Chapter - 3

Software Component Classification Framework

3.1 Overview

Aim of this chapter is to develop comprehensive strategy for the classification of software components. For this, six dimensional classification strategy framework is developed. It consists of dimensions such as *architecture level*, *domain*, *phase*, *source*, *kind* and *functionality* of software components. The rest of the chapter is organized as follows: section 3.2 provides introduction to the classification of software components. Section 3.3 elaborates the concept of six dimensional classification strategy framework. Section 3.4 explores the framework by referencing it with the practical usage. Section 3.5 deals with the validation of SDCS framework and also presents the overall discussion on the developed SDCS framework. Finally section 3.6 provides concluding remarks of the chapter.

3.2 Introduction

Traditional software development approaches fail in cost effective, just-in-time to market and easily maintainable software components. The context of software component based development has become very important in industry and research (John and Andre, 2002). Component based development can be used potentially to reduce software development and maintenance costs. Using CBSD, software systems can be built by two techniques. The first techniques involve integration of software components with the existing system while with the second technique a whole new system can be developed by identifying and integrating appropriate software components. Various classification strategies have been evolved and identified such as taxonomies of application domains (Glass and Vessey, 1995), cartesian space based attributes (Carney and Long, 2000), integration effort for software components (Egyed et al., 2000), origin and modifiability attributes (Carney and Long, 2000), supplier and market conditions by COCOTS model (Abst et al., 2000**)**, on the basis of delivered system (Carney, 1997) and software component solution and intensive system (Wallnau et al., 1998).

There is little concern for the broad classification within and across the domain taxonomies (Glass and Vessey, 1995). The existing literature is not comprehensive to deal with the classification of software components as they concentrate on specific

features/attributes. The intention of the chapter is to develop comprehensive strategy for the classification of software components in order to explore, learn, assess, compare and evaluate software components. For this, six dimensional classification strategy framework is proposed. It consists of dimensions such as *architecture level, domain, phase, source, kind* and *functionality* of software components.

3.3 SDCS: Six Dimensional Classification Strategy Framework

A six dimensional classification strategy framework based on the following broad and comprehensive dimensions is defined: *architecture level (A), domain (D), phase (P), source (S), kind (K)* and *generic functionality (G)* of software components. These dimensions are exhaustive, still the industries/researchers are free to identify more dimensions depending upon their project goals and requirements. Each dimension is the basis for the classification of software components. A SDCS web can be created on the basis of project goals and requirements (shown in Figure 3.1), which depicts how and on what basis software components can be acquired and used.

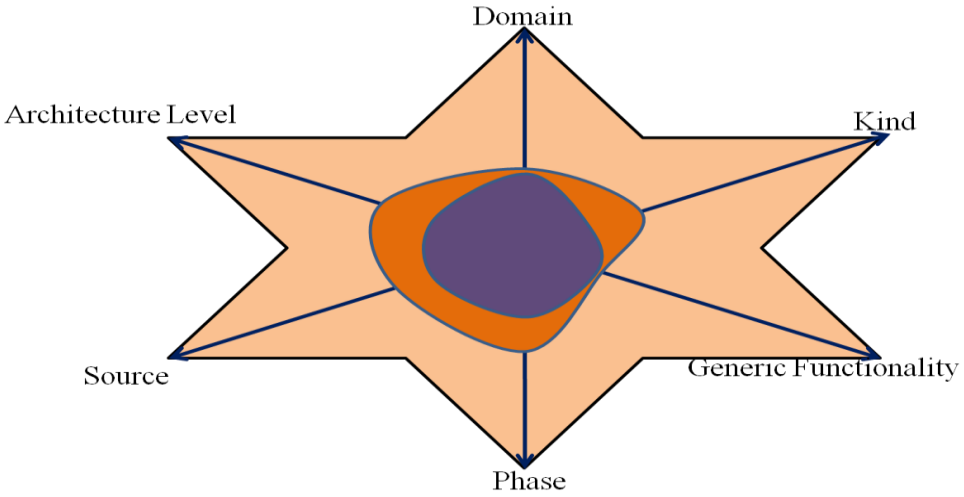

Figure 3.1 Six dimensional classification strategy (SDCS) Web

In Figure 3.1 each point on a dimension reflects specific attribute. For example, the innermost web points for *A, D, K, G, P,* and *S* could reflect *client-server pattern, finance, services, horizontal, execution,* and *commercial* respectively.

The SDCS is comprehensive and the classification leads to the investigation of software components. The classifier can have many views for this dimensional classification. This helps in understanding relationships between classes, its usage and specification. The dimensional discussion is as follows:

- **Architecture level**: It describes the architectural pattern such as client-server, blackboard, control-loop, peer-to-peer, distributed, event-based etc., and also the role that each software components plays. For example, in 3-tier client-server architectural pattern, Figure 3.2, firstly, software component acts as a client when it requests service, secondly, it acts as server when it serves request, and thirdly, it acts as data when it provides data support.

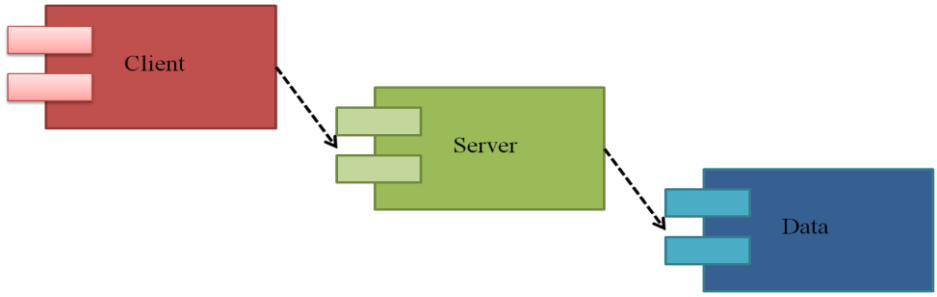

Figure 3.2 3-tier architectural pattern

- **Domain:** Various taxonomies for application domains have been proposed (Glass and Vessey,1995; ISO/IEC, 1999) and the most important ones are IBM, Digital and Reifer (IBM, 1998; Reifer, 1990; AFIPS, 1980). The advent of network infrastructures, information technology and handheld devices has a major impact on mobile application (Upadhyay, 2006) and it is seen and included in the domain taxonomies (current domain is taken as Education). Table 3.1 is an extension of the domain classification (Kotonya et al., 2003)**.**

Domain	**Application**
Avionics	Air traffic control, Electronic warfare
Command and Control	Space, Satellite, Other
Embedded Systems	Operating systems, I/O controllers, ASIC, Other
Electronic Commerce	Agents, Brokerage, Electronic data interchange
Finance	Accounting, Banking, Insurance
Healthcare	Emergency care, Home care, Primary care
Education	Adaptive, Context awareness, Mobile, Other
Real-time	Controllers, Sensors, Signal processors
Simulation	Environmental simulators
	War-gaming
Telecommunications	Network management
	Network engineering
Utilities	Transmission, distribution, marketing and Retailing functions of electric, water and gas Utilities.

Table 3.1 Application domains (extension of Kontony et al., 2003)

- **Kind:** It consists of four main attributes *packaging, delivered, customized* and *size*. The software component can be packaged in different ways. Possible values for this attribute are: *executables, standards* and *services.* This can be further understood as *source code, statically linkable binary library, dynamically linkable library, binary component* and *stand-alone executable program*. Packaging is a form in which the software component is used. It is to be noted that a standalone program does not preclude access to the source code. A *delivered* attribute identifies whether any software component is shipped with software product (as product's integral part) or not. For example if we consider software to be delivered to a customer is made up of C++ language then the delivered software product will not include C++ compiler. However, some tools usually associated with the C++ compiler (e.g. the library of I/O functions) are probably integrated in the final product. Possible values for this attribute are: *integrated* or *separate*. A *customized* attribute is based on the lines of Carney and Long (2000) considering the *modifiability* attribute. But here the attribute *customized* is spilt up into two basic attributes *mandatory modification* and *desirable modification.* The mandatory modification corresponds to the modifiability dimension proposed in (Carney, 1997). If a source code is available then *modification* can be achieved by performing *extensive reworking* or just *internal code revision*. In case software component is a black box then modification is achieved by using inbuilt mechanism provided by software component for *modification. Desirable modification* refers to the internal possible customization of the software component. Such kind of modification is not required by the software component to deliver its basic functionality. For example, the open source web server Apache typically requires only simple parameterization, although its source code is accessible making any in-depth *modification* possible. This can be achieved by doing *modification* on source code, API or interfaces, or by defining macros or configuration files and including certain level of *parameterization* i.e. parameters can be defined for the product so that it can achieve certain level of *customization*. Last attribute *size,* is an important factor of software component. It can have three possible values: *small* (S), *medium* (M) and *large* (L) in terms of MB.
- **Generic functionality:** It is basically divided in to two main attributes – *horizontal* and *vertical*. In *horizontal* attribute, functionality can be reused in various domains such as DBMSs, GUIs, networking protocols, web browsers etc. In *vertical* attribute,

functionality is by no means reused in various domains but rather specific to a particular domain (e.g. Financial Applications, Accounting, Enterprise Resource Planning, Manufacturing, Health Care Management, and Satellite Control). It is to be noted that there is less risk in the usage of *horizontal* software component as compared to *vertical* software component as these have been available on the market for a long time and information about them is widely available.

- **Phase:** It identifies the phase of system life cycle where software component is used (*development* or *execution*).
- **Source:** It depicts the origin of software component and the way to get it. The software component can come from: *in-house*, *existing external*, *externally developed*, *special version of commercial*, *independent commercial* and *open*. Software component can also be freely available for usage or one may have to pay fee to use it. Obtaining it for its usage could certainly means acquiring the source code or executable code. For the *open* source software component, source code is freely available. This can be tailored or customized according to its usage in the domain. But for software component where fee is applicable it means that while acquiring the software component, ownership of the product (including source code) is transferred to the acquirer. In order to use the product, the acquirer has to pay use/license fee. The other factors such as - legal / commercial issues for software component defects, maintenance strategies and concerns, and export restrictions also matters to use the software component.

3.4 SDCS Framework in Practice

This section illustrates how the SDCS framework works in practice. Various software components have been identified and put across each classification dimensions for comprehensive study. The dimensions are covered exhaustively and are further categorized on the basis of possible values that they can have. The proper understanding and knowledge of software components can be depicted in Table 3.2. Specific class name are given to the family of homogeneous software components such as: *server side languages* (SSL), *server side engines* (SSE), *database management system* (DBMS), *client side languages* (CSL), *client side engines* (CSE), *programming languages* (PL), *development environment* (DE) and *executable components* (EC). Two possible values have been identified for software components – '*Y*' and '*N*', which reveals whether the product belongs to classification

dimension or not respectively. If a product belongs to classification dimension then further categories are associated with it. For example: *origin* is categorized as *in-house* (H), *free* (F), *commercial* (C), *externally developed* (ED) and *open* (O); *packaging* is categorized as *executable* (E), *standards* (ST) and *services* (SE); *delivered* has two categories- *separate* (SA) and *integrated* (IN); and *generic functionality* is categorized further as *horizontal* and *vertical*. Table 3.2 demonstrates those categories that are applicable to class names. Table 3.2 has been simplified to model only limited categories in each case.

The dimensional classification leads to number of classes. Calculating SDCS webs for each specific values of dimension can identify classes. This generates the overall structure of viewing homogeneous software components. It helps in achieving know-how about the software components- specification, usage and knowledge. Thus it provides a generic framework to assess, compare and evaluate software components. It might be possible for software component to have no value on dimensions. This will generate innumerable classes and thus increase complexity. For this reason these software components values have been treated as not applicable and ignored for further assessment in class calculations.

Classification		**SSL**	**SSE**	**DBMS**	**CSL**	**CSE**	**PL**	**DE**	**EC**
		SOAP	**Oracle Application Server**	**Oracle**	**HTML**	**Acrobat Reader**	**C++**	**Microsoft Windows XP**	**MS Chart Control**
Architecture Level	Server	Y	Y	N	N	N	N	N	N
	Client	N	N	N	Y	Y	N	N	N
	Data	N	N	Y	N	N	N	N	N
Domain	Avionics	N	N	N	N	N	N	N	N
	Embedded Systems	Y	Y	Y	N	Y	Y	N	N
	Electronic Commerce	Y	Y	Y	Y	Y	Y	Y	Y
	Education	Y	Y	Y	Y	Y	Y	Y	Y
	Finance	Y	Y	Y	Y	Y	Y	Y	Y
	Health Care	Y	Y	Y	Y	Y	Y	Y	Y
	Real-Time	Y	Y	Y	Y	Y	Y	N	N
	Simulation	Y	Y	Y	Y	Y	Y	Y	Y
	Telecomm--unications	Y	Y	Y	Y	Y	Y	Y	Y
	Utilities	Y	Y	Y	Y	Y	Y	Y	Y
	Other	--	--	--	--	--	--	--	--

Continued…

Classification		SSL	SSE	DBMS	CSL	CSE	PL	DE	EC
		SOAP	**Oracle Application Server**	**Oracle**	**HTML**	**Acrobat Reader**	**C++**	**Microsoft Windows XP**	**MS Chart Control**
Phase	Development	N	N	N	N	N	N	N	N
	Execution	Y	Y	Y	Y	Y	Y	N	Y
Source	Origin	Y (C)	Y(C)	Y(C)	Y(C)	Y(C)	Y(C)	Y(C)	Y(C)
Kind	Packaging	Y (ST)	Y (E)	Y (E)	Y (E)	Y (E)	Y (E)	Y (E)	Y (E)
	Delivered	Y	Y (SA)	Y (SA)	Y	Y (SA)	Y (SA)	Y (SA)	Y (SA)
	Customized	N	Y	Y	N	N	N	Y	Y
	Size	Y (M)	Y (L)	Y (L)	Y (S)	Y (M)	Y (M)	Y (L)	Y (M)
Generic Functiona--lity	Horizontal	Y	Y	Y	Y	Y	Y	Y	Y
	Vertical	N	N	N	N	N	N	N	N

Table 3.2 SDCS for software components

On the basis of critical review various software components have been identified, which are grouped under specific class names. Table 3.3, based on **(**Jaccheri and Torchiano, 2001)**,** gives the broad categorization of software components on the basis of class names. The proliferation of information technology and internet has enabled quick and rapid launch of new software components, which makes software components obsolete soon. Special care has been taken to collect most prominent software components. The class based categorization helps in identifying different software components. Thus a classifier can gain knowledge, learn, assess, evaluate and compare software components. On the same line of Table 3.2, Table 3.3 can also be further refined and classified along six dimensions for better perception.

SSL	SOAP, CORBA, Perl, Java Language, SMIL, MS ASP, Java Servlet, Java Beans, Java RMI, Java Server Pages, MS DCOM, CGI, CORBA IIOP spec., Java EJB, Ada Language, Java Connector, Java Message Queue, Java NDI, Java SSE, PHP, RPC, SSH, XQL
SSE	Oracle Application Server, Orion Application Server, Sybase Adaptive Server, IBM HTTP Server, MacroMedia ColdFusion, Appache HTTP Ser., Jigsaw, MS Biztalk Ser. 2000, MS Exch. 2000, ORBacus
DBMS	Oracle, IBM DB2, Sybase, MS SQL Server, MS Access, Borland Interbase, Clustra, MSProj. Central, MySQL, Sybase Indus. Warehouse
CSL	HTML, XHTML, Java Applet, Dynamic HTML, WML, CSS, Java ME MIDP, MS Pocket PC Jscript, Java Phone, Java Script, MacroMedia ColdFusion ML, MathML, WebTV

Continued…

CSE	Acrobat Reader, Winamp, Opera, MS IE, Lynx, Fetchl, MacroMedia ShockWave, NeoPlanet, Java ME Runtime, Java Plugin, Netscape Communicator, MS Pocket PC IE, Palm Reader
PL	C++, Mobile Access, Java Speech, MS ActiveX, XML DTD
DE	Microsoft Windows XP, IBM OS/2, MS Windows X, MS Windows NT
EC	MS Chart Control, MS Excel, MS Office, MS Word, MS Office XP, MS Outlook, Java VM, MS Powerpoint

Table 3.3 Software components Categorization (based on Jaccheri and Torchiano, 2001)

3.5 Validation and Discussion

A survey in three phases was conducted to show the validity of the developed SDCS framework and its applicability to software components. To get a broad view of the validation, sixty three persons were selected and divided into three groups - *Researchers*, *Academicians* and *Practitioners*. The validation was done in three phases with the intention to get feedback on the varying interpretations and perceptions of the classification as well as its usefulness. The candidates in the *Research* group were active in component oriented domain and had published and presented their research work in conferences and journals. The group members for *Academician* group were senior professors who have had long experience in teaching and mentoring courses and projects in component oriented domain and related areas. The third group consisted of all those who practice component philosophy, terminologies and taxonomies in their day-to-day job as they all work in component oriented project in software industries. The survey was conducted in three phases. Phase I consisted of 13 members, Phase II comprised 23 members and Phase III included of 27 members. It is to be noted that the final groups that were identified for each phase are mixture of candidates from *Research*, *Academician* and *Practitioners* Groups. Each person had to answer the questionnaire (*Appendix A*) given to him/her with a rating pattern to each question. The confidence level measure was also associated with the answer to the questions. The confidence was marked from 1 to 5 where 1 was no confidence and 5 was great confidence. Table 3.4 to Table 3.6 presents the results of the survey performed with the *Researchers*, *Academicians* and *Practitioners*. The software component was considered to be within SDCS framework and respective class name, only if the 60% of surveyed people were in agreement. For disagreement the result could be either part or not part of the classification.

Characteristics	Group 'R'							Group 'A'							Group 'D'							Level of Agreement %
	R_1	R_2	R_3	R_4	R_5	R_6	R_7	A_1	A_2	A_3	A_4	A_5	A_6	A_7	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
Architecture Level	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	100
Domain	y	y	y	n	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	95.2
Phase	y	y	n	y	y	n	y	y	y	y	y	y	y	y	y	y	y	y	n	y	y	85.7
Source	y	y	y	y	y	y	y	y	y	y	y	n	y	y	y	y	y	y	y	y	y	95.2
Kind	n	y	y	y	n	y	y	y	y	y	y	y	y	y	y	y	n	y	y	y	y	85.7
Generic Functionality	y	n	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	95.2
Survey Phase	I	I	I	II	II	II	II	I	I	I	I	II	II	II	I	I	I	I	I	I	II	
Level of Agreement %	83.3	83.3	83.3	83.3	83.3	83.3	100	100	100	100	100	83.3	100	100	100	100	83.3	100	83.3	100	100	

Table 3.4 Level of agreement for SDCS framework Group 1

Characteristics	Group 'R'							Group 'A'							Group 'D'							Level of Agreement %
	R_1	R_2	R_3	R_4	R_5	R_6	R_7	A_1	A_2	A_3	A_4	A_5	A_6	A_7	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
Architecture Level	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	100
Domain	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	n	y	y	y	y	y	95.2
Phase	y	y	y	y	y	y	y	y	y	y	y	n	y	y	y	y	n	y	y	y	y	90.4
Source	y	y	y	y	y	y	y	y	n	y	y	y	y	y	y	y	y	y	y	y	y	95.2
Kind	y	y	y	y	n	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	95.2
Generic Functionality	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	100
Survey Phase	II	III	II	III	III	III	III	II	II	III	III	III	III	III	II	II	II	II	II	II	II	
Level of Agreement %	100	100	100	100	83.3	100	100	100	83.3	100	100	83.3	100	100	100	83.3	83.3	100	100	100	100	

Table 3.5 Level of agreement for SDCS framework Group 2

It can be seen that all the group members appreciated the classification framework by putting high level of agreement, where a high level in agreement indicates that all have same opinion when it comes to classification of the software components.

Characteristics	Group 'R'							Group 'A'							Group 'D'							Level of Agreement %
	R_1	R_2	R_3	R_4	R_5	R_6	R_7	A_1	A_2	A_3	A_4	A_5	A_6	A_7	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
Architecture Level	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	n	y	100
Domain	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	100
Phase	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	100
Source	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	100
Kind	n	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	n	90.4
Generic Functionality	y	n	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	95.2
Survey Phase	III	III	III	III	III	III	III	III	III	III	III	III	III	III	II	II	II	III	III	III	III	
Level of Agreement %	83.3	83.3	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	83.3	83.3	

Table 3.6 Level of agreement for SDCS framework Group 3

The developed *SDCS* framework matched with the answers of all the 63 people. This result shows the in depth coverage of software component classification and understanding of the software component terminologies. The *SDCS* framework is comprehensive and has been identified after critical review. It can be concluded that the *SDCS* framework provides the appropriate classification to learn, assess, compare and evaluate software components. In the first phase of the survey, group members were asked to fill up Part II(a). Based on the responses generated in Part II(a), group members proceeded to fill up other sets of questions ranging from Part II(b) to Part II(d). The survey result Part II(b) to Part II(d) is shown in *Appendix A.1*. Part II(b) was designed to assess the difficulty level that was created in Part II(a). At the same time it also adjudicating the confidence level of the group members in filling Part II(a). *Practitioners* found it very easy to fill Part II (a) while some of the members from group *Academician* found little difficulty in filling up Part II (a), see Figure 3.3, as they were not aware of some of the software components and terminologies. *Researchers* found *SDCS* framework easy to use and comprehend. The question in Part II (c)

dealt with the goodness of the model in classifying software components. All most all members had given high ratings for this and appreciated the model, see Figure 3.4. The last question i.e. Part II (d) was aimed at yielding information that included overall satisfaction of the group members as regards the usage of *SDCS* framework and their respective confidence level in understanding it. By giving high rating to satisfaction level and confidence level, see Figure 3.5, both the *Researchers* and *Practitioners* showed their willingness to use *SDCS* framework in their future projects and research. While the *Academicians* showed willingness to include *SDCS* framework in their mentoring and tutorials with an aim to enrich the quality of research and content required in the software component classification and usage.

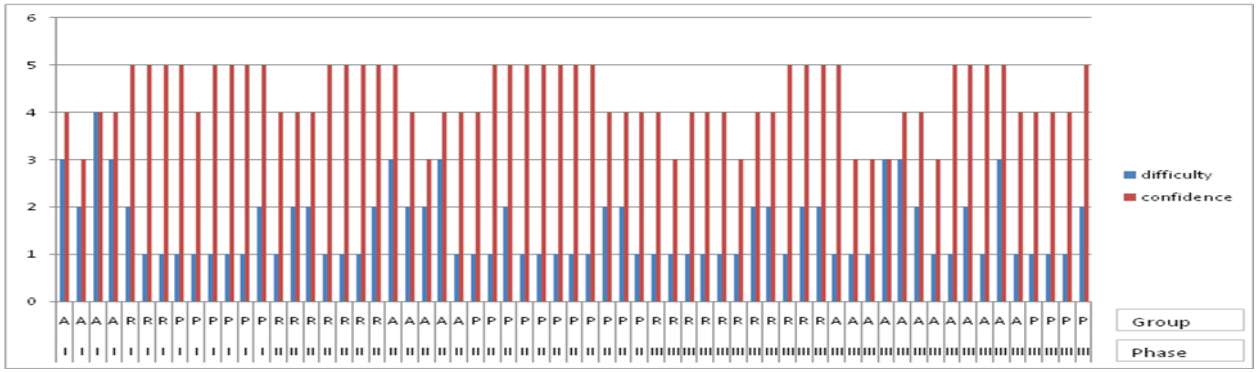

Figure 3.3 Results for Survey Question Part II (b)

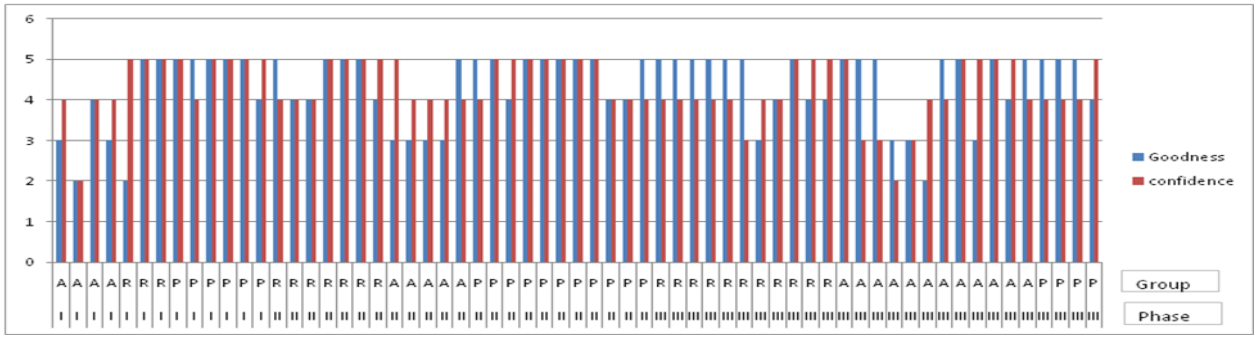

Figure 3.4 Results for Survey Question Part II (c)

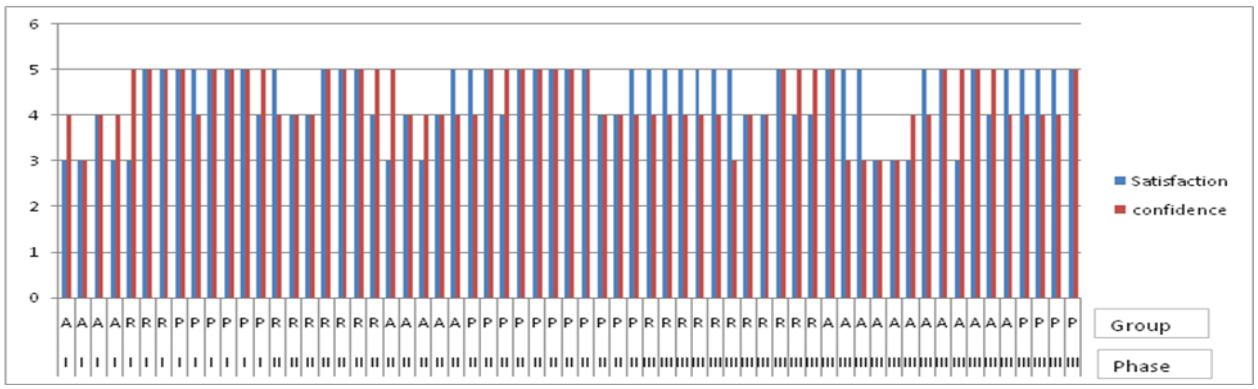

Figure 3.5 Results for Survey Question Part II (d)

The survey result established the fact that the framework covers comprehensive information/knowledge and understanding of software component terminologies. The people were satisfied in software components classification according to dimensions and classes. The framework also gave an insight into the component characteristics belonging to the same class. The *SDCS* framework survey result showed increased level of exposure in understanding new technologies. One of the prominent applications of the *SDCS* framework has been to improve *new technological learning*. This surely establishes potential of *SDCS* framework for the academia, software development and research industry to perceive software components according to their project goals and requirements. In the global market the *SDCS* framework provides the individual (user, group or organization) an edge over their competitors. Moreover so far no study has been conducted that can deal with software component classification in such a comprehensive manner.

3.6 Concluding Remarks

In this chapter, the SDCS framework is developed that can classify software components on the basis of six dimensions - *architecture level, domain, kind, source, generic functionality* and *phase*. These dimensions have been chosen after critical review. Comparison and evaluation of software components can be performed on a homogeneous set of software components such as SSL, SSE, CSL, CSE etc. The framework leads to a broader and an in depth classification.

In the next chapter, *usability* aspect of software component is dealt. The presence of this characteristic in component quality model shows a significant difference as compared to its presence in conventional quality models. The chapter begins with the identification of *usability* sub-characteristics and respective attributes. Later, evaluation and design of a component as per *usability* point of view is discussed. The chapter also presents usefulness of the developed methodology by exploring case study.