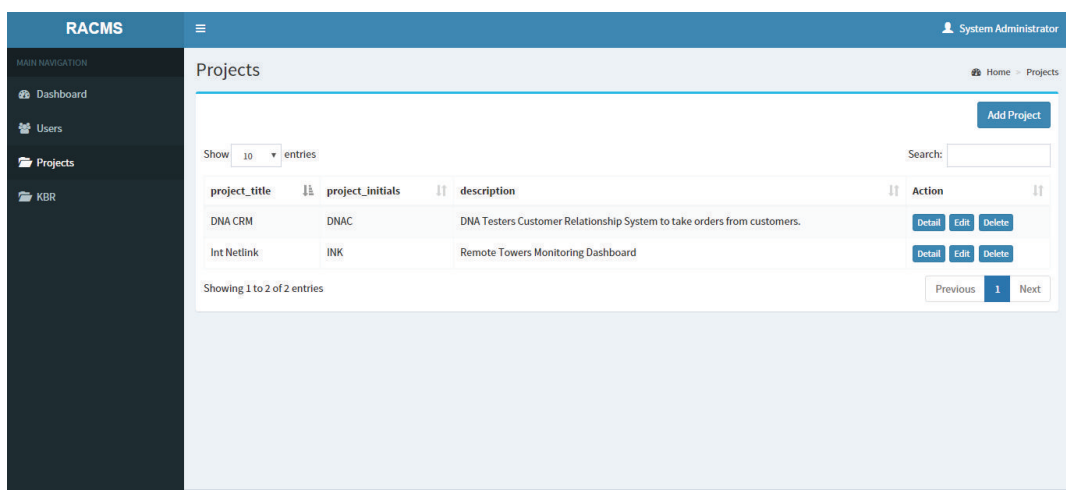

Figure 11: Projects management menu.

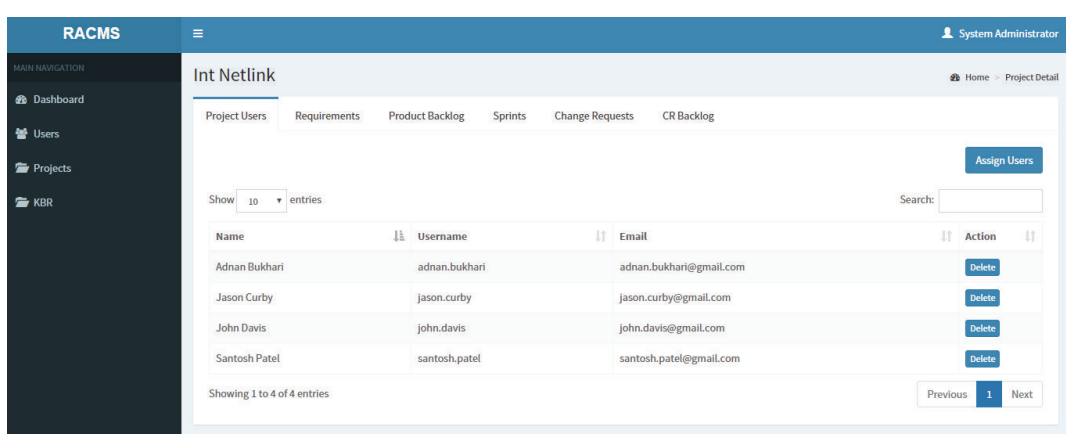

Figure 12: Project users' management menu.

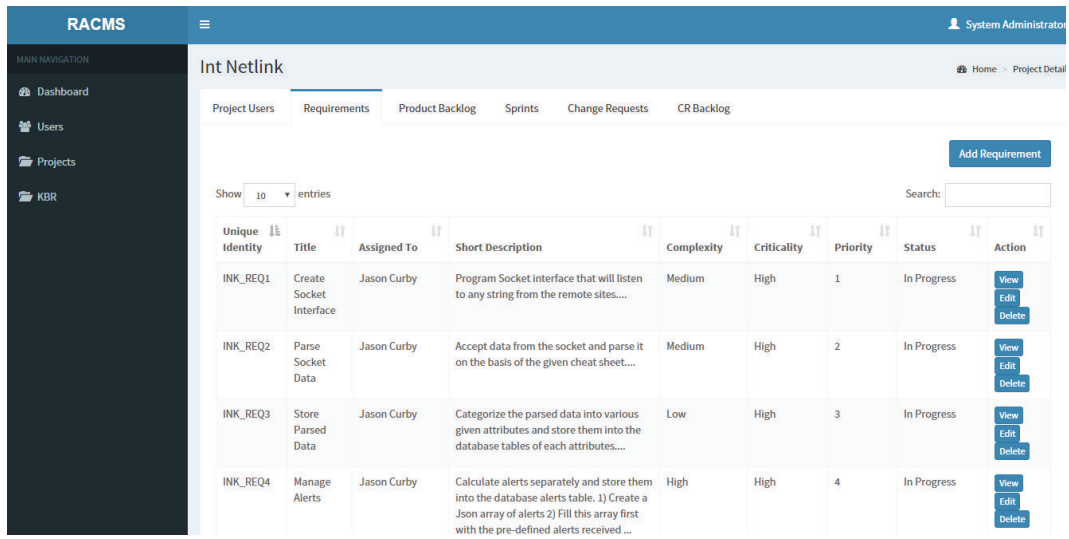

Figure 13: Project requirements' management menu.

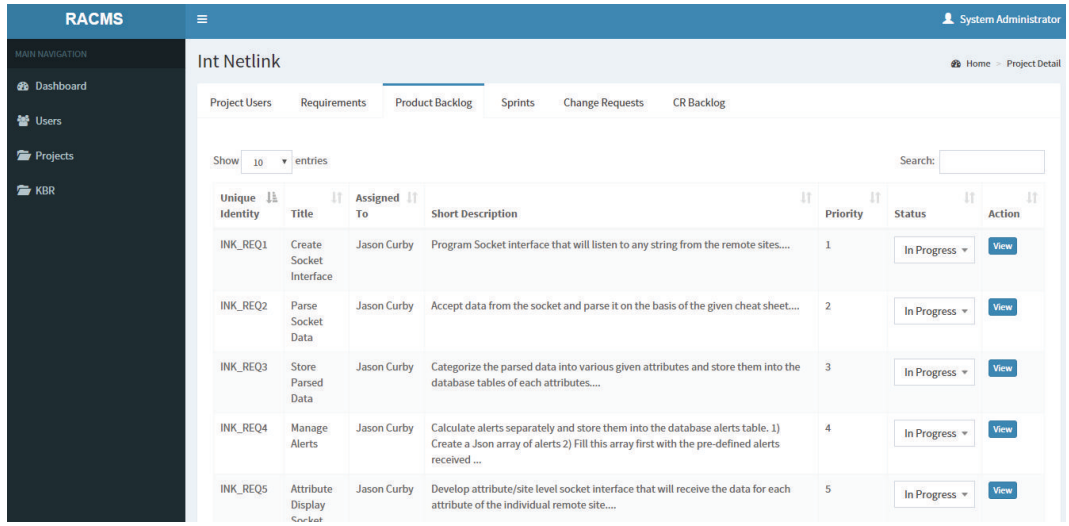

Figure 14: Product backlog.

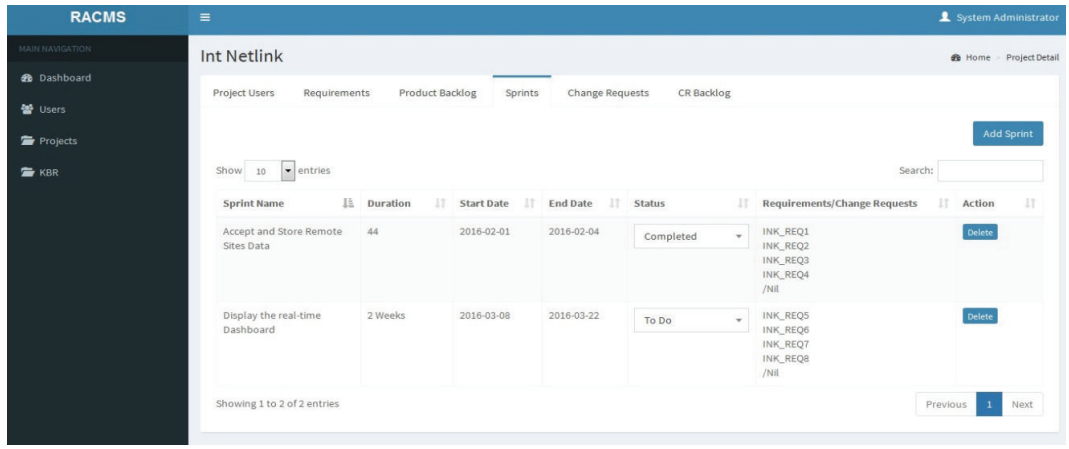

Figure 15: Project sprints.

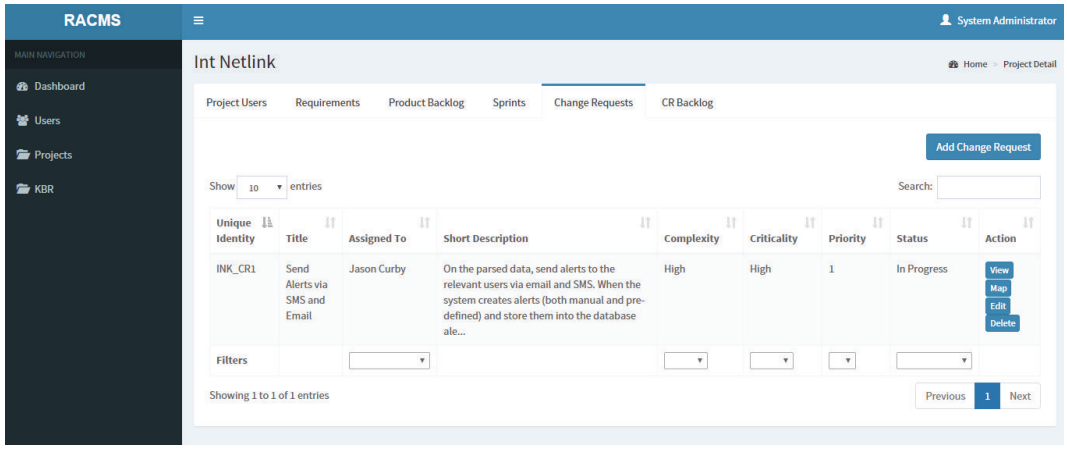

Figure 16: Project change requests' management menu.

(v) *Requirement 5:* deliver the parsed attributed data to socket listeners associated with that site.

(vi) *Requirement 6*: render the data in real-time on the dashboard in the form of gauges and graphs showing the real-time graphical interface.

(vii) *Requirement 7*: Popup in-dash alerts to the user.

5.3.1. Change Occurs in Requirements. During software development, the client was contracted by the telecom company for all telecom towers in Afghanistan (which were 500 towers) with the following problem:

> with 500 remote sites, it is difficult for the client to monitor each remote site.

To solve these problems, the following change request is proposed:

> to generate runtime alerts and send them to the concerned users via SMS and email.

The following change requests were entered into the system.

Change Request 1. On the parsed data, send alerts to the relevant users via email and SMS.

5.3.2. Change Mapping in the Tool. During the initial development phase, all requirements were stored in the system, and a copy was stored in the knowledge repository. After adding this new change, it was mapped with the knowledge repository to determine whether any similar requirement existed previously. This new case matched with an existing requirement, which helped in developing a better strategy toward implementing this change.

Figure 17 shows the change that is entered into the tool. This listed change is the inclusion of an action button next to it labeled "Map," which, when clicked, automatically searches for all similar cases from the knowledge repository and displays the list. Figure 18 shows that, for this particular change, a case titled "Manage Alerts" from the knowledge repository was a match. Figures 19 and 20 show the details of this matched case. Figure 21 shows how the change request is added to the next sprints.

6. Discussion

This research proposed a new approach for handling change management, and the best suited methodology was grounded theory. The issue of global software development was confirmed through interviews from domain experts, among whom requirement change management and its effect on architecture represent a major priority. Interviews allowed us to interact socially with domain experts; hence, it was wise to adopt grounded theory. This was in contrast to other methodologies that tend to restrict the research (e.g., case study or ethnography). The proposed theory was mapped to the six C's coding family proposed by Glaser because it covers all basic and foremost requirements of the theory. The obtained results and innovations are discussed as follows along with their limitations.

6.1. Limitations. Based on our discussions, one major limitation of our study should be noted: we mapped only a single case on our tool. Therefore, this study does not represent a large dataset.

6.2. Theoretical Implications. Regardless of limitations, we also proposed theoretical implications based on an improvised Twin Peaks model alongside agile methodologies, thus making them lightweight. We resolved the issues of agile global software development by incorporating CBR technique and traceability from the requirements to the architecture. Another important implication regarding our research was that the dependency of RCM and the architecture changes simultaneously. Both changes were proved through impact analysis, thus making management universal.

6.3. Practical Implications. Applying the proposed approach to a case study helped in resolving and mitigating challenges of global software development. Similarly, an agile tool based

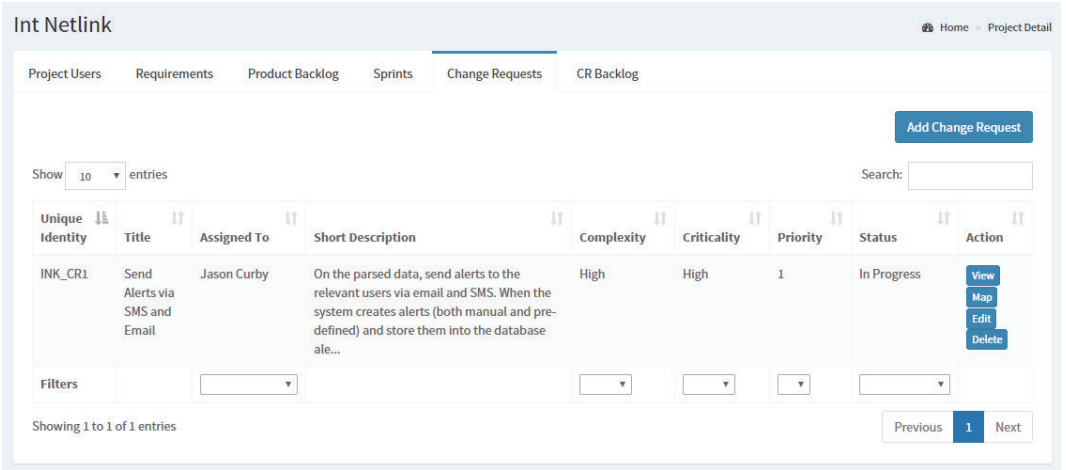

Figure 17: Change request added to the project.

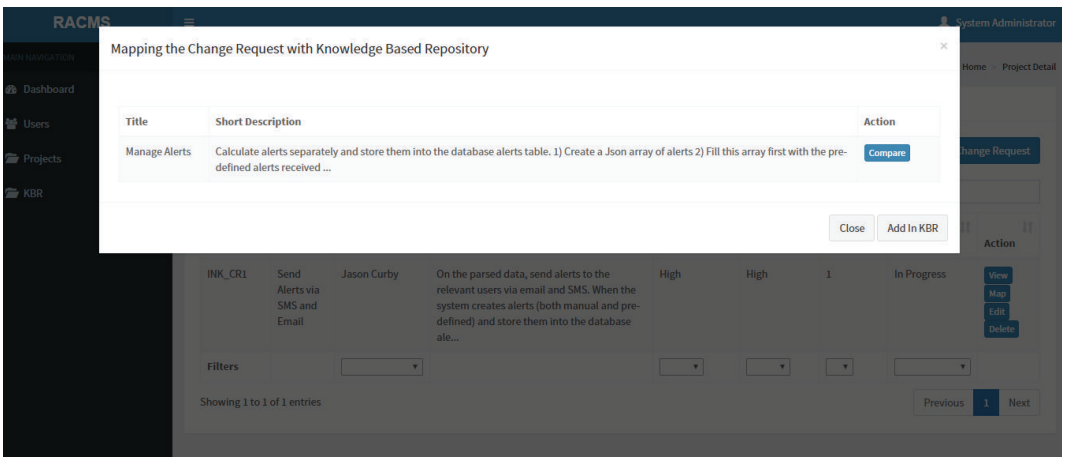

Figure 18: Lists of matched cases from knowledge repository for this change.

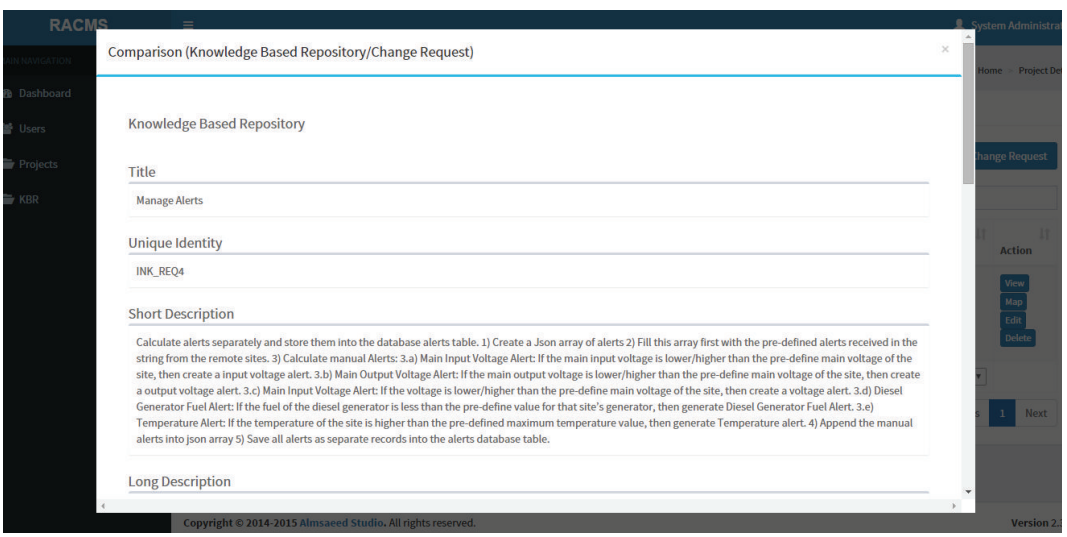

Figure 19: Details of the individual matched case.

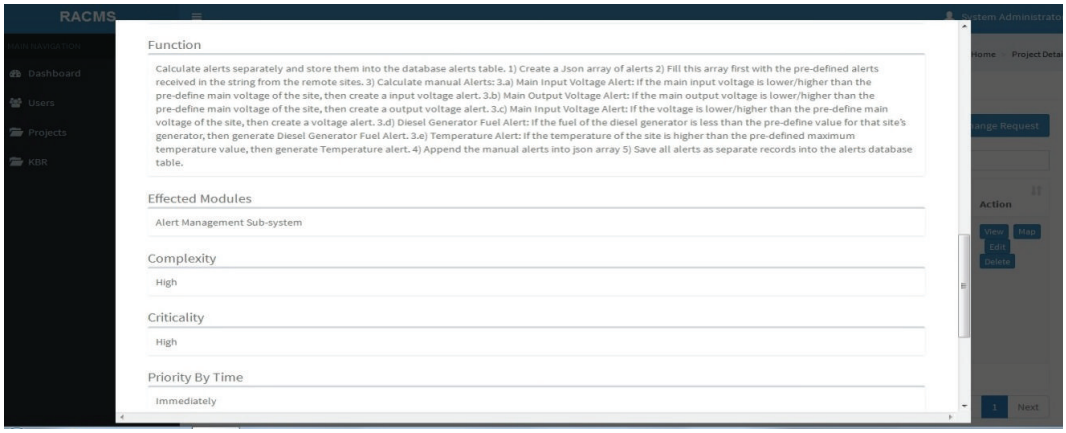

Figure 20: Further details of the individual matched case.

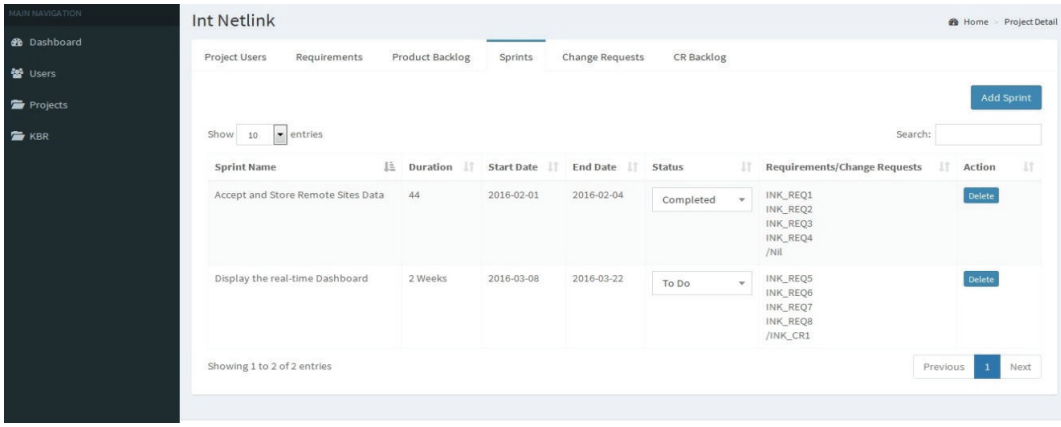

Figure 21: Change request added to next sprint.

on Twin Peaks was developed according to the desired needs. When practically applied to data, it showed that cases were more frequently retrieved and maintained. Change management was achieved at a notable level.

7. Conclusions

The research aimed toward integrating change management at both the requirement and architecture phases simultaneously. The proposed approach made use of agile CBR with the Twin Peaks model. Each requirement from the product backlog was converted to a specific case and was evaluated and stored in the repository as a new case. Associated past experiences and solutions were also stored with the preliminary case. Based on the concepts obtained from agile CBR and Twin Peaks, a generic approach was developed and evaluated through statistical analysis, interviews from domain experts, and the overlapping case study. The results showed that the proposed approach provided better results for managing changes and effectively implementing them.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors are grateful to the Research Centre (RC), College of Computer and Information Sciences (CCIS) for extending to them all categories of support and continuous assistance during this research work. They are also thankful to all those who contributed in any form to make this research successful. Thanks are also due to the Dean of CCIS and the Chairman of the Information Systems Department for their continuous support and encouragement as needed.

References

[1] E. Bjarnason, *Distances between Requirements Engineering and Later Software Development Activities: A Systematic Map*, Springer, Berlin, Germany, 2013.

[2] H. Holmstrom, E. Ó. Conchúir, P. J. Ågerfalk, and B. Fitzgerald, "Global software development challenges: a case study on temporal, geographical and socio-cultural distance," in *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE '06)*, pp. 3–11, IEEE, Florianopolis, Brazil, October 2006.

[3] W. Hussain, "Requirements change management in global software development: a case study in Pakistan," in *Information Systems*, Linnaeus University, 2010.

[4] J. L. Kolodner, "An introduction to case-based reasoning," *Artificial Intelligence Review*, vol. 6, no. 1, pp. 3–34, 1992.

[5] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations, and system approaches," *AI Communications*, vol. 7, no. 1, pp. 39–59, 1994.

[6] M. Ivček and T. Galinac, "Adapting Agile practices in globally distributed large scale software development," in *Proceedings of the International Conference on Telecommunications and Information of the International Convention MIPRO*, 2009.

[7] R. Ferdiana, L. E. Nugroho, P. I. Santoso, and A. Ashari, "Learning from the case studies, how global software development process is executed in an agile method environment," *Jurnal Buana Informatika*, vol. 1, no. 2, 2010.

[8] B. Paech, A. H. Dutoit, D. Kerkow, and A. Von Knethen, "Functional requirements, non-functional requirements, and architecture should not be separated—a position pape," in *Proceedings of the International Workshop on Requirements Engineering: Foundations for Software Quality*, Essen, Germany, 2002.

[9] A. Sutcliffe, "On the inevitable intertwining of requirements and architecture," in *Design Requirements Engineering: A Ten-Year Perspective*, vol. 14, pp. 168–185, Springer, Berlin, Germany, 2009.

[10] J. Pimentel, J. Castro, E. Santos, and A. Finkelstein, "Towards requirements and architecture co-evolution," in *Advanced Information Systems Engineering Workshops*, M. Bajec and J. Eder, Eds., vol. 112 of *Lecture Notes in Business Information Processing*, pp. 159–170, 2012.

[11] P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrík, *Relating Software Requirements and Architectures*, Springer, Berlin, Germany, 2015.

[12] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–119, 2002.

[13] N. Akram and S. Razman, "Requirements change management process models: an evaluation," in *Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering (SE '07)*, Innsbruck, Austria, February 2007.

[14] B. J. Williams, J. Carver, and R. Vaughn, "Change risk assessment: understanding risks involved in changing software requirements," in *Proceedings of the International Conference on Software Engineering Research and Practice*, 2006.

[15] R. Lai and N. Ali, "A requirements management method for global software development," *AIS: Advances in Information Sciences*, vol. 1, no. 1, pp. 38–58, 2013.

[16] A. López, J. Nicolás, and A. Toval, "Risks and safeguards for the requirements engineering process in global software development," in *Proceedings of the 4th IEEE International Conference on Global Software Engineering (ICGSE '09)*, pp. 394–399, IEEE, Limerick, Ireland, July 2009.

[17] S. Patil and R. Ade, "Secured cloud support for global software requirement risk management," *International Journal of Software Engineering & Applications*, vol. 5, no. 6, pp. 23–29, 2014.

[18] H. M. Jani, "Applying Case-Based Reasoning to software requirements specifications quality analysis system," in *Proceedings of the 2nd International Conference on Software Engineering and Data Mining (SEDM '10)*, pp. 140–144, Chengdu, China, June 2010.

[19] A. Tidemann, F. O. Bjørnson, and A. Aamodt, "Case-based reasoning in a system architecture for intelligent fish farming," *Frontiers in Artificial Intelligence and Applications*, vol. 227, pp. 122–131, 2011.

[20] D. Amyot and G. Mussbacher, "Bridging the requirements/design gap in dynamic systems with Use Case Maps (UCMs)," in *Proceedings of the International Conference ojn Software Engineering*, pp. 743–744, May 2001.

[21] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti, "Relating software requirements and architectures using problem frames," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE '02)*, pp. 137–144, Essen, Germany, September 2002.

[22] M. Mirakhorli and J. Cleland-Huang, "Traversing the twin peaks," *IEEE Software*, vol. 30, no. 2, pp. 30–36, 2013.

[23] L. Lingard, M. Albert, and W. Levinson, "Qualitative research: grounded theory, mixed methods, and action research," *British Medical Journal*, vol. 337, no. 7667, pp. 459–461, 2008.

[24] B. G. Glaser, "The future of grounded theory," *The Grounded Theory Review*, vol. 9, no. 2, 2010.

[25] M. Jones and I. Alony, "Guiding the use of grounded theory in doctoral studies industry," *International Journal of Doctoral Studies (IJDS)*, vol. 6, pp. 95–114, 2011.

[26] G. G. Barney and L. Anselm, *The Discovery of Grounded Theory Strategies for Qualitative Research*, 1976.

[27] D. Douglas, *Inductive Theory Generation: A Grounded Approach to Business Inquiry*, 2003.

[28] B. G. Glaser, *Basics of Grounded Theory Analysis: Emergence Vs. Forcing*, 1992.

[29] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Method Approaches*, Sage Publications, Thousand Oaks, Calif, USA, 2003.

[30] B. G. Glaser, *Theoretical Sensitivity*, The Sociology Press, Mill Valley, Calif, USA, 1978.

[31] G. Hole, *Research Skills Mann-Whitney Test Handout*, 2011.