IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 31, 2019, accepted September 14, 2019, date of publication September 26, 2019, date of current version October 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2944119

Design Quality Metrics to Determine the Suitability and Cost-Effect of Self-* Capabilities for Autonomic Computing Systems

ABDUL JALEEL, SHAZIA ARSHAD, MUHAMMAD SHOAIB, AND MUHAMMAD AWAIS

Department of Computer Science and Engineering, University of Engineering and Technology, Lahore, Lahore 54890, Pakistan

Corresponding author: Abdul Jaleel (abduljaleel@uet.edu.pk)

ABSTRACT Every software in the universe requires maintenance and management during its life cycle. The manual management of software is costly and sometimes error-prone. The other solution is autonomic computing that induces self-management capabilities, ''self-*services'', in software systems with the help of autonomic managers. The design quality of a self-management capability affects the computing infrastructure regarding processing load, the memory requirement, data channel demand and performance of perturbation restore. It is critical to assess the design quality of a self-management capability to determine its effect over the computing infrastructure when it gets invoke against some anomaly or perturbation. Moreover, there are two possible host environments for an autonomic manager to offer a self-management capability as a self-* service: the local environment and the cloud environment. A criterion is needed to decide which environment is more suitable and cost-effective to run the service. However, the literature lacks in the assessment of the design quality metrics on self-management capabilities and the suitability and cost-effectiveness of the execution environment. In this work, we have proposed a suite of design quality metrics to determine the design quality of self-management capabilities. We validate the proposed metrics with a stock trade & forecasting system that was designed as an autonomic computing system with self-management capabilities. The proposed metrics were applied to define functions that identify the suitable and cost-effective execution environment for the self-* service. The results proved that these metrics are useful in determining the design quality, suitability, and cost-effectiveness of a self-* capability for an autonomic computing system. The proposed metrics can be used to compare differently designed autonomic solutions for complexity, efficiency, performance, understandability, and maintainability.

INDEX TERMS Autonomic computing, design quality metrics, self-management capabilities, self-* service, stock trade forecasting.

I. INTRODUCTION

During the life cycle of software, a significant amount of human effort is required to control and manage the software. In the case of complex IT systems, an increased number of skilled IT professionals are required for configurations, installations, maintenance, and operation of these systems [1]. Autonomic software management facilitates in minimizing the IT budget by reducing human efforts required to install, maintain and operate an IT system. By shifting the human task of software controlling to just policy and rules defining, autonomic computing automates the manual task of management and introduces self-management capabilities like self-configuration, self-healing, self-optimization, self-protection, and others. These are collectively termed as self-* capabilities [2].

The motivation behind this work is that the design quality of a self-management capability (SMC) can be used to indicate how an SMC will affect the computing infrastructure in terms of processing load, the memory requirement, data

The associate editor coordinating the review of this manuscript and approving it for publication was Malik Jahan Khan.

channel demand and performance of perturbation restore. Therefore, it is critical to assess the design quality of a self-management capability to determine its effect over the computing infrastructure when it invokes against some perturbation. Moreover, there are two possible host environments for an autonomic manager to offer a self-management capability as a self* service: the local environment and the cloud environment. Thus, a criterion or metric is needed to decide which environment is more suitable and cost-effective to run the service.

However, most of the previously defined metrics [2]–[4] of autonomic computing systems are not applicable to design phase output except the work of kaddoum *et al.* [2]. The primary drawback of the kaddoum's work is that it does not measure the complexity and cost of autonomic logic embedded inside the computing system. Also, the literature lacks in studies that select a cost-effective execution environment for a self-management capability when the option of both local and cloud environment is available. Therefore, it is necessary to define metrics that measure the design quality of self-* capabilities and thus of the autonomic computing systems. Also, these metrics should be useful to identify whether the local environment or cloud environment is better to run the SMC service.

The main goal of this research is to define a suite of design quality metrics to calculate the perturbation complexity and performance of SMC at design time, so that two autonomic computing systems can be compared and the execution environment of an autonomic computing system can be identified. More specifically, we have asked the following questions.

(1) How an autonomic computing system can be formally represented in terms of self-management capabilities?

(2) What quantitative measures can be defined for design quality assessment of a self-management capability considering different design quality aspects?

(3) How to determine a cost-effective and suitable execution environment for the self-* service?

(4) How to compare differently designed SMC solutions for complexity, efficiency, performance, understandability and maintainability?

While addressing these research questions, we used IBM's autonomic computing model as reference architecture and draw a schema interaction graph for autonomic systems in terms of self-management capabilities (SMCs). The graph adds formality in the architectural model of the IBM which was required to develop metrics based on the formal definitions. Then, we proposed a suite of metrics to evaluate the design quality of self-management capabilities. These metrics quantify the complexity, recovery time, bandwidth cost, memory requirements, and data traffic load incurred by an SMC. By using the defined metrics, we devised a method to select the most appropriate execution environment (local, cloud) for the application. A 'Stock Trading and Forecasting System' was designed as a self-managing application with high-availability to validate the proposed metrics and methods. The proposed method successfully determines the most appropriate execution environment (local or cloud) of the application based on four characteristics: data channel requirement; network load for perturbation resolving; incurred cost for memory; and time for perturbation resolving. Finally, we map the metrics to ISO-9126 design quality parameters to compare different design solutions for complexity, efficiency, performance, understandability, and maintainability. We conclude that the proposed metrics are useful for quantitative assessment of the design quality of an autonomic computing system in terms of its self-management capabilities. A measure of complexity, performance, and efficiency of a self-management capability at design time facilitates judging its behavior before actual implementation and execution.

The paper is divided into seven sections including the introduction section. In section II, we review the previous work related to autonomic computing and related metrics. Section III formally describe autonomic computing and self-management capabilities. Section IV is entitled to the proposed design quality metrics for the assessment of a self-management capability. Evaluation of the proposed metrics over the design of a highly available self-managing application is given in section V. Section VI is about discussions. Finally, the research is concluded in section VII.

II. RELATED WORK

The idea of autonomic computing is to make computing systems self-managing, behaving and controlling themselves just like the human nervous system to handle the increasing complexity of IT systems [1], [5]. IBM's blueprint for autonomic computing systems [6] is recognized as the most appropriate model for autonomic computing. An autonomic computing system has two kinds of behaviors: the functional behavior representing actual systems, and the self-* behavior enabling the system to handle unexpected dynamics and perturbations [7]. A perturbation in autonomic computing is ''a deviation of the computing system or process from its regular or normal state or path, caused by an outside influence'' [8]. The anomaly behavior of the computing resource or its environment is characterize-able by a set of context attributes [9]. A resource, to be managed, has to shares its running state in the form of context attributes via its touchpoints. Autonomic managers are the software modules that implement the self-* capabilities via feedback control loop with the help of sensors and effectors. Sensors and effectors inside the resource touchpoint act as 'Set' and 'Get' methods for context attributes [10], [11]. A sensor senses anomalies and informs the autonomic manager(s). An effector receives commands and data from the autonomic manager(s) and triggers an action to manage the system state.

A broad literature analysis was performed in [12] to apply autonomic computing concepts over cloud resource provisioning and management with QoS considerations. Reference [13] uses the concept of autonomic computing for QoS-aware cloud resource provisioning and scheduling

which improves the resource utilization and user satisfaction.

McCann *et al.*, 2004, describe a set of metrics to compare and evaluate the performance of autonomic systems in terms of quality of service, cost, failure avoidance, time to adapt, reaction time, sensitivity, and stabilization [4]. The software quality model (ISO 9126-1) identifies the six software quality characteristics/factors as functionality, reliability, usability, efficiency, maintainability, and portability [3], [14], [15]. Tayagi *et al.*, 2013, allocated different software quality factors as metrics to major characteristics of an autonomic computing system [16]. Shazia *et al.*, 2011, defined design quality metrics for web-based applications and shows the importance of measurement at design time [17]. Omid mola, 2011, has given formal specifications for the managed object, autonomic manager, policies, and communication model of an autonomic system [18]. Raibulet, *et al.*, 2014, grouped adaptivity domain metrics into architectural, structural, interaction, and performance categories [19]. Xavia etcheves, 2010, defined qualitative and quantitative metrics for autonomic systems based on the ISO 9126-I standards [20]. Sukhpal singh and inderveer chana, 2014, identified different cloud workloads and defined a set of metrics to allocate appropriate workload to appropriate cloud resources in IAAS for QoS based cloud resource management [21].

Kaddoum *et al.*, 2009, describe the evaluation criteria for self-* systems without actually running the system which is based on the intrinsic characteristics like communication complexity (number of exchanged messages), decentralization, the requirement of local algorithms, and influence of the number of agents [2]. Kaddoum *et al.*, 2010, defined metrics for six different measurements criteria of autonomic features: i.e., for methodological, architectural, systems growth, intrinsic characteristics, influences of the adaptive logic, and for automation of the human tasks [22]. Paul Lin *et al.*, 2005, divides quality metrics for autonomic computing framework into anticipatory and openness [23]. Anticipatory characteristic is further subdivided into context-awareness, self-awareness, and self-management which is further subdivided into self-healing, self-configuration, self-protection, and self-optimization. Authors have just inlined the quality metrics, and their quantifications are not given.

After a comprehensive literature survey, it is identified that most of the existing metrics are qualitative covering functionality, reliability, usability, efficiency, maintainability, and portability characteristics [1], [24]–[26] but the researches have not described how to measure them. Moreover, the metrics already defined [1], [4], [16], [19] for autonomic computing systems are not applicable to design output, except [2] which also does not measure complexity and cost of autonomic logic embedded inside an autonomic computing system. It is identified that few works have defined quantitative metrics and very few of them evaluated their proposed metrics. Also, no metric has been defined to measure complexity, cost, and strength of autonomic logic from the design of an autonomic computing system.

III. AUTONOMIC COMPUTING AND SELF-MANAGEMENT CAPABILITIES

This work is based on IBM's concept of autonomic computing. In IBM's layer model [6], [27] of the autonomic computing systems, a managed resource (MR) is at the bottom layer. MR implements a touchpoint (TP) as its manageability interface [10]. Autonomic managers (AM) above managed resources act as controlling agents using touchpoint interface of the resource. Orchestrating autonomic managers (OAMs) in the next layer facilitates interaction among autonomic managers. An AM can control one or more MRs. Similarly, an MR may be controlled by one or more AMs [9]. A manual manager (MM) at the top layer provides an interface to users/experts to input data and the commands when OAMs or AMs have no solution for a situation. Managers implement the MAPE-K (monitor, analyze, plan, execute, and knowledgebase) feedback control loop, except MM where actions are asked from users/experts. In this bottom-to-top approach, a perturbation is handled by an AM with the help of OAMs, and MM. Thus an autonomic computing system consists of a set of manager modules (M) and resource modules (R). The interaction between modules is a connection over which data/message transmits. The required sequence of interactions (between modules of the system) against a specific anomaly can be determined at design time. We used IBM's model as reference architecture and draw a schema graph for autonomic systems in terms of self-management capabilities.

A. SCHEMA REPRESENTATION OF AN AUTONOMIC COMPUTING SYSTEM IN TERMS OF SELF-MANAGEMENT CAPABILITIES

A self-management capability corresponds to an autonomic behavior of the computing system. We define an SMC as a collection of interactions between modules of an autonomic system, with each interaction initiating from a manager or a resource inside the autonomic computing system (ACS). These interactions activate sequentially to counter an abnormal activity. A schema interaction graph of an example autonomic computing system is depicted as a directed multi-graph in Fig. 1. The required sequence of interactions (between modules of the system) against a specific anomaly can be determined at design time using this graph. This is required for design quality assessment of SMCs and in turn, of the ACS.

The managers and resources of an ACS are taken as nodes of the schema interaction graph. A directed edge represents either a sensor-monitor interaction or an executor-effector interaction between manager-resource or manager-manager modules. Let k and k' be some integer values. The weight of k^{th} interaction initiating from a manager node M_i is represented as $\alpha^k_{M_i}$ and the weight of k'^{th} interaction initiating from a resource node R_j is represented as $\alpha^{k'}_{R_j}$. Both are quantified as the size of data (in bytes) transferred over one-time activation of the interaction. Fig. 1 depicts three

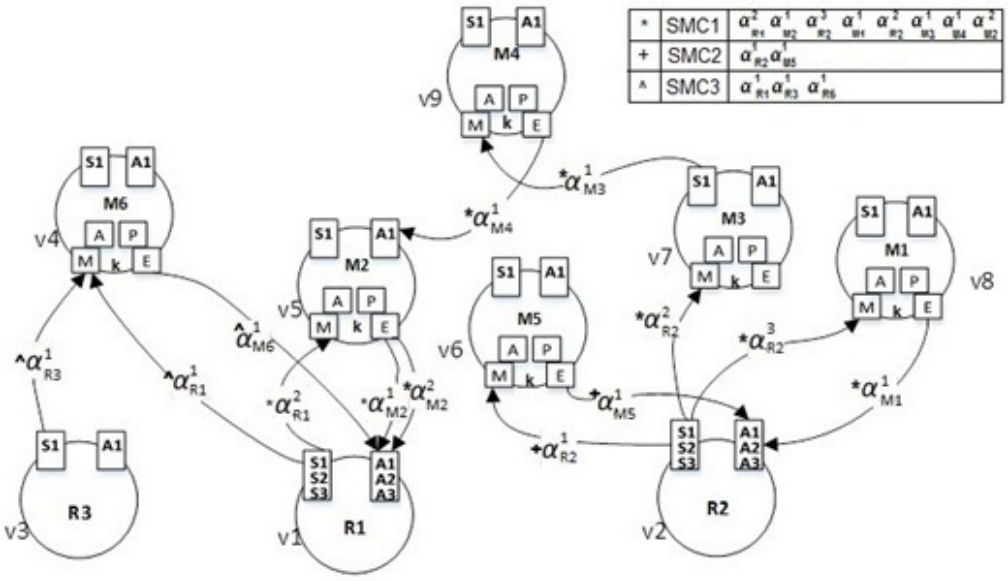

FIGURE 1. A schema interaction graph for an example Autonomic Computing System based on IBM's concept [6], [7], [27] of Autonomic Computing.

self-management capabilities SMC1, SMC2, and SMC3 where the contributing edges of each SMC are distinguished by *, +, and ˆ symbols, respectively. The sequence of execution of interactions for each SMC is given in the tabular form in the upper right corner of Fig. 1.

We formally write an SMC as (1), in terms of the modules involved in the interaction graph and related interaction weights.

$$SMC = \{(D_{M_i}, w_{M_i}, \alpha^k_{M_i}) | 1 \leq i \leq p\} \cup \{(w_{R_j}, \alpha^{k'}_{R_j}) | 1 \leq j \leq q\} \cup w_{R_e} \quad (1)$$

- *Here D_{M_i} represents the average decision complexity of a manager, quantified in Section IV-D*
- *Whereas w_{M_i} and w_{R_j} represents the memory complexity weights of manger M_i and resource R_j, respectively. Both are quantified in Section IV-F*
- *Also, 'p' and 'q' represent the number of managers and resources, respectively, each contributing an interaction inside the SMC.*

All managers M_i and resources R_j inside an SMC originate an interaction, except the last resource R_e where SMC ends. Thus, the length of an SMC is the number of interactions it consists of. The simplified form of an SMC, in terms of interaction edges originating from nodes of the schema graph, is defined as (2).

$$SMC = \{\alpha^k_{M_i} | 1 \leq i \leq p\} \cup \{\alpha^{k'}_{R_j} | 1 \leq j \leq q\} \quad (2)$$

An SMC initiates from a resource and ends at the same or some other resource. The self-manageability index of an autonomic computing system is the number of distinct SMCs inside the system.

IV. DESIGN QUALITY METRICS FOR ASSESSMENT OF A SELF-MANAGEMENT CAPABILITY

In this section, we define design quality metrics which focus design quality assessment of an SMC from its schema interaction graph. These measurements cover perturbation complexity, decision complexity, perturbation resolving load, perturbation resolving time, related data channel and memory requirements of an SMC.

A. PERTURBATION RESOLVING INTERACTIONS COMPLEXITY (PRIC) METRIC

The restoring effect against a perturbation can be determined by calculating the length of a self-management capability from the schema graph. PRIC metric calculates the length of a self-management capability. By taking weight value $\alpha = 1$ for the interactions involved in a perturbation, PRIC value can be calculated using (3).

$$PRIC_{SMC_y} = Length_{SMC_y} = \sum_{i=1}^{p} \alpha^k_{M_i} + \sum_{j=1}^{q} \alpha^{k'}_{R_j} \quad (3)$$

The PRIC value for an SMC is simply calculated to be 'p+q', for each $\alpha^k_{M_i} = \alpha^{k'}_{R_j} = 1$.

More value of PRIC shows an increase of complexity which effects understandability & maintainability. Complexity scale for PRIC is defined considering Chidamber's [28] work on software complexity. PRIC scale is drawn with the following ranges.

$$Low \leq 5 > Medium < 10 \geq High$$

In general, more length of an SMC means restore will take more time. Hence, latency will be observed during recovery. Each extra interaction costs time to waste and delay.

PRIC value can be reduced by combining multiple extra interactions into a single interaction. PRIC metric helps to choose a better option amongst the number of available solutions against an anomaly.

B. DATA CHANNEL REQUIREMENT (DCR) METRIC

Data channel requirement of an ACS is based on identifying the maximum interaction weight from all of the interactions of all SMCs. DCR metric chooses maximum interaction weight value from two sets of interaction weights (i.e. M set of interactions consisting of all interactions originating from manager modules, and R set of interactions consisting of all interactions originating from resource modules), as given in (4). The maximum interaction weight value shows the maximum size of data that may transfer inside ACS.

$$DCR = max(max(\alpha^{k}_{M_i}), max(\alpha^{k'}_{R_j})) \tag{4}$$

DCR calculations help to determine the bandwidth required in distributed systems and paging size in local systems. More value of DCR means an increased cost of bandwidth purchase. DCR metric helps to decide either we can afford the cost of a self-management capability or not. While calculating DCR, one can identify any extra information being sent over the most costly link and can reduce it. Miscalculations in DCR effects efficiency of the ACS.

C. PERTURBATION RESOLVING DATA LOAD (PRDL) METRIC

PRDL metric calculates network traffic generated by an SMC against a perturbation. It guesses burden over the system in terms of the size of messages generated for each SMC, as given in (5). The worst case is when all SMCs activate concurrently, given in (6).

$$PRDL_{SMC} = \sum_{i=1}^{p} \alpha^{k}_{M_i} + \sum_{j=1}^{q} \alpha^{k'}_{R_j} \tag{5}$$

$$PRDL_{ACS} = \sum_{y=1}^{N} SMC_y \tag{6}$$

Recovery Performance of an ACS is inversely proportional to PRDL. More value of PRDL means the system will go slow or even can be stuck, if the traffic load is too high than the system capacity. PRDL can be controlled by checking if there is any extra information being sent in a message, otherwise system resources has to be increased to keep the system working.

D. RULE-BASE COMPLEXITY AND DECISION COMPLEXITY METRICS

For traditional software, the control flow graph is used to determine the number of available decision paths. In the case of a rule-based system, the physical order of rules firing is determined by the inference engine. The rule-based system establishes more dynamic search paths as compared to conventional software [29], [30]. To find the number of available logical paths at design time, Kiper [29] suggested sketching of logical path graph (LPG) of rule-base and extended the work of McCabe [31], [32] to cover AND/OR combinations of rules while sketching LPG. LPG is the rule-base equivalent of a program's control flow graph. Once the LPG of a rule-base is sketched, the rule-base complexity of the autonomic manager is computed as the number of logical paths present in the LPG. The rule-base complexity is measurable using McCabe's cyclomatic complexity metric, as given in (7).

$$Rulebase\ Complexity = nP_{RS} = E - N + 2 \tag{7}$$

- *nP_{RS} is the number of paths in rule-base of an Autonomic Manager.*
- *E is the number of edges and N is the number of decision nodes inside the LPG.*

By determining the number of paths in the LPG of a rule-set, the rule-base complexity metric determines the effectiveness of rule-base. We define the decision complexity of a rule-base as the number of decisions made by the most lengthy path inside LPG. The decision complexity metric is defined in (8).

$$Decision\ Complexity = max(nD_{Path_i}) \text{ for i=1 to } nP_{RS} \tag{8}$$

The average decision complexity of a rule-base is defined as a ratio of the total number of decisions made in all paths to the number of paths in the LPG of rule-base. Let nD_{path} be the number of decisions made in a specific path of LPG and nP_{RS} be the number of paths in the rule-base. The average decision complexity is defined in (9).

$$Average\ Decision\ Complexity = \sum_{i=1}^{nP_{RS}} (nD_{Path})/nP_{RS} \tag{9}$$

E. PERTURBATION RESOLVING TIME (PRT) METRIC

Perturbation resolving time metric is defined by combining the basic concepts of different research works [29]–[31], [33], [34]. PRT counts the number of average decisions made inside each of the p numbers of managers involved in an SMC, and adds the number of interactions originating from each of the p managers and q resources involved in the SMC. PRT consider that each of the interaction and the decision takes a unit time value. Thus, taking $\sum_{j=1}^{q}(1) = q$, the PRT metric is defined as (10).

$$PRT\_SMC = \sum_{i=1}^{p} (D_{M_i} + I) + q \tag{10}$$

It calculates recovery time for a perturbation. From the Chedembra's complexity [28], we define best range for PRT as, ''p*5+p+q''. PRT metric helps to evaluate the performance of autonomic logic at design time. With PRT, we can choose a better option amongst the number of solutions.

F. AUTONOMIC LOGIC'S MEMORY REQUIREMENT (MR) METRIC

A managed resource shares control over its running behavior in the form of Set and Get methods for 'context attributes' [9]. For a context attribute 'C', its memory

weight w(C) is the number of bytes required for its storage in memory. For a managed resource R_k, its memory weight $w(R_k)$, is the memory space required by distinct context attributes of the resource and the resource's touchpoint (TP). Whereas, TP's memory weight is the summation of memory weight of all sensors and effectors inside the TP. For a sensor S_i, its memory weight $w(S_i)$ is the local memory required to retrieve and process values of context attributes. For an effector E_j, its memory weight $w(E_j)$ is the local memory required to receive and process action commands. Thus, for the resource R_k, with 'c' number of context attributes, we define its memory weight as given in (11).

$$w(R_k) = \sum_{l=1}^{c} w(C_l) + \sum_{i=1}^{s} w(S_i) + \sum_{j=1}^{e} w(E_j) \tag{11}$$

For an autonomic manager M_u, its memory weight $w(M_u)$ is the memory space required by MAPE-K loop of the autonomic manager. Memory weight of MAPE-K loop is the sum of memory space required by context parameters, thresholds, system-state variables, symptoms, policy, and action related variables used by the MAPE-K loop. $w(M_u)$ also includes memory space required by the manager's touchpoint. Thus, we define memory weight of a manager as given in (12).

$$w(M_u) = w(CA_{(\text{MAPE-K})}) + w(TP) \tag{12}$$

Memory Requirement of total autonomic logic inside an autonomic computing system is the summation of memory required by its all managers and resources. Thus, by combining (11) and (12), the memory factor for an ACS is defined in (13).

$$MR_{ACS} = \sum_{u=1}^{m+n+1} w(M_u) + \sum_{k=1}^{l} w(R_k) \tag{13}$$

Calculating the memory requirement at design time helps to guess memory demand of autonomic logic when the computing system executes.

G. SMC'S COST- EFFECT AND SUITABILITY OF THE EXECUTION ENVIRONMENT

A self-* service can be dynamically adapted inside an autonomic computing system from different sources like a local system, server, or cloud. An autonomic computing system can opt for a self-management capability from any of these sources. To determine the cost-effect of an SMC for an execution environment, the proposed suit of metrics can be used. The execution cost for a self-management service can be formalized as (14).

$$\begin{aligned} Cost(SMC) = {} & PRIC * UC_{PRIC} + PRDL * UC_{PRDL} \\ & + DC * UC_{avg(DC)} + PRT * UC_{PRT} \\ & + DCR * UC_{DCR} + MR * UC_{MR} \end{aligned} \tag{14}$$

The description of the terms used in the above equation is given below.

$PRIC$: Perturbation Resolving Interaction Complexity,
UC_{PRIC}: Unit Cost for $PRIC$,
$PRDL$: Perturbation Resolving Data Load,
UC_{PRDL}: Unit Cost for $PRDL$,
DC: Decision Complexity,
$UC_{avg(DC)}$: Unit Cost for $avg(DC)$,
PRT: Perturbation Resolving Time,
UC_{PRT}: Unit Cost for PRT,
DCR: Data Channel Requirement,
UC_{DCR}: Unit Cost for DCR,
MR: Memory Requirement,
UC_{MR}: Unit Cost for MR

To determine the most appropriate execution environment (local, server, or cloud) of the application, boolean suitability of an execution environment can be judged for four characteristics: data channel requirement; perturbation resolving data load; memory demand factor; and perturbation resolving time. We define suitability metric as (15), to check pass/fail for each of the four characteristics. A single 'fail' outcome for any of the four characteristics means execution environment is unsuitable.

$$Suitibility = \begin{cases} yes & \text{if } a - b < c \\ no & \text{otherwise} \end{cases} \tag{15}$$

Here, 'c' represents the percentage of resource required. 'a' is the total availability of a resource and 'b' denotes current utilization of the resource. So 'a-b' means the free percentage of the resource.

At the first step, the system evaluates the suitability of the local environment. If it is suitable, the system assigns the environment to the application. Otherwise, it checks the suitability of the server and if required it checks the suitability of cloud. In this way, the system can use these metrics in a bottom-up manner to assign the execution container to a self-*service.

V. EXPERIMENTS & RESULTS

A. CASE STUDY FOR METRICS VALIDATION

To validate the proposed metrics, a 'Stock Trading and Forecasting System' and a 'metrics evaluation framework' were developed using scripting languages, javascript and hypertext preprocessor (PHP ver 5.4) and python (ver 3.7.1). Structured query language (MYSQL ver 5.6) was used for the management of the stock trading database. The case study system was designed as a highly available, self-managing application which runs over a virtualized serverFarm to offer self-* services without interruption. We have used the 'Stock Trading and Forecasting' case study due to three primary reasons. First, as the system requires high availability, in case of a failure the system should recover itself automatically because the crashes are not acceptable at real time. Second, the system has different natural use cases that require the autonomic computing. The self-* capabilities included in the design are autonomic stock trade forecasting service, installed application protection, server-to-database connectivity control, server farm & load management, and

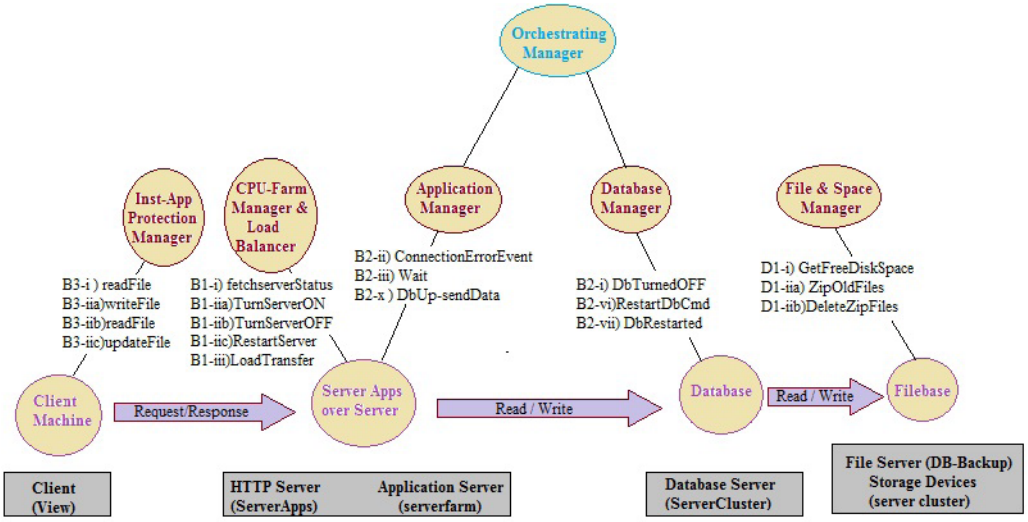

FIGURE 2. Interaction flow graph for selected self-* capabilities.

activity-log memory management. These features are based on the algorithms and concepts given in [35]–[37]. Third, the same case study has been used in previous research [38]. We get all these self-* features benchmarked by experts for the expected values of the proposed metrics. The average values have been used for metrics validation.

B. EXPERIMENTAL RESULTS

We performed experiments with five self-* services of the stock trading system to evaluate the proposed metrics. The detailed working and relevant results of each autonomic manager (delivering a self-* service) are given in sections (V-B.1) through (V-B.5). The interaction graph for the introduced self-management capabilities is given in Fig. 2. The graph represents the perturbation effect whenever a specific self-* capability gets activated to cover-up the respective anomaly behavior. For example, in the case of SMC- B1, ServerFarm & load management, the CPU-farm load manager continuously monitors the serverFarm. In step (B1-i), it fetches the status of server instances inside the serverFarm. It then executes its rule-base to determine the required action, and apply one of the three actions (B1-iia / B1-iib / B1-iic) to start/stop/restart a server instance, respectively or it directly jumps to step (B1-iii) and transfers the processing task to one of the server instances.

To evaluate the proposed metrics, we first identified the resources (R) and managers (M) inside each SMC. Then the context attributes (C) and related sensors and effectors were identified for each manager and resource. Next, the memory weight values for each of the identified elements were distinguished. And then interactions involved in each of the SMCs, related interaction data weights, and decision complexities were determined. Metrics were evaluated and refinements were made in the design of each SMC. The description of each SMC and related calculation are given below.

1) ServerFarm & LOAD MANAGEMENT SERVICE

This self-management service is to control the virtual environment consisting of five virtual machines (VMs). The serverFarm load manager activates or deactivates VMs with an increase or decrease of the processing load. Server load is considered as the number of user tasks queued at a server [39]. Table 1 describes serverFarm management SMC and related evaluations. ProcessesQueue and serverFarm were taken as the resource under observation.The serverFarm manager was the autonomic manager controlling the serverFarm. QueueLoad and userTasks were taken as the context attributes for ProcessesQueue. ServerQueueStatus, userTaskID and serverInstanceID were the context attributes for serverFarm. FetchServerStatus, activateRule-base, and callEffector were the context attributes inside the MAPE-K loop of serverFarm manager. The sensors and effectors defined for each resource's touchpoint are given in Table 1.

Red marked tasks represent the basic execution sequence. The use of blue method calls is situational, whereas the usage of green methods is seldom. A 15 number of interactions are counted for routine activities involved. Perturbation resolving data load is determined to be 2048 bytes. The total memory bytes required by this SMC are 6629 bytes. Next, we calculate the number of paths and the decision complexity of the rule-base for a one-time execution of the SMC. Based on fuzzy logic, the rule-base of the autonomic manager is given in Table 2, (taken from the author's previous work [38]).

Logical path graph for the execution of rules (of Table 2) is presented in Fig. 3. Rules are denoted with 'R' inside LPG. The LPG shows seven numbers of logical paths which takes it to the END state, (note: The rules that could not be fired or which may not take the system to END state were eliminated). An arc over R3 represents, 'AND' condition to show R3 will be triggered only when both R2a, R2d gets active. Arc over R3 means it is a single path from start to end

TABLE 1. ServerFarm management capability.

Resource	Context Attribute(CA)	w(CA)	Sensor (S)	w(S)	Effector (E)	w(E)
Process Queue	QueueLoad	4	sendQueueLoad	4	finishTask	10+4
	UserTask	1024	sendTask	1024	delayTask	9+4
ServerFarm	ServerInstanceID	4	sendInstanceID	4	serverON/OFF	17
	ServerInstance Load	4	sendServerX Load	4+4	transferTask	24
					resetInstance	15
	UserTask	1024	sendTask	1024	alotResources	1024
Manager	**MAPE-K**	**W(MAPE-K)**	**Sensor (S)**	**W(S)**	**Effector (E)**	**w(E)**
Server Farm Manager	queueLoadReq	12				
	userTaskReq	11				
	getInstancesLoad	40				
	policyExecution	255				
	alotServertoTask	1024				
	transferTask	24				
	serverON/OFF	17				
	resetServer	15				
		3458		2064		1107

TABLE 2. Rule-Base of server Farm manager for load management.

Rule 1	IF no server is active THEN Turn initial Server ON
Rule 2	IF a processing task is received and ServerFarm is active THEN transfer user task to a server (having min load) in ServerFarm
Rule 2a	IF Server X Queue is Full THEN set Server X as Overloaded
Rule 2b	IF Server X is not listening to request THEN Server X is Hanged
Rule 2c	IF Server X has completed it's all tasks THEN Server X is IDLE
Rule 2d	IF ServerX Latency is crossing the LatencyThreshold THEN set Server X as Overloaded
Rule 3	IF all Server are overloaded in serverFarm THEN TurnON a New Server in ServerFarm
Rule 4	IF ServerX is hanged in serverFarm THEN Restart the ServerX
Rule 5	IF a ServerX is IDLE in serverFarm THEN TurnOFF the idle Server
Rule 6	IF deadline of a job cannot be met THEN Dispatch the job to a Lightly-loaded Processor
Rule 7	IF Lightly-loaded Processor THEN say YES to migration request
Rule 8	IF ServerX is under Load and serverY accept migration request THEN transfer serverX jobs to serverY
Rule 9	IF more than one server is active with Server X having Load i and Server Y having Load j and $i + j \leq 50$ % THEN if $i \leq j$ transfer the load of Server Y to Server X Otherwise transfer Load of Server X to Server Y
Rule 10	IF alive Servers Count is equal to available servers THEN increase ServerX OverLoad threshold
Rule 11	IF alive Servers Count is less than available servers THEN decrease ServerX OverLoad threshold
Rule 12	IF all Servers' OverLoad threshold is 100% THEN respond user with server busy message

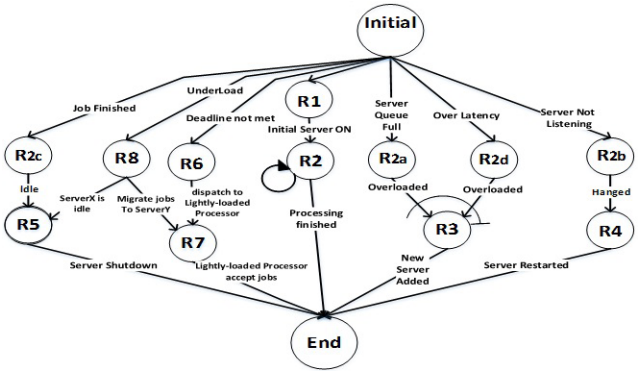

FIGURE 3. Logical path graph for rule-base of ServerFarm&Load manager.

nodes. The number of logical paths in LPG are counted as 8, representing the rule-base complexity. Decision complexity of 'serverFarm & load management control' is determined to be 4 which is the lengthiest path, i.e. Initial->R1->R2->R2->End.

2) STOCK TRADE FORECASTING SERVICE

The forecastManager predicts the future value for a company's stock and financial trade. The service was developed to help users in deciding whether to sell or buy shares. There are nine numbers of context attributes, divided into 5 groups, to get stock market data for prediction generation. Context attributes set for stock trade prediction consists of given period's close-ups and close-downs, most recent closing price, lowest and highest of previous 14 trading sessions (L14, H14), high, low and closing stock prices, and volume traded. Design quality measurements for trade forecasting SMC are given in Table 3.

Interactions involving basic executions are marked with red color. The use of methods shown in purple color is

TABLE 3. Design quality measurements for trade Forecasting SMC.

Resource	Context Attribute(CA)	w(CA)	Sensor	W(S)	Effector (E)	w(E)
Stock Trading Database	CloseUp	4	SendRSI	4	setCloseUp	4
	CloseDown	4	SendSO	4	setCloseDown	4
	RecentClosingPrice	4	sendW%R	4	setRecentClosingPrice	4
	Lowest of Prev 14 Trading Sessions	4	SendMFI	4	setLowestofPrev14	4
	Highest of Prev 14 Trading Sessions	4	sendMACD	4	setHighestofPrev14	4
	Highest Price	4			setHighestPrice	4
	Lowest Price	4			setLowestPrice	4
	Closing Price	4			setClosingPrice	4
	Volume Traded	4			setVolumeTraded	4
Manager	**MAPE-K**	**W(MAPE-K)**	**Sensor (S)**	**W(S)**	**Effector (E)**	**w(E)**
Stock Forecasting Manager	RSIReq	6				
	SOReq	5				
	W%Rreq	6				
	MFIReq	6				
	MACDReq	7			DisplayForecast	4
	generateForecast	48				
	sendForecast	4				
	callEffector	21+4				
		143		20		40

optional and not involved in regular calls. A 12 number of interactions are counted for the routine procedure. Perturbation resolving data load was determined to be 58 bytes. The total memory bytes required by this SMC are 203 bytes. The number of logical paths in the logical path graph of the rule-base of a 'stock trade forecasting manager' was determined to be 2. The decision complexity was calculated to be 6 as the number of decisions made in the most lengthy branch of LPG.

3) INSTALLED APPLICATION PROTECTION

This service keeps track of installed application against corruption or deleting of files. It recovers the files from the backup to keep the system running. In ServerAppProtection SMC, installed and backup applications were taken as the resources under observation. Protection manager was the autonomic manager in action. FileCount and FileHashValue were taken as context attributes. The context attributes inside the MAPE-K loop of protectionManager were getFileCount, getFileHash compareFiles and update-deleteFile. Four numbers of sensors and three numbers of effectors were defined inside the resource touchpoint. The calculations were performed with the following considerations.

- The file hash was calculated with SHA256.
- Variables were considered of long int category with 4-byte memory.
- Ethernet MTU size was taken as 1500 bytes.

These calculations determine the minimum requirements of memory and the number of interactions among autonomic modules. The resultant values are given in Table 4.

The interactions involved in the 'installed application protection' SMC were counted to be 14 and are marked red. The perturbation resolving data load was determined to be 7714 bytes. The total number of memory bytes required by this SMC was 7978 bytes. The number of logical paths was determined to be 3 and the decision complexity was calculated to be 4.

4) SERVER-TO-DATABASE CONNECTIVITY CONTROL

This self-management service keeps track of the server and database connections. In case of any interruption, it tries to reconnect. In case of any hang-ups or failures, it restarts the services to keep the connection live. In server-to-database connectivity SMC, the resources under observations were application writing to the database and the database server. The context attributes for application writing to the database was DB-connectionState. The context attributes for the database server was DB-instanceState. Two number of sensors and three number of effectors had been defined inside the resource touchpoint. The initial stage calculations are given in Table 5.

The number of interactions involved in the connectivity management SMC was counted to be 8 (marked red). The perturbation resolving data load was determined to be 11316 bytes. The total number of memory bytes required by this SMC was 18048 bytes. The number of logical paths was 8, whereas the decision complexity calculated was 5.

5) ACTIVITY LOG MEMORY MANAGEMENT

Continuous client requests to trading database fill the log memory. This service archives the old files to clear the required space. In log-memory management, file-system was the resource under observation whereas freespace Manager

TABLE 4. Installed application protection capability.

Resource	Context Attribute(CA)	w(CA)	Sensor (S)	W(S)	Effector (E)	w(E)
Installed Application	fileCount	4	sendFileCount	4	fileOverwrite	1500
	fileHashValue	64	sendFileHash	64	fileDelete	255
Backup(B) Application	FileBCount	4	sendBFileCount	4	sendFileCopy	1500
	fileBHashValue	64	sendBFileHash	64		
Manager	**MAPE-K**	**W(MAPE-K)**	**Sensor (S)**	**W(S)**	**Effector (E)**	**w(E)**
Application Protection Manager	getFileCount	12+255				
	getFileHash	11+255				
	getBFileCount	13+255				
	getBFileHash	12+255				
	compareFilesHash	128				
	getFileCopy	1500				
	fileOverwrite	1500				
	Filedelete	255				
		4587		136		3255

TABLE 5. Server to database connectivity management capability.

Resource	Context Attribute(CA)	w(CA)	Sensor (S)	W(S)	Effector (E)	w(E)
Application writingToDB	Database Connection State	255	Send CBE DbInstance Down	1024	reconnectToDb	1024
Database Server	Database Instance State	255	WriteToDb Log	1024	RestartDbInstance, StopDbInstance	1024
Manager	**MAPE-K**	**W(MAPE-K)**	**Sensor (S)**	**W(S)**	**Effector (E)**	**w(E)**
Database Manager	Read-DB-Log	1024				
	GenerateCBE	1024+26				
	Extract-DB-State					
	RestartDBInstance	1024				
Application Manager	ReadDBConnectionCBE	1024	Send Connection State	1024	recieveDBState	1024
	ReadDBInstanceState	26				
	ActionPlanner	1024				
	NotifyApplication					
Context Outputter Manager	Read DBLog	1024	OutPutter	1024		
	GenerateCBE	1024+26				
	ExtractDBState					
	NotifyWebServiceManager	26				
webService Manager	Read-CBE	1024			RecieveCBE	1024
	ExtractDBState	1024+26				
	NotifyApplicationManager					
		9856		4096		4096

was the autonomic manager to manage the disk space and log data. The stage1 calculations are given in Table 6.

The number of interactions involved in the connectivity management SMC was counted to be 3 (marked red). The perturbation resolving data load was determined to be 275 bytes. The total number of memory bytes required by this SMC was 554 bytes. The number of logical paths was determined to be 4 whereas the decision complexity was calculated as 3.

C. OVERALL DESIGN EVALUATION AND METRICS VALIDATION

The evaluation framework (developed to evaluate the metrics) was used to evaluate the metrics for five SMCs of serverFarm & loadManager (SF&LM), forecastingService (FS), installedApplicationProtection (IAP), Server-to-database connectivity control (CC), and memory manager (MM). The metrics evaluation results of five SMCs, obtained for each design quality metric, namely, 'Perturbation Resolving Interactions (PRI)', 'Perturbation Resolving Data Load (PRDL)', 'Rule-base Complexity (RbC)', 'Decision Complexity (DC)', 'Data Channel Requirement (DCR)', 'Perturbation Resolving Time (PRT)', and 'Memory Requirement (MR)' are combined in Fig. 4. These obtained values are coherent with known parameters (i.e. expected results, as bench-marked by the experts), which validates our metrics. The results depicted in Fig. 4a to 4g also shows a comparative

TABLE 6. Memory log management capability.

Resource	Context Attribute(CA)	w(CA)	Sensor (S)	W(S)	Effector (E)	w(E)
File System	freeSpace	4	sendFreeBytes	4	Zip/Delete oldFiles	255
Manager	**MAPE-K**	**W(MAPE-K)**	**Sensor (S)**	**W(S)**	**Effector (E)**	**w(E)**
Activity Log Memory Manager	getFreeDiskSpace	16				
	ThresholdTest	4+4				
	ZiporDeletePlan	4+4+4				
	ZipOldFiles / DeleteOldFiles	255				
		295		4		255

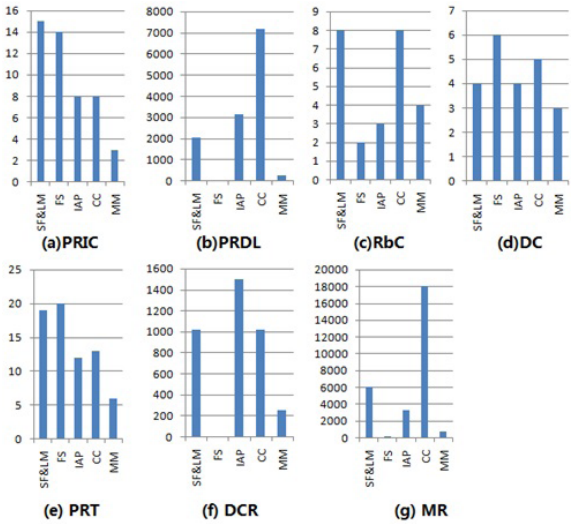

FIGURE 4. Metrics evaluation results of five SMCs, obtained for each design quality metric.

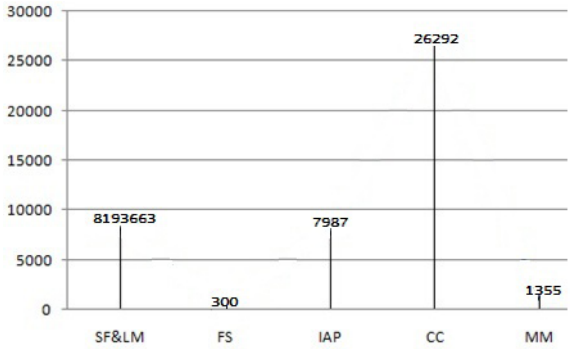

FIGURE 5. Cost-effect value for each of the SMCs.

relation of five SMCs on the base of design quality metrics. The graph bars depict that which SMC is most or less costly on the base of a specific metric.

Also, the design of an SMC can be optimized during the metrics evaluation process. As an example, we discuss the findings for stock trade forecasting SMC.

Interaction Complexity can be reduced by reducing the number of extra interactions. For example, we grouped the nine number of interactions for all of the nine number of context attributes, into three sets in prediction generation capability. It was done by implementing the sensors that send grouped data over an interaction.

Perturbation Resolving Data Load actually requires 5 * 4 = 20 bytes of memory for generating a sell or buy signal. However, when the prediction is to be tested for future values, the CAs value needs to be set which requires 14 *4 = 56 bytes of memory.

Data Channel Requirement for the forecasting manager is 4 bytes in the case of 14 interactions. However, if we combine interaction data into groups to reduce interaction complexity, data channel requirement will increase. Both of these are inversely proportional. If one decreases, the other one will increase. It is up to the designer to decide about what suits as per available resources.

Perturbation Resolving Time is directly proportional to Interaction Complexity and the number of decisions per SMC. By decreasing the Interaction Complexity, Perturbation Resolving Time can be reduced but at the same time, the Data Channel Requirement will be increased.

D. SMC'S COST-EFFECT EVALUATION AND FRAMEWORK FOR SUITABILITY DETERMINATION

Although in practice, a service provider and admin of the autonomic system have to set an agreement for cost value of each attribute in (14). However, here we take unit cost (UC = 1) value for each of the attributes, i.e. $UC_{PRIC} = UC_{PRDL} = UC_{DC} = UC_{PRT} = UC_{DCR} = UC_{MR} = 1$, to determine the cost-effect of each of the five self-management capabilities. The cost-effect value determined for each of the SMCs, is given in Fig. 5. The determined execution cost for a self-management service helps the autonomic system administrator in deciding either it is suitable to run the service in local environment or to be executed from a cloud service provider.

In future, we are going to develop a framework based on equations (14 and 15), to determine the suitability of execution source and the cost-effect of a self-* service. The framework will help to keep a check over the cost and in hand resource utilization for some self-* service and to determine

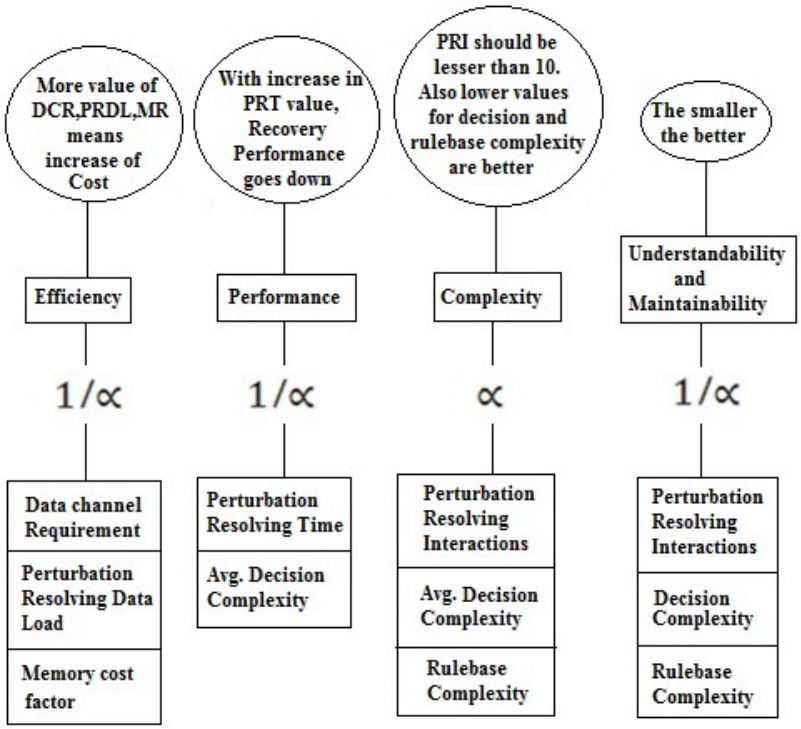

FIGURE 6. Design quality metrics for autonomic computing systems and their mapping to the design quality parameters.

a suitable execution environment (local, server, or cloud) for the application.

E. METRICS MAPPING TO DESIGN QUALITY PARAMETERS

To affirm the importance of our newly defined metrics, we map the defined suite of metrics to the ISO-9126 design quality parameters. This will help to compare differently designed SMC solutions for complexity, efficiency, performance, understandability and maintainability. The metrics, namely, Data Channel Requirements, Perturbation Resolve Data Load, Memory Requirement, are inversely proportional to the efficiency, performance, complexity, understandability and maintainability. Whereas, the metrics, namely, Perturbation Resolving Interactions, Decision Complexity, Rule-base Complexity are directly proportional to complexity and inversely proportional to understandability and maintainability. Metrics Mapping to design quality parameters with interpretation is depicted in Fig. 6. The symbol α represents a directly proportional relation whereas the symbol $1/\alpha$ denotes inversely proportional relation.

The interpretations of the mappings are taken as follows.

Efficiency: More value of the efficiency related metrics (Data Channel Requirement, Perturbation Resolving Data Load, and Memory Factor) means an increase in cost and hence a deficiency in resource usage is observed.

Performance: An increased value of Perturbation Resolving Time metric shows recovery performance going down.

Complexity: Perturbation Resolving Interactions should be lesser than 10, and lower values for Decision Complexity and Rule-base Complexity are better.

Understandability and Maintainability: Smaller values for Perturbation Resolving Interactions, Decision Complexity, and Rule-base Complexity are better for understandability and maintainability.

VI. DISCUSSIONS

The work raises four primary research questions. To address the first question, how to formally design Autonomic Computing System (ACS), we proposed an interaction graph (Fig. 1) based on IBM's model of autonomic computing. The graph represents an ACS in terms of its self-management capabilities. The required sequence of interactions (between modules of the system) against a specific anomaly can be determined at design time using this graph. Thus, it is useful for the design quality assessment of SMCs and in turn of the ACS.

The second question was how to evaluate the self-management capabilities. To address the second research question, we developed seven design quality metrics (section IV-A to section IV-F) using the schema interaction graph (Fig. 1). We designed an autonomic computing scenario, stock management system, with five self-management capabilities to validate the proposed metrics. The system was designed and developed with known parameters such as interaction complexity, decision complexity, rule-base complexity, bandwidth requirements, data traffic, recovery time, and memory complexity. Then, the system was evaluated through five different case studies. In each case study, all the proposed metrics were used to calculate their

corresponding values (Fig. 4). These results gave insights about autonomic systems of the stock exchange which were very much coherent with the expected results (known parameters). The PRIC metric (Fig. 4a) determines the complexity of an SMC which in turn represents the recovery time of a perturbation. The PRIC value of an SMC is directly proportional to the understandability and maintainability efforts. The PRDL metric (Fig. 4b) determines the communication burden over the system when an SMC activated. It is identified that the communication load is inversely proportional to the recovery performance. The rule-base complexity metric (Fig. 4c) determines the effectiveness of a rule-base in terms of how many autonomic solutions it contains. With the decision complexity metric (Fig. 4d), all SMCs can be sorted as per their decisions factor. Average decision complexity metric measures the average number of decision to be made by the system for all SMCs. The PRT metric (Fig. 4e) calculates the recovery time of an SMC which in turn represents the performance of the SMC. The DCR (Fig. 4f) metric informs the bandwidth requirement for an SMC execution. DCR value is inversely proportional to the cost efficiency of an SMC. The MR metric (Fig. 4g) identifies the efficiency of an SMC, and the cost is also increased when MR value is increased. These calculated values were compared with the known parameters which proved that these proposed metrics are successfully evaluating the autonomic system.

The third research question was to calculate the cost-effectiveness and suitable execution environment. Using the proposed metrics, we defined two functions that determine the cost effect (refer to (14)) and suitability (refer to (15)) of the execution environment for a self-* service. We calculated the cost for all five case studies (Fig. 5). Further, a framework can be developed on the base of the proposed functions to keep a check over the cost and in hand resource utilization for some self-* service and to determine a suitable execution environment (local, server, or cloud).

Finally, the last research question has been answered in section V-E, where we mapped the metrics to ISO-9126 design quality parameters (Fig. 6) to compare different design solutions for complexity, efficiency, performance, understandability, and maintainability. We concluded that the proposed metrics are useful for quantitative assessment of the design quality of an autonomic computing system in terms of its self-management capabilities and helpful to compare the differently designed solution of an SMC.

VII. CONCLUSION AND FUTURE WORK

In this research, metrics were proposed to measure the design quality of self-* capabilities for autonomic computing. To validate the design quality metrics, a case study of stock trading & forecasting was developed with self-management capabilities using the principle of autonomic computing. The results of experiments on the case study SMCs proved that the design quality metrics are useful to measure perturbation resolving complexity (in terms of interaction and decisions complexities), perturbation resolving time, perturbation resolving data load, memory cost incurred and data channel requirements of an SMC. We also showed that a framework can be developed on the bases of these metrics to determine which execution container, a local environment, a server, or a cloud environment, is suitable for the execution of an SMC service. The metrics were mapped to the ISO-9126 design quality parameters suggested for ACSs to compare different designs of an autonomic solution. Finally, we conclude that the proposed design quality metrics are necessary to compare and evaluate design level quality of autonomic computing systems, and these design metrics are useful for the selection of the execution environment.

REFERENCES

[1] C. Raibulet and L. Masciadri, "Metrics for the evaluation of adaptivity aspects in software systems," *Int. J. Adv. Softw.*, vol. 3, nos. 1–2, pp. 238–251, 2010.

[2] E. Kaddoum, M.-P. Gleizes, J.-P. George, and G. Picard, "Characterizing and evaluating problem solving self-*systems," in *Proc. Comput. World, Future Comput., Service Comput., Cogn., Adapt., Content, Patterns*, 2009, pp. 137–145.

[3] M. Kumar and N. Agrawal, "Analysis of different security issues and attacks in distributed system a-review," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 4, pp. 232–237, 2013.

[4] J. A. McCann and M. C. Huebscher, "Evaluation issues in autonomic computing," in *Proc. Int. Conf. Grid Cooperat. Comput.* Berlin, Germany: Springer, 2004, pp. 597–608.

[5] R. Rufus, W. Nick, J. Shelton, and A. Esterline, "An autonomic computing system based on a rule-based policy engine and artificial immune systems," in *Proc. Mod. Artif. Intell. Cogn. Sci. (MAICS)*, Ohio, OH, USA, 2016, pp. 105–108.

[6] IBM-Autonomic Computing, "An architectural blueprint for autonomic computing, version 4," IBM, Armonk, NY, USA, White Paper, 2006, vol. 31, pp. 1–6. [Online]. Available: https://scholar.google.com/scholar?q=%27IBM%20and%20autonomic%20computing%3A%20an%20Architectural%20Blueprint%20for%20Autonomic%20Computing%27%2C%20IBM%20Publication%20%28April%202003%29-http%3A%2F%2F%20www.ibm.com%2Fautonomic%2Fpdfs%2FACwpFinal.pdf

[7] P. Horn, "Autonomic computing: IBM's perspective on the state of information technology," Int. Bus. Mach. Corp., Armonk, NY, USA, Tech. Rep., 2001. [Online]. Available: https://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf

[8] Oxford Dictionaries. *Perturbation*. Accessed: May 1, 2016. [Online]. Available: https://en.oxforddictionaries.com/definition/perturbation

[9] Y. Abuseta and K. Swesi, "Design patterns for self adaptive systems engineering," Aug. 2015, *arXiv:1508.01330*. [Online]. Available: https://arxiv.org/abs/1508.01330

[10] B. A. Miller, "The autonomic computing edge: Keeping in touch with touchpoints," IBM Corp., Armonk, NY, USA, Tech. Rep., Aug. 2005. [Online]. Available: http://www.ibm.com/developerworks/autonomic/library/ac-edge5

[11] E. Gandrille, C. Hamon, and P. Lalanda, "Linking reference and runtime architectures in autonomic systems," in *Proc. Symp. Archit. Definition Eval. (IST)*, Toulouse, France, 2013, pp. 13–14.

[12] S. Singh and I. Chana, "QoS-aware autonomic resource management in cloud computing: A systematic review," *ACM Comput. Surv.*, vol. 48, no. 3, p. 42, 2016.

[13] S. Singh, I. Chana, and M. Singh, "The journey of QoS-aware autonomic cloud computing," *IT Prof.*, vol. 19, no. 2, pp. 42–49, Mar./Apr. 2017.

[14] H.-W. Jung, S.-G. Kim, and C.-S. Chung, "Measuring software product quality: A survey of ISO/IEC 9126," *IEEE Softw.*, no. 5, no. 5, pp. 88–92, Sep./Oct. 2004.

[15] M. Salehie and L. Tahvildari, "Autonomic computing: Emerging trends and open problems," in *Proc. ACM SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[16] D. K. Tyagi, A. Awasthi, and R. Rastogi, "Analysis and design of metrics for autonomic computing," *NIET J. Eng. Technol.*, vol. 1, no. 2, pp. 59–63, 2013.
[17] M. Shoaib, A. Shah, and F. Majeed, "Software design quality metrics for Web based applications," *Pakistan J. Sci.*, vol. 63, no. 1, pp. 20–26, 2011.
[18] O. Mola and M. A. Bauer, "Collaborative policy-based autonomic management: In a hierarchical model," in *Proc. 7th Int. Conf. Netw. Service Manage.*, Oct. 2011, pp. 1–5.
[19] C. Raibulet, "Hints on quality evaluation of self-systems," in *Proc. IEEE 8th Int. Conf. Self-Adapt. Self-Organizing Syst. (SASO)*, Sep. 2014, pp. 185–186.
[20] X. Etchevers, T. Coupaye, and G. Vachet, "Experiences in benchmarking of autonomic systems," in *Proc. Int. Conf. Auto. Comput. Commun. Syst.* Berlin, Germany: Springer, 2009, pp. 48–63.
[21] S. Singh and I. Chana, "Metrics based workload analysis technique for iaas cloud," Nov. 2014, *arXiv:1411.6753*. [Online]. Available: https://arxiv.org/abs/1411.6753
[22] E. Kaddoum, C. Raibulet, J.-P. Georgé, G. Picard, and M.-P. Gleizes, "Criteria for the evaluation of self-*systems," in *Proc. ICSE Workshop Softw. Eng. Adapt. Self-Manag. Syst.*, 2010, pp. 29–38.
[23] P. Lin, A. MacArthur, and J. Leaney, "Defining autonomic computing: A software engineering perspective," in *Proc. Austral. Softw. Eng. Conf.*, 2005, pp. 88–97.
[24] B. Bjorn. *Charateristics of Great Software Design*. Accessed: Oct, 14, 2014. [Online]. Available: http://bybjorn.com/30/
[25] R. C. Calinescu, "Reconfigurable service-oriented architecture for autonomic computing," *Int. J. Adv. Intell. Syst.*, vol. 2, no. 1, pp. 38–57, 2009.
[26] J. Coleman, T. Lau, B. Lokhande, P. Shum, R. Wisniewski, and M. P. Yost, "The autonomic computing benchmark," in *Dependability Benchmarking for Computer Systems*, vol. 72. Hoboken, NJ, USA: Wiley, 2008, p. 1.
[27] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
[28] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994.
[29] J. D. Kiper, "Structural testing of rule-based expert systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 1, no. 2, pp. 168–187, 1992.
[30] Z. Chen and C. Y. Suen, "Measuring the complexity of rule-based expert systems," *Expert Syst. Appl.*, vol. 7, no. 4, pp. 467–481, 1994.
[31] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.
[32] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Commun. ACM*, vol. 32, no. 12, pp. 1415–1425, 1989.
[33] A. Madi, O. K. Zein, and S. Kadry, "On the improvement of cyclomatic complexity metric," *Int. J. Softw. Eng. Appl.*, vol. 7, no. 2, pp. 67–82, 2013.
[34] L. N. Preeti and S. J. Bhadula, "A methodology for obtaining complexity of component-based software," *Int. J. Softw. Eng. Appl.*, vol. 11, no. 4, pp. 1–10, 2017.
[35] E. Manoel, M. J. Nielson, A. Salahshour, K. V. L. S. Sampath, and S. Sudarshanan, *Problem Determination Using Self-Managing Autonomic Technology*. Armonk, NY, USA: IBM International Technical Support Organization, 2005.
[36] K. P. Kumar and N. S. Naik, "Self-healing model for software application," in *Proc. Int. Conf. Recent Adv. Innov. Eng.*, 2014, pp. 1–6.
[37] W. D. Zhu, "High availability implementation for IBM filenet P8 system components," in *IBM High Availability Solution for IBM FileNet P8 Systems* (International Technical Support Organization). Armonk, NY, USA: IBM Red Books, 2009, ch. 5.
[38] A. Jaleel, S. Arshad, and M. Shoaib, "A secure, scalable and elastic autonomic computing systems paradigm: Supporting dynamic adaptation of self-*services from an autonomic cloud," *Symmetry*, vol. 10, no. 5, p. 141, 2018.
[39] P. Mittal, A. Singhal, and A. Bansal, "A study on architecture of autonomic computing-self managed systems," *Int. J. Comput. Appl.*, vol. 92, no. 6, pp. 6–9, 2014.

ABDUL JALEEL received the B.S. degree in computer science and engineering from the University of Engineering and Technology, Lahore, Lahore, Pakistan, in 2006, the M.S. degree in computer science in 2010, and the Ph.D. degree in computer science from the University of Engineering and Technology, Lahore, in 2019, where he is currently an Assistant Professor with the Department of Computer Science, Rachna College. His research interest includes the development of self-managing software applications. His major interests include autonomic computing and software quality measurement metrics.

SHAZIA ARSHAD received the M.S. degree in computer science from Agriculture University, Faisalabad, Pakistan, and the Ph.D. degree from the University of Engineering and Technology, Lahore, Pakistan.

She is currently a Professor with the Department of Computer Science and Engineering Department, University of Engineering and Technology, Lahore, and also serving as the Head of the Department. Her research interests include blockchain, software engineering, and adaptive learning.

MUHAMMAD SHOAIB received the M.Sc. degree in computer science from Islamia University, Pakistan, and the Ph.D. degree from the University of Engineering and Technology, Lahore, Pakistan, in 2006. His Postdoctoral is from Florida Atlantic University, USA, in 2009.

He is currently a Professor with the Computer Science and Engineering Department, University of Engineering and Technology, Lahore. His current research interests include information retrieval systems, information systems, software engineering, and semantic web.

MUHAMMAD AWAIS received the B.S. degree (Hons.) in computer science from Punjab University, and the M.S. and Ph.D. degrees in computer science from the University of Engineering and Technology, Lahore, Lahore, Pakistan, where he is currently an Assistant Professor with the Computer Science and Engineering Department. His research interests include artificial intelligence, reinforcement learning, adaptive eLearning systems, and affective computing.