A Review on Present State-of-the-Art of Self Adaptive Dynamic Software Architecture

Sridhar Gummalla,
Shadan College of Engineering & Technology

G. Venkateswara Rao,
Gitam Institute of Technology,
GITAM University.

Abstract— Enterprises across the world are increasingly depending on software to drive their businesses. It is more so with distributing computing technologies in place that pave way for realization of seamless business integration. On the other hand those complex software systems are expected to adapt changes dynamically without causing administrative overhead. Moreover software systems should exhibit fault tolerance, location transparency, availability, scalability self-adaptive capabilities to fit into present enterprise business use cases. To cope with such expectations software systems are to be built with a dynamic and self-adaptive software architecture which drives home quality of services perfectly. The point made here is that software systems are facing unprecedented level of complexity and aware of self-adaptation. Therefore it is essential to have technical knowhow pertaining to self adaptive dynamic software architecture. Towards this end, we explore present state-of-the-art of this area in software engineering domain. It throws light into dynamic software architectures, distributed component technologies for realizing such architectures, besides dynamic software composition and metrics to evaluate the quality of dynamic adaptation.

***Keywords** – Software engineering, dynamic software architecture, self-adaptation, metrics*

*****

I. INTRODUCTION

Software development process has undergone tremendous changes since its inception. These changes are in tune with the ever increasing needs of clients and machine critical applications. Software architectures utilize rich set of abstractions and idioms. These abstractions and idioms can represent different scenarios of the system besides the nature of interactions among the components [6]. Allen *et al*. [7] opined that the most challenging factor of complex software architecture is the need for dynamic adaptation at runtime to the changing needs. They also conceived the possibilities of building such architectures that are robust to changes. According to Shaw and Clements [9] in 1980s software architecture emerged prominent in software engineering discipline. Later on the self-adaptive and dynamic software architectures were conceived. Towards self adaptive dynamic software architecture distributed computing technologies contributed in the recent past. Distributed Component Object Model (DCOM) and Windows Communication Foundation (WCF) are technologies from Microsoft for realizing dynamic software composition. Common Object Request Broker Architecture (CORBA) is from Object Management Group (OMG) which is another such technology. In Java platform distributed technologies include Enterprise Java Beans (EJB), Java Messaging Service (JMS), Remote Method Invocation (RMI) and Web Services. These technologies are being used to realize dynamic software architectures that can self-adapt to the runtime needs of complex software systems.

TABLE 1 – Acronyms/Abbreviations

ACRONYM/ABBREVIATION	DESCRIPTION
(RMI)	Remote Method Invocation
(JMS)	Java Messaging Service
(EJB)	Enterprise Java Beans
(OMG)	Object Management Group
(CORBA)	CommonObject Request Broker Architecture
(WCF)	Windows Communication Foundation
(DCOM)	Distributed Component Object Model
pIA	Performance Influence on
pQoR	Performance Quality of Response
pLatency	Performance Latency
UiAI	User interaction adaptivity index
AiAI	Administrator Interaction Adaptivity Index
SDG	Storage Dimension Growth
IALFL	Influence of the Adaptive Logic on the Functional Logic
IFLAL	Influence of the Functional Logic on the Adaptive Logic
MaAC	Minimum Architectural Adaptive Cost
aSCI	Architectural Separation of Concerns Index
UML	
ADLs	Architecture Description Languages
DSSA	A Domain Specific Software

	Architecture

Since changes to software frequently are a costly affair, there should be an architecture that can dynamically adapt to changes. Therefore architectural design needs to be made keeping this need in mind. Moreover an architectural design should have structure, design constraints, style, behaviour, and refinement [22]. Towards realizing this some architectural design decisions are to be made. Jansen & Bosch [10] proposed a model for design decisions at architecture level. The architectural design decisions involve motivation, problem, decision, trade-off, cause, and architectural modification that lead to satisfactory solutions. A Domain Specific Software Architecture (DSSA) was proposed by Hayes-Roth *et al.* [11] for adaptive intelligent systems. The reference model they proposed contains architecture styles pertaining to blackboard, pipe and filter. Software architecture styles were also represented using graph grammars as explored in [12]. Taylor *et al.* [15] proposed an architecture that involves both component and message-based style of functionality. Distributed computing architectures were defined for wearable devices as well [18]. Architectural compatibility also required dynamic compilation as discussed in [19]. Some researchers focused on dynamic software architectures in the presence of cloud computing [16], [20], [23], [28]. As software architecture is linked to business goals in the recent past there is ever growing need for self-adaptive dynamic software architecture [44].

II. DYNAMIC STRUCTURE IN SOFTWARE ARCHITECTURES

Software architectures are defined by using Architecture Description Languages (ADLs) [3]. Architecture of a system refers to a set of organized components, their relationships, design principles and other aspects that govern the design and implementation of the system. Dynamic software architecture, as the name implies, is the architecture that will undergo changes while the system is under execution. ADL is used to describe the architecture of a system. One such language is Darwin which involves components and services, instantiation and binding, configurations, lazy instantiation, direct dynamic instantiation, and dynamic binding [2]. Muccini *et al.* [1] used software architecture specification for testing the implementation.

III. RECONFIGURATION OF SOFTWARE ARCHITECTURE

The ability to reconfigure software architecture is essential to adapt to changing requirements. Towards this end standards based component development is a good practice. Wermelinger and Fiadeiro [4] proposed an approach to achieve this using three characteristics. First, they defined components using a high-level language with notion of the state. Second, focus on ensuring correct state when new components are introduced. Third, explicit management of reconfigurations if any. They employed graph transformation approach for reconfiguration of software architecture. Algebraic construction of equivalent program is the technique used to by providing semantics to given architecture. Advances in integrated circuit technologies also led to the faster growth of software architectures [35]. In the context of reconfigurable software architectures, visualization techniques are also used to present complex architectures. Visualization concepts are explored in [34],

Architecture level dependency analysis was explored in [33] in the context of emerging formal software architectural models. When system architecture details and recovery details are not documented, it is essential to have abilities to recovery system architecture. Towards software architecture recovery Maqbool & Babri [32] proposed an approach based on hierarchical clustering. In this approach the quality of the system depended on the similarity measures, algorithm and peculiarities of software system. Similar kind of work for software architecture recovery was made in [30] using a lightweight workbench that can show the information on the architectural dynamics. State charts can be used to analyze software architectures [31]. In the process of self-adaptive and dynamic software architecture it is essential to characterize software architecture changes. In [29] very useful review can be found for the same.

UML profiles are also explored to describe software architectures as UML provides key abstractions [27]. Software architectures are also used for code testing as explored in [26]. Architecture can also be used as a basis for reliability testing [25]. This is because the component based systems depend on the architecture that supports the realization of loosely coupled and highly cohesive software components. Another interesting fact is that software architecture can be used to exploit the underlying features towards software repair. This capability is already built into operating systems and other system software [24].

IV. ARCHITECTURE – BASED SELF ADAPTATION

Architecture based self adaptation helps in flexible reuse of software infrastructure in order to cope with runtime demands of a system under execution. Cheng *et al.* [5] proposed a framework known as “Rainbow” which was designed to be capable of monitoring system at runtime and adapt to its dynamic needs. It employs externalized control mechanisms at global level that separate system functionalities from the external behaviours. The framework provides a low-cost approach for self-adaptive dynamic software architecture that caters to the needs of runtime situations of software system. The separation of concerns made this architecture more manageable and maintainable. The adaptation infrastructure is made reusable while the knowhow on adaptation is not reusable. The infrastructure is built at two layers. They are known as system-layer infrastructure and architecture – layer infrastructure. The former deals with system-level functionalities including measures to know the state of the system. Whereas the latter is to deal with aggregating probe information collected from system layer and takes care of adaptation issues [5].

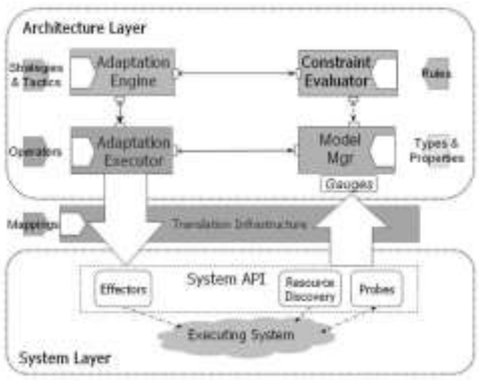

Figure 1 – Overview of Rainbow framework [5]

As can be seen in Figure 1, the framework has two layers namely system layer and architecture layer. Between the two layers there is translation infrastructure that acts as glue between them. The system layer represents an execution system that makes use of different kinds of standard API that is implemented and realized. The architecture layer has components like adaptation engine, adaptation executor, constraint evaluator, and model manager. These components are self explanatory and they perform intended tasks [5]. Rainbow was improved further in [38] with notional customization points. The new architectural style accommodates vocabulary, design rules, properties and analysis. Gomma & Hussein [13] proposed an architecture that involves software reconfiguration patterns for product lines with dynamic software architecture. The overview of their framework is shown in figure 2.

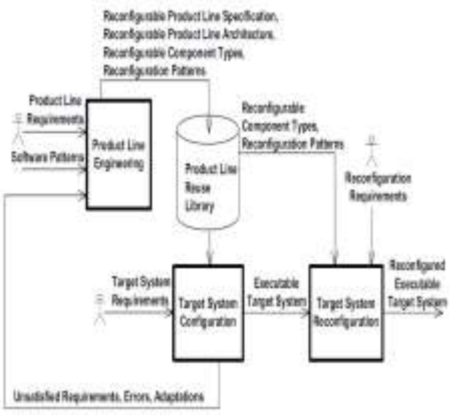
Reconfigurable Product Line Specification,
Reconfigurable Product Line Architecture,
Reconfigurable Component Types,
Reconfiguration Patterns
Product Line Requirements
Software Patterns
Product Line Engineering
Product Line Reuse Library
Reconfigurable Component Types, Reconfiguration Patterns
Reconfiguration Requirements
Target System Requirements
Target System Configuration
Executable Target System
Target System Reconfiguration
Reconfigured Executable Target System
Unsatisfied Requirements, Errors, Adaptations

Figure 2 – Reconfigurable product line life cycle that evolves over a period of time [13]

The framework has important components like product line engineering, target system configuration, and target system reconfiguration. The product line engineering involves reconfigurable product line specification, product line architecture and component types and patterns. There is reusable product line library hat is used to realize target system configuration based on the target system requirements. The feedback is taken from the users of target system and given to product line engineering again. When reconfiguration is required, the target system reconfiguration component takes executable target system, reconfigurable component types and reconfiguration patterns based on the reconfiguration requirements in order to realize the final system with dynamic changes. This is a cyclic process that can help in evolution of new products in the given product line [13]. Therefore it is understood that the product line also needs self-adaptive dynamic architecture. Cheng *et al*. [17] proposed dynamic adaptation architecture for pervasive systems. Since pervasive computing needs self-adaptive software architecture, they explored it and proposed their own architecture as presented in Figure 3.

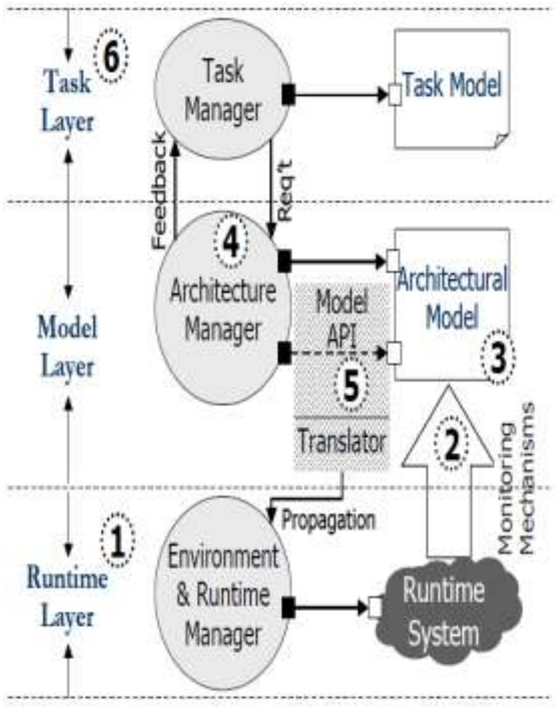

Figure 3 – Self-adaptation framework [17]

There are three layers in the framework namely runtime layer, model layer and task layer. These three layers work together in order to realize self-adaptation. The runtime layer encapsulates the runtime system, environment and runtime manager. The monitoring mechanisms employed in this layer provide the details to the model layer. The model layer contains model API translator that that takes care of runtime changes and propagates them back to environment and runtime manager.

The architectural model is managed by architectural manager. The model layer interprets runtime behaviour of the system including constraints and has provision for adapting new changes. The quality of service requirements are determined by the task manager. The high-level representation of specific computational need of user is known as a task. The low-level operations to adapt to changing requirements are taken care of by the runtime layer. However, model layer is the actual layer that takes care of high-level changes to be made to the system on the fly [17]. Thus the architecture based adaptation is realized. Self- healing is one of the characteristics of dynamic software architecture with self-adaptive feature [21]. A dynamic architecture was proposed for artificial neural networks which proved to be highly efficient in terms of pattern recognition, prediction, clustering and classification [43]. Irrespective of the frameworks that came into existence, there are certain common principles that were leveraged in all solutions. Those principles include high cohesion, low coupling, standard interfaces, and concise semantics [42]. Architecture centric software development can help software engineers to align their activities with the underlying design principles [41].

Identifying or predicting maintenance issues and addressing them at architecture level can lead to dynamic software architectures [40]. Self-adaptive dynamic system architecture should have provision for commercial off-the-shelf components and also synthetic components in order to adapt to changes at runtime dynamically [39]. GiPSi [37], a framework for open source architecture, was proposed for shared development of software models with heterogeneity.

In [36] metrics were proposed in order to evaluate the dynamic adaptability feature of self-adaptive dynamic architectures. The metrics are categorized into architectural metrics, structural metrics, interaction metrics, and performance metrics.

TABLE 2 - Shows the summary of the metrics

Metric Category	**Metric**	**Equation**	**Description**
Architectural Metrics	aSCI (Architectural Separation of Concerns Index)	aSCI = Number of adaptive elements/Number of functional	Indicates the difference between adaptive elements and functional logic.
	MaAC (Minimum Architectural Adaptive Cost)	MaAC = Minimum number of elements for adaptivity	It is related to the growth of architecture in terms of number of elements for adaptivity.
Structural Metrics	IFLAL (Influence of the Functional Logic on the Adaptive Logic)	IFLAL = Number of inputs in the adaptive logic	Indicates the role of adaptive logic in terms of inputs
	IALFL (Influence of the Adaptive Logic on the Functional Logic)	IALFL = Number of outputs from the adaptive logic	Indicates the role of adaptive logic in terms of outputs
	SDG (Storage Dimension Growth)	SDG = $KB_{withAdaptivity} - KB_{wthoutAdaptivity}$	Expresses structural growth in size
Interaction Metrics	AiAI (Administrator Interaction Adaptivity Index)	AiAI = $\sum \{ \begin{matrix} 1\ if\ an \\ 0 \end{matrix}$	Analyzes actions performed by administrator
	UiAI (User interaction adaptivity index)	UiAI = $\sum \{ \begin{matrix} 1\ if\ an\ action \\ 0\ otherw \end{matrix}$	Analyzes actions performed by user
Performance Metrics	pLatency (Performance Latency)	pLatency = Response time in presence of adaptivity / Response time in absence of adaptivity * 100	Indicates delay in system performance in the presence of adaptivity
	pQoR (Performance Quality of Response)	pQoR = Quality of response in the presence of adaptivity/quality of response in the absence of adaptivity * 100	Indicates the quality dynamics in the presence of adaptivity
	pIA (Performance Influence on Adaptivity)	$pIA = \{ 0\ if$	Influence of performance metrics on adaptive strategies.

V. DYNAMIC SOFTWARE COMPOSITION WITH SHARING

Distributed computing technologies paved way for reusing software components that can be leverage a great design principle "do not reinvent the wheel". Software industry adapted the gist of hardware systems where components are reused. For instance, in a computer system all components are not made by same company. Though they were built by different manufacturers, the system is working perfectly. The reason behind this is that component technology uses standard and global interfaces that are common across the world. This kind of idea the software industry borrowed from the hardware industry. The realization of this idea is made possible with the invent of distributed computing technologies. As discussed in [8], distributed component technologies provide a way to reuse software components running anywhere in the world. Another key fact here is that the component technology supports reuse of components irrespective of the platform in which they are built thus promoting interoperability. The technology is rich in this area of software architecture development with dynamic software composition with sharing resources. Distributed Component Object Model (DCOM) and Windows Communication Foundation (WCF) are technologies from Microsoft for realizing dynamic software composition. Common Object Request Broker Architecture (CORBA) is from Object Management Group (OMG) which is another such technology. In Java platform distributed technologies include Enterprise Java Beans (EJB), Java Messaging Service (JMS), Remote Method Invocation (RMI) and Web Services. With these rich set of technologies it is possible to realize self-adaptive dynamic software architecture.

The component model allows the components to be reused recursively and with nesting. The component technology leverages the reuse of business components instead of reinventing the wheel. This design principle is possible with above described distributed computing technologies. The grid computing and cloud computing (newly emerged technology that allows shared pool of resources to be used in pay per use fashion) are also best examples in which realization of software reuse is abundant. The component model as explored in [8] leverages encapsulation and identity, composition, sharing, life-cycle, activities, control, and mobility. The fractal component model proposed in [8] is based on Java components, interfaces and names.

Dynamic adaptive optimization is another feature required by dynamic software architectures. As Bruening [14] states clearly dynamic optimization is an emerging field in software engineering that can be coupled with dynamic adaptive software architectures. They provide interfaces for different activities such as dynamic translation, profiling, instrumentation and optimization. Then they realized the same with 40% improvement in performance and 12% in average speedup.

VI. CONCLUSIONS AND FUTURE WORK

Software development industry has undergone tremendous changes. From the trial and error basis development to more systematic approach towards software development has been witnessed in the last few decades. In the process an important aspect is the software architecture and its implications on the life of software and the real world. Software architecture played pivotal role in achieving goals of organizations. Dynamic software architecture that can adapt to changing needs at runtime is the main focus of this paper. Self-adaptive dynamic software architecture is an important research topic now as there are rich set of distributed computing technologies that can be used to realize it. In this context, we attempted to review the present state-of-the-art of self-adaptive dynamic software architecture. We observed that the self – adaptive dynamic software architecture can be realized considering the insights presented in this paper. This paper also explores the metrics available to evaluate dynamic software architectures. This research can be extended further to investigate more fine grained aspects of such architectures for comprehensive and reliable adaptation to changes in the execution time.

VII. REFERENCES

[1] Muccini,Henry , Bertolino, Antonia,and Inverardi,Paola . (2004). Using Software Architecture for Code Testing. *IEEE*. 30 (3), p.160-170.

[2] Magee,Jeff and Kramer,Jeff . (1996). Dynamic Structure in Software Architectures. *Computer Security Division Information Technology Laboratory*.p.80-90.

[3] Medvidovic,Nenad . (1996). ADLs and Dynamic Architecture Changes.Department *of Information and Computer Science*.p.45-56.

[4] Michel Wermelingera;, Jos'e Luiz Fiadeirob. (2002). Agraph transformation approach to software architecture recon"guration.*Elsevier*. 44 (1), p.133 – 155.

[5] Shang-Wen ,Cheng An-Cheng ,Huang David, Garlan, Bradley Schmerl ,Peter Steenkiste. (2004). Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure. *IEEE*. p.560-600.

[6] Mary ,Shaw. (1995). Abstractions for Software Architecture and Tools to Support Them. *IEEE*. 21 (4), p.45-56.

[7] Robert Allen, Rdmi ,Douence, and David, Garlan. (1998).Specifying and Analyzing Dynamic Software Architectures. *School of Computer Science, Carnegie Mellon University, Pittsburgh*. p.780-800.

[8] E. Brunetony, T. Coupayey, and J. B. Stefaniz. (1997).Recursive and Dynamic Software Composition with. *Computational Sciences and Engineering Division*.p.65-70.

[9] Mary, Shaw. (2006). The Golden Age of Software Architecture. *IEEE*. p.54-65.

[10] Anton ,Jansen. (2005). Software Architecture as a Set of Architectural Design Decisions. *Department of Computing Science*. p.90-100.

[11] Barbara Hayes-Roth, Karl Pfleger, Philippe Lalanda, Philippe Morignot, and Marko Balabanovic. (1995). A Domain- Specific Software Architecture for Adaptive Intelligent Systems. *IEEE*. 21 (4), p.30-45.

[12] Daniel Le, Métayer. (1998). Describing Software Architecture Styles Using Graph Grammars. *IEEE*. 24 (7), p.34-55.

[13] Hassan, Gomaa and Mohamed, Hussein. (2004). Software Reconfiguration Patterns for Dynamic Evolution of Software Architectures. *IEEE*. p.234-305.

[14] Derek Bruening, Timothy Garnett, and Saman Amarasinghe. (2003). An Infrastructure for Adaptive Dynamic Optimization. *IEEE*. p.90-103.

[15] Richard N. Taylor, Nenad Medvidovic, Kenneth M. Anderson. (1996). A COMPONENT- AND MESSAGE-BASED ARCHITECTURAL STYLE FOR GUI SOFTWARE. *IEEE*. 22 (6), p.60-70.

[16] Trieu C. Chieu, Ajay Mohindra, Alexei A. Karve and Alla Segal. (2009). Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. *IEEE*.p.234-305.

[17] Shang-Wen Cheng, David Garlan, Bradley Schmerl, João Pedro Sousa. (2002). Software Architecture-Based Adaptation for Pervasive Systems. *Springer-Verlag Berlin Heidelberg*. p.45-56.

[18] Juha Lehikoinen, Jussi Holopainen, Marja Salmimaa, Angelo Aldrovandi. (1999). MEX: A Distributed Software Architecture for Wearable Computers. *Department of Computer Science*.p1-16.

[19] Kemal Ebciofjlu and Erik R. Altman. (1997). DAISY: Dynamic Compilation for 100% Architectural Compatibility. *ACM*. p.234-305.

[20] Namjoshi, J,. and Gupte, A. (2009). Service Oriented Architecture for Cloud based Travel Reservation SoftwarService. IEEE. P1-4.

[21] Qun,Y,. and Xian-chun,. and Y,. Man-wu,. (2005). A Framework for Dynamic Software Architecture-based Self-healing. P1-4.

[22] Namjoshi, J,. and Gupt, A. (2009). Service Oriented Architecture for Cloud based Travel Reservation Software as a Service. IEEE. P1-4.

[23] Liang-Jie ,Zhang and Qun ,Zhou. (2009). CCOA: Cloud Computing Open Architecture. *IEEE*.p.45-56.

[24] Schmerl, B and Garlan, D. (2002). Exploiting Architectural Design Knowledge to Support Self-Repairing Systems. *ACM*. p1-8.

[25] Popstojanova, G.K,. and Trivedi, S.K. (2001). Architecture-based approach to reliability assessment of software systems. *Elsevier Science*. p1-26.

[26] Muccini, H,. Bertolino, A,. and Inverardi, P. (2004). Using Software Architecture for Code Testing. *IEEE*. 30 (3), p1-12.

[27] Kandé, M.M,. and Strohmeier, A. (2000). Towards a UML Profile for Software Architecture Descriptions. *Springer*. p1-15.

[28] Namjoshi, J and Gupte, A. (2009). Service Oriented Architecture for Cloud based Travel Reservation Software as a Service. *IEEE*. p1-4.

[29] Yeh, S.A,. Harris, R.D and Chase, M.P. (1997). Manipulating Recovered Software Architecture Views*. *ACM*. p1-12.

[30] Williams, J.B,. and Carver, C.J. (2009). Characterizing software architecture changes: A systematic review. *Elsevier*. p1-31.

[31] Dias, S.M and Vieira, R.E.M. (2000). Software Architecture Analysis based on Statechart Semantics. *IEEE*. p1-133.

[32] Maqbool, O and Babri, A.H. (2007). Hierarchical Clustering for Software Architecture Recovery. *IEEE*. 33 (11), p1-22.

[33] Stafford, A. J and Wolf, L.A. (2001). ARCHITECTURE-LEVEL DEPENDENCE ANALYSIS FOR SOFTWARE SYSTEMS. *World Scientic Publishing Company*. 11 (4), p1-21.

[34] Jerding, D and Rugaber, S. (197). Using Visualization for Architectural Localization and Extraction. *IEEE*. p1-5.

[35] Kozyrakis, E.C and Patterson, A.D . (1998). A New Direction for Computer Architecture Research. *IEEE*. p1-9.

[36] Raibulet, C and Masciadri, L. (2009). Evaluation of Dynamic Adaptivity Through Metrics: an Achievable Target?. *IEEE*. p1-4.

[37] C¸ avus¸o˘glu, C.M. (2009). GiPSi: A Framework for Open Source/Open Architecture Software Development for Organ-Level Surgical Simulation. *IEEE*. 10 (2), p1-11.

[38] Garlan, D,. Schmerl, B and Cheng, W.S. (2009). Software Architecture-Based Self-Adaptation. *Springer*. p1-29.

[39] Ravindran, B. (2002). Engineering Dynamic Real-time Distributed Systems: Architecture, System Description Language, and Middleware.*IEEE*. 28 (1), p1-28.

[40] Bengtsson, P and Bosch, J. (1999). Architecture Level Prediction of Software Maintenance. *IDE*. p1-9.

[41] Keller, F and Wendt, S. (2003). FMC: An Approach Towards Architecture-Centric System Development. *IEEE*. p1-10.

[42] Battin, D,. Crocker, R and Kreidler, J. (2001). Leveraging Resources in Global Software Development. *IEEE*. p1-8.

[43] Ghiassi, M and Saidane, H. (2004). A dynamic architecture for artificial neuralnetworks. *Elsevier*. p1-21.

[44] Jens ,Knodel. (2007). A Comparison of Static Architecture Compliance Checking Approaches. *IEEE*. n.d (n.d), p.45-56.