Journal of Liberal Arts and Humanities (JLAH)
Issue: Vol. 4; No. 5; June 2023 (pp. 39-46)
ISSN 2690-070X (Print) 2690-0718 (Online)
Website: www.jlahnet.com
E-mail: editor@jlahnet.com
Doi:10.48150/jlah.v4no5.2023.a3

The Structure-Behavior Coalescence (SBC) Architecture-Driven Model for Enhanced Testability and User-Friendly Design of Smart Service Systems

Shuh-Ping Sun
Department of Digital Media Design
I-Shou University
Kaohsiung, Taiwan

Abstract—This study introduces the structure-behavior coalescence (SBC) Architecture-Driven model to optimize the design of Smart Service Systems, focusing specifically on Smart Parking Service Systems (SPSS). The SBC architecture utilizes the Architecture Description Language (ADL) to effectively design both the overall architecture and detailed components of SPSS. Key ADL diagrams, such as the framework diagram (FD)-ADL, component operation diagram (COD)-ADL, and interaction flow diagram (IFD)-ADL, are utilized during the design phase. In addition to architectural design considerations, the study highlights the significance of improving testability in the successful development of Smart Parking Service Systems. To evaluate the system's testability behavior and identify factors that enhance design efficiency and reduce cognitive load, the study employs the DEA-CCR efficiency evaluation model specifically tailored for SPSS. By incorporating these methodologies and evaluations, the objective is to optimize the performance and usability of the Smart Parking Service Systems while streamlining the testing process. This comprehensive approach ensures that the design of SPSS is efficient, reliable, and user-friendly, meeting the evolving needs and expectations of users in the field of smart parking solutions. The research contributes to the advancement of Smart Service Systems by emphasizing the importance of architecture-driven design and improved testability in optimizing their functionality.

Keywords—Smart Service System; Testability; Architecture-Oriented Design; DEA-CCR Modeling

I. INTRODUCTION

As the numbers of vehicles on the road are increasing day by day parking problems are bound to exist. Parking is costly and limited in almost every major city in the world. A city consists of a group of parking garages. A parking garage consists of a group of parking slots. Searching for a vacant parking slot in a metropolitan area is difficult for most drivers. It commonly results more traffic congestion and air pollution by constantly cruising in certain area only for an available parking slot. For instance, a recent survey shows that during rush hours in most big cities, the traffic generated by cars searching for parking slots takes up to 40% of the total traffic. To alleviate such traffic congestion and improve the convenience for drivers, many Smart Parking Service Systems (SPSS) aiming to satisfy the drivers as well as parking service providers have been deployed [2].

When we talk about “design for testability,” we are talking about the design decisions in order to enable us to easily and effectively test our system [8]. We first must understand the context on which we are writing tests in. There are four main factors to enhance the testability of SPSS.

(A) Disciplined System Layering (DSL). In almost all cases, a single component does not work alone. In SPSS, each component interacts with other components and depends on them to function properly. When writing tests, our ability to isolate the given component from all others dependencies is crucial. And we must think of putting mechanisms in place to enable us to do so easily.

(B) Well-Defined Components (WDC). When we approach writing automatic unit tests, the main difficulty we face is the need to isolate the tested components in SPSS from the rest of it. In order to test a functionality of a component, we first need to detach it from the rest of SPSS in which it is designed to work.

Once the component is there, we then need to activate the tested functionality and finish by making sure the resulting behavior matches our expectations. However, unless SPSS is designed specifically to enable this, in most cases, it will not be simple.

(C) Published Interfaces (PI). When writing automatic unit tests we face a few issues. For example: Creating a component. In most cases a component is not meant to be created in a standalone manner, as we do when writing tests. Normally, components are created as part of an entire system. Since they depend on other components of SPSS, they make sure those components are there and working correctly. Setting these components in the testing environment is expensive and complex. Therefore, we need a mechanism to create the tested component without creating the rest of the dependencies as well.

(D) Well-Defined Behaviors (WDB). In order to write meaningful tests, the expected behavior must be checked. In some cases this behavior can only be observed by looking at the resulting state of the component at the end of the test. However, in many cases, the tested component has no meaningful state of its own, and its purpose is to correctly interact with other parts. In order to verify this interaction we need a way to allow it during testing, making sure all expected interactions were carried out as they should. In order to write effective unit tests for SPSS, we need to effectively isolate each component, and surround it with fakes (created as part of the test), which will enable verification of all interactions carried out by the component under test. The ease of writing unit tests is in direct correlation to this ability.

Architecture-oriented design uses the structure-behavior coalescence (SBC) approach to formally design the integration of systems structure and systems behavior of a system. Architecture-oriented design contains three fundamental diagrams: a) framework diagram, b) component operation diagram, and c) interaction flow diagram. Architecture-oriented approach uses three fundamental diagrams: a) framework diagram, b) component operation diagram, and c) interaction flow diagram to accomplish the design of SPSS. Through framework diagram, architecture-oriented design of SPSS demonstrates tremendous effects of disciplined system layering. Through component operation diagram, architecture-oriented design of SPSS demonstrates large effects of well-defined components and published interfaces. Through interaction flow diagram, architecture-oriented design of SPSS demonstrates tremendous effects of well-defined behaviors.

II. MATERIALS AND METHODS

Architecture-oriented design provides an elegant way to integrate the systems structure and systems behavior of a system. Architecture-oriented approach uses three fundamental diagrams: a) framework diagram, b) component operation diagram, and c) interaction flow diagram to accomplish the design of SPSS.

A. Framework Diagram of Smart Parking Service System

A framework diagram (FD) designs the decomposition and composition of a system in a multi-layer manner. FD is the framework diagram we obtain after the architecture-oriented design is finished. Figure 3-1 shows a FD of the Smart Parking Service System. Through FD, architecture-oriented design of SPSS demonstrates tremendous effects of disciplined system layering (DSL).

In Figure 1, Presentation_Layer and Logic_Layer are sub-layers of Application_Layer. Presentation_Layer contains the Parking_Garages_CityMap_UI, Inquire_Parking_Fees_UI, and Pay_Parking_Fees_UI components; Logic_Layer contains the Parking_Starting_Time_Daemon and Parking_End_Time_Daemon components; Data_Layer contains the SPSS_Database component; Technology_Layer contains the Driver_GPS_P (P = AAA0000 to ZZZ9999), Parking_Starting_Time_Sensor_Q (Q = 000 to 999), and Parking_End_Time_Sensor_R (R = 000 to 999) components.

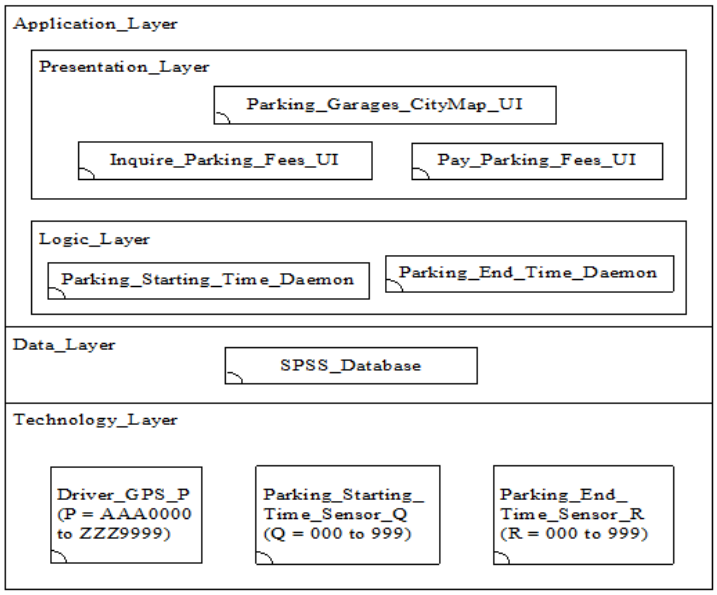


Figure 1 FD of SPSS

B. Component Operation Diagram of Smart Parking Service System

A component operation diagram (COD) designs all components' operations in a system. An operation provided by each component represents a function, procedure, or method of that component. A component should not exist in a system if it does not own any operation.

An operation formula is used to fully define an operation. An operation formula includes a) operation name, b) input parameters, and c) output parameters. Operation name is the name of this operation. In a system, every operation name should be unique. Duplicate operation names shall not be allowed in any system. An operation may have several input and output parameters. The input and output parameters, gathered from all operations, represent the input data and output data views of a system.

COD is the component operation diagram we obtain after the architecture-oriented design is finished. Figure 2 shows a COD of the Smart Parking Service System. Through COD, architecture-oriented design of SPSS demonstrates tremendous effects of well-defined components (WDC) and published interfaces (PI).

In Figure 3-2, component Parking_Garages_CityMap_UI has two operations: Show_Parking_Garages_CityMap and Reserve_One_Parking_Slot; component Inquire_Parking_Fees_UI has one operation: Show_Parking_Fees; component Pay_Parking_Fees_UI has one operation: Pay_Parking_Fees; component Parking_Starting_Time_Daemon has one operation: Fork_PSTD_Process; component Parking_End_Time_Daemon has one operation: Fork_PETD_Process; component SPSS_Database has six operations: SQL_Select_Parking_Garages, SQL_Insert_One_Parking_Slot, SQL_Insert_Parking_Starting_Time, SQL_Select_Parking_Duration, SQL_Insert_Parking_Fees_Payment, and SQL_Insert_Parking_End_Time; component Driver_GPS_P (P = AAA0000 to ZZZ9999) has one operation: Driver_GPS_Positioning; component Parking_Starting_Time_Sensor_Q (Q = 000 to 999) has two operations: Sense_Parking_Starting_Time and Return_Parking_Starting_Time; component Parking_End_Time_Sensor_R (R = 000 to 999) has two operations: Sense_Parking_End_Time and Return_Parking_End_Time.

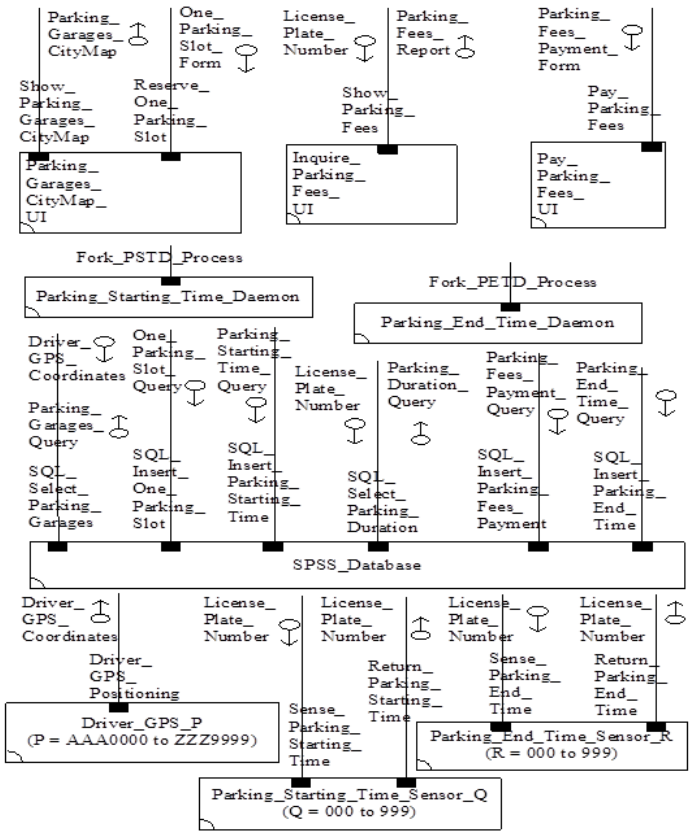


Figure 2 COD of SPSS

C. Interaction Flow Diagram of Smart Parking Service System

The overall behavior of a system is the collection of all its individual behaviors. All individual behaviors are mutually independent of each other. They tend to be executed concurrently. Each individual behavior represents an execution path. An interaction flow diagram (IFD) designs this individual behavior. In a system, if the components, and among them and the external environment's actors to interact, these interactions will lead to the systems behavior. That is, "interaction" plays an important factor in integrating the systems structure and systems behavior for a system. Interaction flow diagrams are the interaction flow diagrams we obtain after the architecture construction is finished. The overall behavior of the Smart Parking Service System includes five individual behaviors: Finding_and_Reserving_a_Vacant_Parking_Slot, Sensing_Parking_Starting_Time, Inquiring_Parking_Fees, Paying_Parking_Fees, Sensing_Parking_End_Time.

Each individual behavior is represented by an execution path. We use an IFD to define each one of these execution paths. Through IFD, architecture-oriented design of SPSS demonstrates tremendous effects of well-defined behaviors (WDB). Figure 3 shows an IFD of the Finding_and_Reserving_a_Nearby_Vacant_Parking_Slot behavior. First, actor Driver interacts with the Parking_Garages_CityMap_UI component through the Show_Parking_Garages_CityMap operation call interaction. Next, component Parking_Garages_CityMap_UI interacts with the Driver_GPS_P (P = AAA0000 to ZZZ9999) component through the Driver_GPS_Positioning operation call interaction, carrying the Driver_GPS_Coordinates output parameter. Continuingly, component Parking_Garages_CityMap_UI interacts with the SPCASIS_Database component through the SQL_Select_Parking_Garages operation call interaction, carrying the Driver_GPS_Coordinates input parameter and Parking_Garages_Query output parameter. Continuingly, actor Driver interacts with the Parking_Garages_CityMap_UI component through the Show_Parking_Garages_CityMap operation return interaction, carrying the Parking_Garages_CityMap output parameter. Continuingly, actor Driver interacts with the Parking_Garages_CityMap_UI component through the Reserve_One_Parking_Slot operation call interaction, carrying the One_Parking_Slot_Form input parameter. Finally, component Parking_Garages_CityMap_UI interacts with the SPCASIS_Database component through the SQL_Insert_One_Parking_Slot operation call interaction, carrying the One_Parking_Slot_Query input parameter.

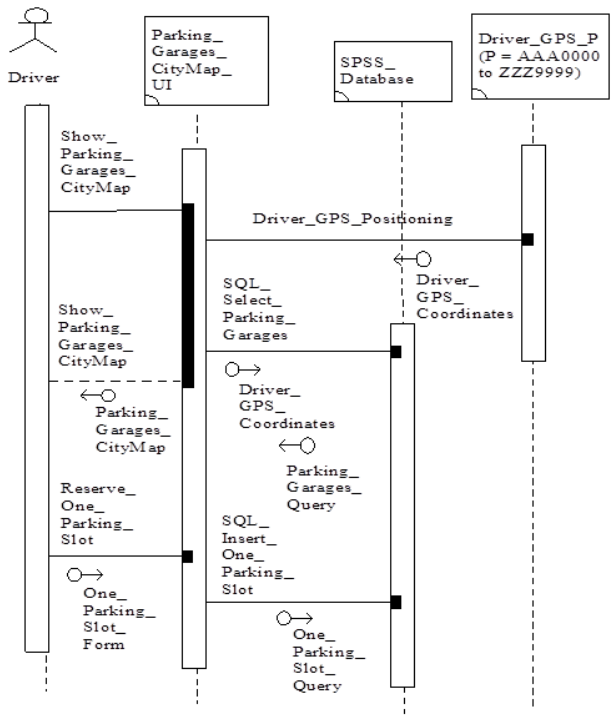
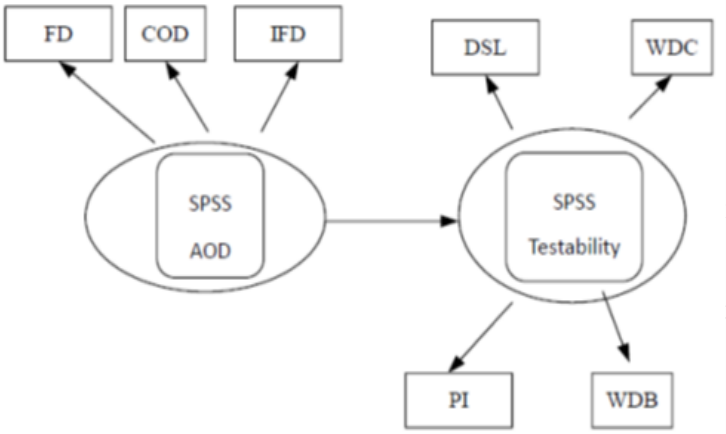


Figure 3 IFD of SPSS

III. RESULTS AND DISCUSSION

VERIFYING THE TESTABILITY IMPROVEMENT OF SPSS

The SPSS-SBC design model is shown in Figure 1 ~ 3. To assess the "testability" of this design model, the schematic diagram is shown as Figure 4.



The decisio ion

The decisio SPSS model. The four output variables are I y" CCR efficiency evaluation procedure is des

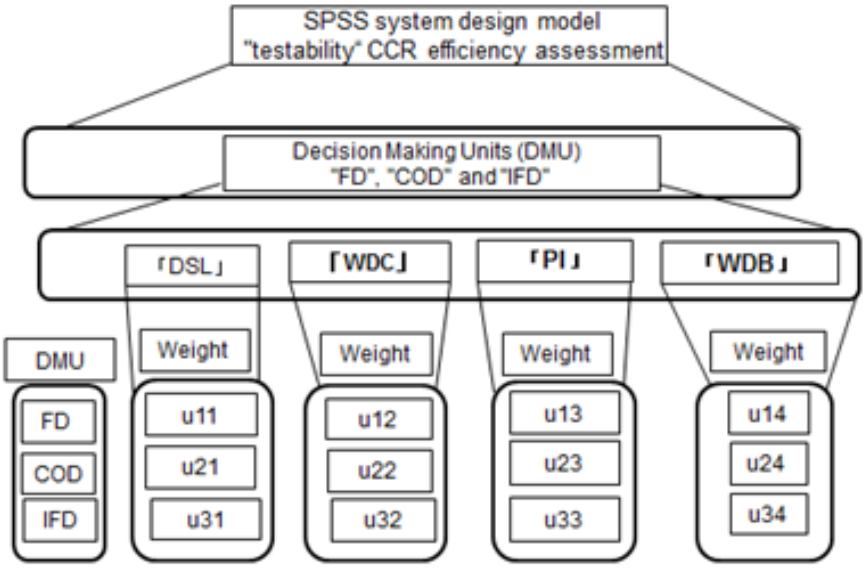


Figure 5 SPSS system design model "testability" CCR efficiency assessment

Charnes, Cooper and Rhodes (1978) proposed the CCR efficiency evaluation model (DEA-CCR model) first. The main idea of this efficiency assessment model is to figure out a reasonable set of input and output variables of the weight. The CCR model is to assess a system (DMU), using the most important weight value to enhance the system model. This study planning the CCR efficiency evaluation model of SPSS system "testability" is shown below. The input variables of the system are fixed to 1, because it is fixed as SBC-ADL. The output variables are the "decision making units (DMU)" of the SPSS system model "FD", "COD", and "IFD". Regardless of the input variables, so input variables can be set to "1", this time the estimated efficiency value is equal to the performance value. The expert questionnaire was designed after the SPIS-CCR efficiency evaluation plan was integrated. Under the Likert 1-5 scale, n experts were asked to rate the importance of the four variables and to conduct an SPSS system design model "testability" CCR efficiency evaluation.

In this study, 10 experts completed the SPSS system design model expert assessment questionnaire. Through the above-mentioned CCR efficiency evaluation model, the SPSS system design model "testability" of FD, COD and IFD respectively evaluates the efficiency Θ, and generate the system's "testability" performance and the weight of each output variable ur.

The mathematical equation of input-oriented CCR model of this study is programming as follows:

$$Max.h_k = \sum_{i=1}^{s} u_{rk} y_{rk} \qquad (1)$$

$$\sum v_{ik} x_{ik} = 1$$

$$\sum u_{rj} y_{rj} - \sum v_{ij} x_{ij} \le 0$$

$$u_{rj}, v_{rj} \ge 0,\ j = 1,\ldots\ldots,n$$

hk：The relative efficiency of the k-th DMU.

yrk：The r-th output item of the k-th DMU.

xik：The i-th entry of the k-th DMU.

yrj：The r-th output item of the j-th DMU.

xij：The i-th input item of the j-th DMU.

urk：The weight value of the r-th item of the kth DMU.

vik：The weight value of the i-th input of the kth DMU.

The target values(Θ) of Eq. (1), which are the efficiency value to be evaluated. In this study each input item has been set to "1". Eq. (1) has become a convenient calculation of the linear programming model.

The corresponding output variables and output weights of the decision making units (DMU) are shown as Table 1.The decision making units (DMU) and the corresponding table provide valuable insights into the importance of various output items. When the value uij is 0, it indicates a lower importance of the corresponding output item. By referring to this table, users can determine the effectiveness of FD, COD, and IFD in presenting the Smart Parking Service Systems (SPSS) system design model's "testability".

Based on the Table 1, it can be observed that FD effectively presents the SPSS system design model's DSL (Design Structure Language), COD effectively presents the WDC (Weighted Design Complexity), and PI (Performance Index), and IFD effectively presents the WDB (Weighted Design Behavior). This implies that each of these decision making units contributes effectively to the representation of the SPSS system design model's "testability".

Table 1 Decision Making Units (DMU) and output weight corresponding table

Output DMU	**DSL**	**WDC**	**PI**	**WDB**	**Θ**
FD	$u_{11=0.1402}$	$u_{12=0.1402}$	$u_{13=0.1402}$	$u_{14=0.1184}$	**Θ1=1.0**
COD	$u_{21=0.0}$	$u_{22=0.2222}$	$u_{23=0.2222}$	$u_{24=0.0}$	**Θ2=1.0**
IFD	$u_{31=0.0}$	$u_{32=0.0}$	$u_{33=0.0}$	$u_{34=0.2326}$	**Θ3=1.0**

Overall, the SPSS-SBC system design model demonstrates a highly optimized level in terms of "testability" within the system. These findings highlight the effectiveness of the SBC architecture-oriented design approach in achieving a robust and efficient SPSS system design.

IV CONCLUSION AND FUTURE WORKS

The research findings, as depicted in Table 1, present compelling evidence that supports all hypotheses concerning the Decision Making Units (DMU) in Smart Parking Service Systems (SPSS). The results underscore the significant relationships between the decision making units, namely the Framework Diagram (FD), Component Operation Diagram (COD), and Interaction Flow Diagram (IFD), and the output variables related to "Testability" (User-Friendly) efficiency. Notably, the study demonstrates the positive influence exerted by these decision making units (FD, COD, IFD) on the testability and user friendly of SPSS.

The importance of the SPSS system design model on improving "testability" efficiency has gained increasing recognition. The effects of the Framework Diagram (FD), Component Operation Diagram (COD), and Interaction Flow Diagram (IFD) on "testability" efficiency within the context of the SBC architecture-oriented design method cannot be underestimated. These findings contribute to a deeper understanding of the intricacies involved in service system design and the quest for enhanced performance.

The research outcomes underscore the critical significance of emphasizing the development and design of the SBC architecture-oriented design model. These findings serve as a guiding paradigm for future studies on service system design, recognizing that the challenges in designing service systems go beyond mere system design and encompass the comprehensive cycle of system development.

As future research progresses, it is recommended to further explore service system design, particularly in the context of developing architecture-oriented design models. This will enable the attainment of greater efficiency in system design, reduction of cognitive load, and ultimately facilitate more effective and streamlined service system development processes.

References

[1] Bagozzi R.P.; & Yi, Y. On the evaluation of structural equation models. Journal of the Academy of Marketing Science, 16, 1988; 74–94.

[2] Bi, Y. et al., "A Parking Management System Based on Wireless Sensor Network," ACTA AUTOMATICA SINICA, Vol. 32, No. 6, 2006, pp. 38-45.

[3] Chao W. S. General Systems Theory 2.0: General Architectural Theory Using the SBC Architecture, Create Space Independent Publishing Platform, 2014.

[4] Chao W. S. Variants of SBC Process Algebra: The Structure-Behavior Coalescence Approach, Create Space Independent Publishing Platform, 2015.

[5] Chao W. S. System: Contemporary Concept, Definition, and Language, Create Space Independent Publishing Platform, 2016.

[6] Chao W. S. Generalized SBC Process Algebra for Communication and Concurrency: The Structure-Behavior Coalescence Approach, Create Space Independent Publishing Platform, 2016.

[7] Cronbach L.J. Coefficient alpha and the internal structure of tests. Psychometrik,16,1951; 297–333.Title of Site. Available online: URL (accessed on Day Month Year).

[8] Evans M.W. et al. Software Quality Assurance & Management, Wiley-Interscience, 1987.

[9] Fornell C Hair J.F.; Anderson R.E., Tatham R.L.; and Black W.C., Multivariate data analysis with readings, 1998; (4th ed), Prentice Hall; New Jersey. and Larcker D.F. Evaluating structural equation models with unobservable variables and measurement error, Journal of Marketing Research, 18 (1), 1981; 39–50.

[10] Gharajedaghi J. Systems Thinking: Managing Chaos and Complexity: A Platform for Designing Business Architecture, Morgan Kaufmann, 2011.

[11] Hair J.F.; Anderson R.E. Tatham R.L; and Black W.C., Multivariate data analysis with readings, 1998; (4th ed), Prentice Hall; New Jersey.

[12] Hoare C. A. R. Communicating Sequential Processes, Prentice-Hall, 1985.

[13] Joreskog K.G.; and Sorbom D. LISREL 8: Structural equation modeling with the SIMPLIS command language, 1993; Scientific Sotware International; Chicago.

[14] Kendall K. et al. Systems Analysis and Design, 8th Edition, Prentice Hall, 2010.

[15] Kishen Iyengar.; Jeffrey R. Sweeney.; and Ramiro Montealegre Kishen. Information technology use as a learning mechanism: the impact of it use on knowledge transfer effectiveness, absorptive capacity, and franchisee performance, MIS Quarterly Vol, 2015; 39 No. 3, pp. 615-641/September.

[16] Milner R. Communication and Concurrency, Prentice-Hall, 1989.

[17] Milner R. Communicating and Mobile Systems: the π-Calculus, 1st Edition, Cambridge University Press, 1999.

[18] Pressman R. S. Software Engineering: A Practitioner's Approach, 7th Edition, McGraw-Hill, 2009.

[19] Scholl C. Functional Decomposition with Applications to FPGA Synthesis, Springer, 2010.

[20] Sommerville I. Software Engineering, 8th Edition, Addison-Wesley, 2006.