A Review on Software Performance Analysis for Early Detection of Latent Faults in Design Models

[1]G.Kasi Reddy,
Research Scholar,
CSE, JNTU Hyderabad,
India
gkasireddy@rediffmail.com

[2]Dr. D Sravan Kumar,
Professor and Head Dept. of CSE,
SR International Institute of Technology,
Hyderabad, India.
dasojusravan@gmail.com

[3]Dr. D Vasumathi,
Professor, CSE,
JNTUH College of Engineering,
Hyderabad, India.
vasukumar_devera@yahoo.co.in

Abstract-Organizations and society could face major breakdown if IT strategies do not comply with performance requirements. This is more so in the era of globalization and emergence of technologies caused more issues. Software design models might have latent and potential issues that affect performance of software. Often performance is the neglected area in the industry. Identifying performance issues in the design phase can save time, money and effort. Software engineers need to know the performance requirements so as to ensure quality software to be developed. Software performance engineering a quantitative approach for building software systems that can meet performance requirements. There are many design models based on UML, Petri Nets and Product-Forms. These models can be used to derive performance models that make use of LQN, MSC, QNM and so on. The design models are to be mapped to performance models in order to predict performance of system early and render valuable feedback for improving quality of the system. Due to emerging distributed technologies such as EJB, CORBA, DCOM and SOA applications became very complex with collaboration with other software. The component based software systems, software systems that are embedded, distributed likely need more systematic performance models that can leverage the quality of such systems. Towards this end many techniques came into existence. This paper throws light into software performance analysis and its present state-of-the-art. It reviews different design models and performance models that provide valuable insights to make well informed decisions.

Keywords *– Software performance engineering, UML, design models, performance models*

*****

I. INTRODUCTION

Software performance engineering a quantitative approach for building software systems that can meet performance requirements [54]. It is not just another functional test as conceived by any people. Instead it strives to fill gaps in the software development process [55]. Queuing models can help in software performance engineering [59]. According to Woodside *et al*. [60] performance measurements are pertaining to comparing, load testing, testing, instrumentation, profiling, and tracing while the performance modelling is pertaining to scenario analysis, exploration of options, sensitivity, model generation and high-level models. Poor performance costs millions of dollars to the software industry. The make-or-break quality of software is based on responsiveness, performance, and scalability [8]. Performance testing plays a vital role in improving software quality and usability by unearthing potential faults and rectifying them [2].

UML models can be used to derive performance models [4], [11], [12]. Petriu & Wang [5] explored architectural patterns to derive software performance models. Similarly Williams & Smith [10] used architectural approach to solve performance issues of software. Menascee and Gomaa [6] advocated the usefulness of combining both design and performance models for leveraging quality in C/S systems. Layered performance modelling can have profound impact on software development and delivery [9]. Queuing Network Models (QNM) are widely used for predicting performance of software. Balsamo *et al*. [21] made a very good review on this. To address delay and capacity estimation a scenario based performance engineering model was explored in [22] by employing Use Case Map (UCM) and the need for concurrency. Simulation models were explored in [17] and [23] for software performance modelling. To enhance software quality software refactoring was explored in [24], [89] and [93]. It is the extension to [104] and [102] towards improving accuracy in determination of components that are to be refactored. Moreover refactoring could be part of a quality cycle in software engineering [101]. A good review of refactoring models can be found in [94]. Performance models can also be used to design self managing computer systems with Quality of Service (QoS) controllers [25].

Simulation and algebraic models are used for performance analysis of software architectures and concluded that integration of different techniques is more advantageous [27]. A good survey on model based performance engineering can be found in [29]. According to a survey in software industry 20% of modules cause 80% of faults. This notion was positively proved in Module-Order Model explored by Khoshgoftaar *et al*. [30] for performance prediction. Software performance estimations were made by Johansson & Wartenberg [31] with respect to embedded platform where data is represented in the form of scenarios. Middleware performance attributes can also be incorporated into UML models [33]. A good review of UML based software process modelling languages can be found in [66]. Omari *et al*. [36] explored performance models for layered server systems in the environment containing replicated server equipped with parallel processing. Software performance analysis is made on UML models in [42] using component-based LQN.

TABLE 1

ABBREVIATIONS

LQN	Layered Queuing Network
MSC	Message Sequence Chart
QNM	Queuing Network Model
C/S	Client/Server
VPN	Virtual Private Network
OMG	Object Management Group
MDA	Model Driven Approach
MDSPE	Model Driven Software Performance Engineering
UCM	Use Case Map
SPT	Schedulability Performance and Time
SE	Software Engineering
UML	Unified Modelling Language
OCL	Object Constrained Language
PMIF	Performance Model Interchange Format
XML	Extensible Markup Language
UML-SPT	Unified Modelling Language-Schedulability Performance and Time
EJB	Enterprise Java Beans
CORBA	Common Object Request Broker Architecture
WCF	Windows Communication Foundation
DCOM	Distributed Component Object Model
OLE	Object Linking and Embedding
TDPM	Template Driven Performance Modelling
SysML	Systems Modelling Language
DSE	Design Space Exploration
PETTS	Performance Engineering Tool for Tiered Systems
MVC	Model View Controller
UI	User Interface
SPEDP	Software Performance Engineering Development Process
MOM	Message Oriented Middleware
SPL	Software Product Line
SOA	Service Oriented Architecture

Ganesan & Prevostini [45] proposed a method that fills gap between Systems Modelling Language (SysML) and Design Space Exploration (DSE). The latter is used to gain optimal performance in terms of size, cost, performance and power consumption. The combined model when applied to embedded systems for efficient modelling. Geo & Sair [48] focused on long term bottlenecks that have bearing on performance. They tool for analyzing such bottlenecks improved average performance by 7%. According to Woodside *et al*. [52], the future of performance engineering depends on the comprehensive models that are result of robust combination of many models with convergence between modelling approaches and measurements. Software models and platform models could be integrated for better performance analysis.

Unknown service demand parameters are one of the problems in software performance models. Menasce [56] focused on this issue and provided a method using closed form solution with accuracy in predicted response times. There are some Java Modelling Tools for performance evaluation [103]. Software performance models can also be used to derive performance metrics that can be used to evaluate performance [68]. The rest of the paper throws more details into the software performance modelling.

Our main contribution in this paper is the study and review of the present state-of-the-art on software performance analysis. The design models and performance models which are found in the literature are reviewed to provide insights. The remainder of the paper is structured as follows. Section 2 provides software performance models. Section 3 throws light into performance prediction of component based applications. Section 4 focuses on object constrained language and its impact on SE activities. Section 5 provides performance engineering tools for tiered software systems. Section 6 provides more details on software performance engineering. Section 7 throws light into important discussion and findings while session 8 concludes the paper besides providing directions for future work.

II. SOFTWARE PERFORMNACE MODELS

Software quality characteristics like as performance, reusability, reliability and maintainability are influenced by software architecture. Towards this end Petriu *et al*. [1] proposed Layered Queuing Network (LQN) models based on the UML descriptions. The performance model and system architecture are related. This model was applied to telecommunication system where the model identified the bottleneck movements among components based on the load. Similar case study was explored in [13] and [26] for qualitative analysis of different protocols performance validation respectively. Omari *et al*. [40] also explored LQNs as part of an analytical model for C/S systems. Aquilani *et al*. [3] used Message Sequence Charts (MSC) to derive performance models pertaining to software architectures automatically. In the process Queuing Network Models (QNMs) are automatically derived and evaluated. Modelling of software contention was explored in [7], [14] and [98] using queuing networks and two-level iterative queuing model respectively.

Virtual Private Networks (VPNs) are widely used in the real world for secure communications and they have their influence on performance of networks due to measures like compression and encryption. Pena & Evans [15] evaluated performance of such networks and found that CPU usage and transference speed was degraded due to VPN. Xu *et al*. [16] explored UML profile for performance analysis of software design in terms of time, performance and schedulability. Profile is the language extension for UML given by OMG. Performance targets considered are throughput, response time and utilization of resources. The performance improvement strategy described in [16] makes use of LQN, cloning bottlenecks and other possible solutions. It is an iterative process where bottlenecks are identified and resolved until a satisfactory solution is converged. Normalized utilization is the technique used to determine bottlenecks. The system also distinguishes between real bottlenecks from others. UML Profiles are also explored in [20] and [32] in terms of roles Schedulability Performance and Time (SPT). UML-SPT is a software performance tool explored in [41].

Balsamo & Marzolla [17] proposed a simulation based model for performance evaluation that makes use of architectures available in the form of UML. The simulation model is derived from the UML model that includes workloads, resources, steps, and parameters. The simulation model provides sufficient feedback to the UML model. Thus this approach can iteratively improve the performance of the system. UML Profile is used for annotations as also done in [16]. There are some issues such as statistical uncertainty with UML performance annotations [99]. However, unlike [16], a simulation model is used for performance modelling. A web based video streaming application is used for experiments that revealed that the system satisfies performance requirements.

Traore *et al*. [18] proposed a Model Driven Approach (MDA) to analyze performance of distributed systems. Model Driven Software Performance Engineering (MDSPE) is the main focus of the work of Traore *et al*. The model make use of performance related activities at every phase of life cycle, performance requirements in analysis phase, performance annotation and performance analysis at design level, and performance testing at coding level. Queuing model is employed to for better performance. Different performance models can be integration in software development process. The performance models include Queuing Networks (QN), Stochastic Process Algebras (SPA) and Stochastic Timed Petri nets (STPN). These models do have their own expressiveness in order to realize performance analysis of software systems. Integration of the models can yield synergetic effect in improving quality [28].

III. PERFORMANCE PREDICTION OF COMPONENT-BASED APPLICATIONS

Distributed computing technologies such as Enterprise Java Beans (EJB), DCOM (Distributed Component Object Model), WCF (Windows Communication Foundation), and Common Object Request Broker Architecture (CORBA) help in building component based applications. Wojtek Kozaczynski and Grady Booch (1998) [106] explored component based software engineering. Their article throws light into the need for component based engineering practices in software development. Component can be understood as a reusable software program or piece of code that can be accessed from remote place as well. Moreover the component can be accessed from other platforms as well. A component developed in one language can be invoked from other language program as well. This encourages interoperable software development. The component technology is predominant in case of hardware. In a computer system mouse, keyboard, monitor and all parts are from different companies but still they are working together. This is actually known as componentization of development. This makes people easy to switch to different components based on the requirements. Software development also started using this component technology. It is realized in the form of Enterprise Java Beans originally released by Sun Microsystems, SAP’s component technology, Microsoft’s DNA project and IBM’s San Francisco Project to mention few among the component technologies or projects (Wojtek-1998-1). Microsoft’s Component Object Model (COM) and Distributed Component Object Model (DCOM) are best examples in which software is built in the form of component based software development. The Enterprise Java Beans (EJB) and Web services technologies support component development. Irrespective of the platform in which a component is built, it can be involved from program in any other platform. All distributed component technologies support component development and invocation from a remote place.

Java platform and Microsoft.NET platform provide distributed component technologies. From Java platform EJB, Remote Method Invocation (RMI), Web Services and Java Messaging Service (JMS) are examples of the technologies that support component development. The components that are run in distributed environment can help integrate businesses in the real world irrespective of their technology platforms (Wojtek-1998-1). Object oriented programming and its flexible development with code reusability and resembling real world solutions made it possible to have component based software development (CBSE). CBSE goes a long way in developing useful applications in the real world. The e-commerce applications such as Amazon are leveraging the component technologies in order to have communication among the servers of different vendors seamlessly. Therefore it is essential to promote CBSE kind of programming so as to get benefits from business integration and reusability of software so that time and money can be saved in the software industry and in the real world.

Common Object Request Broker Architecture (CORBA) is another distributed component development platform that helps developing reusable software components (Wojtek-1998-1). Object Linking and Embedding (OLE) is one of the best examples to know how component technology can be leveraged to reuse software. An application might have certain features. Other applications need not to rebuild such features. They can use OLE technology in order to reuse those features without rebuilding it. Therefore the theme behind CBSE is that do not reinvent the wheel. Based on this theme component software development goes on in future also. Towards this end many companies including Object Management Group (OMG) contributed. The concept of reusable software component development will go on in future also. The way hardware components can be replaced with new components, the software components also can be replaced with new ones. Therefore the component development technology is based on the standard interfaces and that will work fine as far as interface standards are followed (Wojtek-1998-2).

Hon Hopkins (2000) [105] focused on the component technology and the need for software component development in order to leverage reusability and gain economic advantages in the real world. Component is defined as unit of composition with standard interfaces in such a way that other components can interact with it. There have been attempts to build reusable software components that can be used quickly to build complex business systems. Reusability ad Ease of Maintenance is the two advantages of the software component development. The object oriented model and component based model go hand in hand. Both complement each other. Thus there is promotion of component based software development. Distributed Component Object Model (DCOM) is the best example from Microsoft that is based on the component technology. However, these components are platform independent and largely limited to Windows Operating System. From OMG Common Object Request Broker Architecture (CORBA) came into existence. This is also a distributed technology that enables components developed in any platform to invoke CORBA components with standard interfaces (Hon-2000-2). The main difference between DCOM and CORBA is that CORBA is platform independent while the DCOM is not. Extensible Mark-up Language (XML) plays a vital role in achieving communication between heterogeneous components.

Unified Modelling Language (UML) also incorporated provision for component based development. A component is a piece of software that can be reused. Moreover, it is interoperable with other components. The components developed in different languages can interact with each other so as to avoid reinventing wheel in future endeavours. The reusability and ease of maintenance are the two major advantages of the component development model. Components are mainly used in distributed applications. This is because the distributed applications run on different machines with different platforms. In this complex scenario, it is essential to have software component that can be used or invoked by other components irrespective of their location with standard interfaces. DLL hell was the problem with component technology using COM and DCOM. This problem is limited to Windows platform. However, Microsoft improved DCOM in order to improve the situation to leverage the component development technologies and their usage in the software industry. However, there are many other issues that are to be overcome in order to speed up the software developing based on component technologies. The issues include availability of different platforms, different architectures, specificity, versioning, quality, and the features that form hurdles for component development (Hon-2000-4).

The performance predication methodology presented in [34] considers two aspects such as modelling container components and application components that work in tandem with container components. The methodology includes performance predication model at design-level. It contains a quantitative performance model that use of inputs from application-independent performance profiles. EJB application was considered to have predictions based on the said methodology.

Web services can also be used to participate in performance modelling. Since web services are the components that can be involved in inter-operable fashion, performance model web service was built in [38]. Performance models might have different format for model information. Smith & Llado [39] explored an interchange format namely Performance Model Interchange Format (PMIF) that facilitates transforming model information from one performance model to another performance model. Thus it is possible to have multiple models in place based on the requirements. The PMIF is based on XML standards. In [44], [65] and [51] similar kind of interchange format was presented. Template Driven Performance Modelling (TDPM) was made in [43] on EJB components.

IV. OBJECT CONSTRAINED LANGUAGE AND ITS IMPACT ON SE ACTIVITIES

Software Engineering (SE) is the branch of computer science which deals with developing good software. Unified Modelling Language (UML) has associated Object Constrained Language (OCL) which is used to make the design models more precise and accurate. Therefore it can be assumed that OCL has its impact on the SE activities. The hypothesis that OCL can improve the precision of software design models was investigated in [35] on three software engineering activities such as impact analysis of changes, functionality of software and the robustness of system logic. The experiments on hypothesis proved that the OCL has significant impact on SE activities and it also leverages the abilities of software engineers. The OCL usage also costs but it depends on the tool being used. The experiments covered defect detection, comprehension and maintenance. Performance evaluation is influenced by certain factors as explored in [37]. For instance the metrics used in performance evaluation are sensitive to different job classes. The measures are not working impartially with different workloads. Therefore it is essential to consider these facts and some workload attributes do not need modelling.

V. PERFORMANCE ENGINEERING TOOLS FOR TIERED SOFTWARE SYSTEMS

Tired architecture is commonly used in software systems. Generally web based applications are built using 3-tier model. That way there are n-tier applications that need to be considered for performance engineering. To evaluate performance of such systems Sharma *et al*. [46] presented a tool. Different tiers might run on different machines in distributed environment. The performance model needs to consider such environment and needs to consider performance issues such as think time of client, range of clients and so on. They used performance analysis tool named Performance Engineering Tool for Tiered Systems (PETTS) developed by them for performance analysis of tiered systems. Enterprise applications with multiple tiers built in either JAVA or .NET platforms can have different factors to be considered for performance evaluation. The factors include security, cost, web services, vendor and dependency. The complexity of tiers brings about performance issues in enterprise applications. There are architectural patterns that could reduce the complexity of enterprise applications. They include Model View Controller (MVC), PCMEF and XWA [49]. Modelling of such systems can be improved or enhanced using LQNs [61].

Web based software systems need certain resources based on the tiers they have. Mean Value Analysis (MVA) algorithm proposed by Boga´rdi-Me´szo¨ly *et al*. can help in building improved performance models as they can reduce computational cost and complexity besides producing measures or metrics with high accuracy [62]. Rule based navigations are included in web applications. Xu [63] proposed a framework that supports performance diagnosis and improvement of role based software systems. The model contains many phases like extract model, solve, generate rules, make decisions and design software. The design level consideration helps in early modification of models for improving quality of software.

VI. FRAMEWORKSFOREFFICIENT SOFTWARE PERFORMANCE EVALUATION AND OPTIMIZATION

Response Surface Methodology (RSM) was proposed in [50] as part of automatic framework for performance evaluation and optimization. The framework was named as Response Surface and Importance Framework (RS-IF). Capacity planning and total cost problems were explored with this tool. There were three servers included for performance model namely UI server, event server and database server. Markov model was used to demonstrate the proof concept. RSM is a mathematical technique used to evaluate performance of software systems. Importance based process can help reduce unnecessary computations. Thakkar *et al*. [57] proposed a framework for automatically building performance models based on measurements. Pelliccione *et al*. [64] proposed a framework known as CHARMY. This framework is used to design and verify architectural specifications. They also had transformation to generate Java code from verified design specifications. Thus the tool is supporting iterative modelling and evaluation of architectures involved in software development.

VII. MORE ON SOFTWARE PERFORMANCE ENGINEERING

Software development process involves a plethora of performance issues. It is essential to evaluate software performance. Mapping the UML model at design phase into a performance model is one of the widely used approaches [91]. Distefano *et al*. [67] came up with guidelines that can be used to have performance models that are ArgoPerformance compliant. The design process is considered as software performance engineering development process (SPEDP) which must use certain notations, rules and guidelines in order to have pre-defined syntax and semantics that can help in evaluation. Since performance is one of the overlooked aspects in software development, it is imperative to have such thing in the design phase itself. From the software architecture model, performance model is built. Then the model is applied to web-based video application. The performance model is thus able to provide required feedback so as to improve the quality of the system.
Performance prediction is the ability to estimate the performance of an application in the design phase. This kind of research was found in [68] where an algorithm was proposed. The algorithm was tested with a web based application in terms of computational complexity, response time, and computation time. Regression techniques are widely used in performance prediction of software. Especially parameter estimation pertaining to software

performance can be achieved using these techniques [70]. Performance modelling or performance predication became more important due to the invent of distributed technologies like Web Services. In [71] and [79] performance engineering was applied a video application that depends on web services. The performance of video provider web service was predicted. It could fix bugs in the design phase. Traore *et al*. [73] explored UML-based performance modelling for applications that are made up of distributed components. Performance annotations are given to every operation in the enterprise application. The annotations are used to model performance evaluation. The model-driven SPE process thus used UML profile for SPT. Similarly networking and OLTP applications can be subjected to performance modelling [74], [75].

Software Product Line (SPL) is nothing but a set of products that are related and formed from a shared set of assets. Tawhid & Petriu [76] derived performance models automatically from SPL. UML Profile MARTE from OMG was used to achieve this. The standard annotations provided by MARTE can help in deriving performance model that can be represented as a LQN. This is applied to a distributed e-commerce application based on variability model of SPL. Consistency rules play a vital role in software performance models. In [77] they are used to present an automated approach that detects inconsistencies in the design of a software system. Palladio architectural models are also used in software engineering. Performance antipatterns might be associated with such models. Trubiani & Koziolek [78] presented a model that could automatically detect software performance antipatterns thus improving performance of the system by 50%. Song *et al*. [80] proposed a learning based mechanism that predicts defect-proneness in given software system. Another important form which is used along with QN models is product-form which was used to represent different systems in communication and production. Service Oriented Architecture (SOA) UML profile was explored in [90] for real time embedded systems in distributed environment. Stereotypes and metamodels were used for achieving performance evaluation. The solution was also applicable to multiple devices that run in a distributed environment. In [97] software performance evaluation is made for an embedded system pertaining to polar satellite antenna control.

VIII. DISCUSSION & FINDINGS

Requirements gathering should also include performance requirements of the system [82] while performance targets are to be identified in the process clearly [83]. The performance requirements can be evaluated using UML-JMT proposed in [85]. Capacity management can also be covered at the time of design [84]. High Performance Computing (HPC) is the area where performance engineering is lacking [69]. Message Oriented Middleware (MOM) is found another area where further research is required. The middleware and its clients work in loosely coupled environment. The dependency among the components can be analyzed besides predicting performance [72]. Product – forms [81] is one of the models used in performance engineering which were used to analyze resource sharing systems. UML to Petri Nets concept was explored for performance modelling in [86]. The model was based on Performance Context Model (PCM). It maps PCM to performance domain. Queuing Petri Nets is an extension to Petri Nets that are very powerful in modelling hardware contention and strategies for scheduling [92]. Incorporating SLAs at design level and achieving a performance model is very challenging problem that needs further research [87]. Software architects can explore performance and cost model besides using design space exploration tool and modern performance models for effectively predicting performance [88]. Performance models can help organizations to choose best hardware and software with expert decision making strategies as explored in [95].
Software performance models can provide automated feedback that can be used to improve systems [95]. This is the motivation for all modern model-driven techniques. Filters such as Kalman Filter are also used with performance models. To estimate parameter and tracking then can be done using filters [100]. Annotated UML model is explored in [91] for software performance modelling. Software product line is the less explored area with respect to software performance modelling [76]. Consistency rules play a vital role in software performance models [77]. From the review of different kinds of performance models it is understood that there are many case studies which were less explored. For instance distributed component models that make use of distributed computing technologies like EJB, DCOM, CORBA and so on can be explored further for performance modelling. There is scope for more research into embedded systems. The emerging technologies like cloud computing and mobile computing technologies are good candidates for further research.

IX. CONCLUSIONS AND FUTURE WORK

In this paper we focused on software performance engineering. We reviewed considerable literature to have insights into the performance aspects of software industry. Interestingly it is understood that performance modelling was the neglected area. Identifying and addressing performance issues play vital role in improving quality of software. Since software performance is one of the most critical factors in the modern software development, this paper throws light into different design models and performance models and the need for performance modelling. The essence of performance modelling in the design phase is to unearth potential bugs in the early stages of SDLC. This will save time, money and effort besides increasing success rate in software development and delivery of time-to-market products. The design models available are UML, Petri Nets and Product-Forms that could be mapped to performance models. Stated differently, the design models can be effectively used to derive performance models. Performance models provide essential feedback for improving quality of software. Model-Driven software process engineering has been around and that needs to be improved keeping performance requirements in mind. In fact, the performance aspects are to be considered in the design itself to ensure that the outcome of the development will meet performance requirements. This research is extended further to have case-studies that were less explored besides proposing new performance models.

REFERENCES

[1] Petriu,Dorina.(2000).Architecture-Based performance Analysis applied to a Telecommunication System. *IEEE*,. 26 (11), p1-17.

[2] Weyuker, J, Elaine. (2000). Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study. *IEEE*. 26 (12), p1-10.

[3] Andolfi, F,. Aquilani, F,. Balsamo, S,. Inverardi, P. (2000). Deriving Performance Models of Software Architectures from Message Sequence Charts. *ACM*, p1-11.

[4] Dr Petriu, Dorina,. (2000). Deriving Performance Models from UML Models by Graph Transformations. *WOSP*. p1-52.

[5] Dorina, C. Petriu and Xin Wang. (2000). Deriving Software Performance Models from Architectural Patterns by Graph Transformations. *Carleton University,Ottawa, ON, Canada*. p1-14.

[6] MenasceÂ , D.A,.. (2000). A Method for Design and Performance Modelling of Client/Server Systems. *IEEE*. 26 (11), p1-20.

[7] MenasceÂ , D.A,. (2001). Simple Analytic Modelling of Software Contention. *George Mason University*. p1-7.

[8] Ph.D Williams, L.G and Ph.D Smith, C.U. (2002). Five Steps to Solving Software Performance Problems. *Software Engineering Research and Performance Engineering Services*. p1-10.

[9] Woodside, M. (2002). An Overview of Layered Performance Modelling.*Carleton University, Ottawa, Canada*. p1-42.

[10] Williams, L.G and Smith, C.U. (2002). PASASM: An Architectural Approach to Fixing Software Performance Problems. *Software Engineering Research and Performance Engineering Services*. p1-15.

[11] Amer, H and Petriu, D.C. (2002). Software Performance Evaluation: Graph Grammar-based Transformation of UML Design Models into Performance Models. *Carleton University*. p1-33.

[12] Petriu, D.C and Woodside, M. (2002). Software Performance Models from System Scenarios in Use Case Maps. *Springer*. p1-18.

[13] Castaldi, M,. Inverardi, P,. and Afsharian, S . (2002). A Case Study in Performance, Modifiability and Extensibility Analysis of a Telecommunication System Software Architectur *IEEE*. p1-10.

[14] Menasc´e, D.A. (2002). Two-Level Iterative Queuing Modelling of Software Contention. *IEEE*. p1-10.

[15] Peiia, J.C.C and Evans J. (2000). Performance Evaluation of Software Virtual Private Networks (VPN). *IEEE*. p1-522.

[16] Jing Xu,. Woodside, M,. and Petriu, D. (2003). Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time. *Carleton University*. p1-8.

[17] Balsamo, S,. and Marzolla, M. (2003). A Simulation-Based Approach to Software Performance Modelling. *ACM*. p1-4.

[18] Traore, I,. and Woungang, I,. (2003). Performance Analysis of Distributed Software Systems: A Model-Driven Approach. *IEEE*. p1-8.

[19] Pavlo Bazilinskyy and Markus Brunner. (n.d). Performance Engineering and Testing. *IEEE*. P1-4.

[20] Merseguer, J,. and Campos, J. (2003). Exploring roles for the UML diagrams in software performan *Dpto. de Informatica e Ingeniera de Sistemas Universidad de Zaragoza, Zaragoza, Spain*. p1-5.

[21] Balsamo, S,. Personè, V.D.N and Inverardi, P. (2003). A review on queueing network models with finite capacity queues for software architectures performance prediction. *Elsevier Science*. p1-22.

[22] Petriu, D.B,. Amyot, D,. and Woodside, M. (2003). Scenario-Based Performance Engineering with UCMNAV. *Carleton University*. p1-16.

[23] Balsamo, S,. and Marzolla, M. (2003). Simulation Modelling of UML Software Architectures. *Università Ca' Foscari di Venezia*. p1-12.

[24] Yijun, Y,. Mylopoulos, J and Eric Y. (2003). Software refactoring guided by multiple soft-goals. *IEEE*. p1-6.

[25] Menascé, D.A,. and Bennani, M.N. (2003). ON THE USE OF PERFORMANCE MODELS TO DESIGN SELF-MANAGING COMPUTER SYSTEMS. *Menascé and Bennani*. p1-9.

[26] Compare, D,. D'Onofrio, A,. Marco, A.D and Inverardi, P. (2004). Automated Performance Validation of Software Design: An Industrial Experience. *IEEE*. p1-4.

[27] Balsamo, S,. Marzolla, M,. Marco, A.D and Inverardi, P . (2004). Experimenting different software architectures performance techniques: a case study. *Dipartimento di Informatica*. p1-5.

[28] Balsamo, S,. and Simeoni, M. (2004). Integrating Performance Modelling in the Software Development Process. *Springer*. p1-5.

[29] Balsamo, S,. Marco, A.D,. and Inverardi, P. (2004). Model-Based Performance Prediction in Software Development: A Survey. *IEEE*. 30 (5), p1-16.

[30] Khoshgoftaar, T.M . (2004). A Multiobjective Module-Order Model for Software Quality Enhancement. *IEEE*. 8 (6), p1-16.

[31] Johansson, E,. and Wartenberg, F. (2004). Proposal and Evaluation for Organising and Using Available Data for Software Performance Estimations in Embedded Platform Development. *IEEE*. p1-8.

[32] Woodside, M and Petriu, D. (2004). Capabilities of the UML Profile for Schedulability Performance and Time (SPT). *Dept of Systems and Computer Engineering, Carleton University, Ottawa Canada*. p1-4.

[33] Verdickt, T,. Dhoedt, B,. Gielen, F,. and Demeester, P. (2005). Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models. *IEEE*. 31 (8), p1-17.

[34] Brad Clark and Dave Zubrow. (2001). How Good is the Software: A Review of Defect Prediction Techniques . *Carnegie Mellon University*. 1 (2), p1-35.

[35] Liu, Y. (2005). Design-Level Performance Prediction of Component-Based Applications. *IEEE*. 31 (11), p1-14.

[36] Briand, L.C. (2005). An Experimental Investigation of Formality in UML-Based Development. *IEEE*. 31 (10), p1-17.

[37] Omari, T,. Franks, G,. Woodside, M,. and Pan, A. (2005). Efficient Performance Models for Layered Server Systems with Replicated Servers and Parallel Behaviour. *Elsevier Science*. p-1-33.

[38] Feitelson, D.G. (2005). Experimental Analysis of the Root Causes of Performance Evaluation Results: A Backfilling Case Study. *IEEE*. 16 (2), p1-5.

[39] Smith, C.U,. and Lladó, C.M. (2005). Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation. *Smith and Lladó*. p1-10.

[40] Omari, T,. Franks, G,. Woodside, M,. and Pan, A. (2005). Solving Layered Queueing Networks of Large Client-Server Systems with Symmetric Replication. *ACM*. p1-8.

[41] Mart´ınez, E.G and Merseguer, J. (2005). A Software Performance Engineering Tool based on the UML-SPT. *Dpto. de Inform´atica e Ingenier´ıa de Sistemas. Universidad de Zaragoza, Spain.*. p1-2.

[42] Distefano, S,. Scarpa, M,. and Puliafito, A. (2005). Software Performance Analysis in UML Models. *IEEE*. p1-11.

[43] Xu, J and Woodside, M. (2005). Template Driven Performance Modelling of Enterprise Java Beans. *Carleton University*. p1-8.

[44] Smith, C.U,. and Lladó, C.M. (2005). From UML models to software performance results: An SPE process based on XML interchange formats. *Workshop on Software and Performance*. p1-12.

[45] Ganesan, S and Prevostini, M. (2006). Bridging the Gap between SysML and Design Space Exploration. *University of Lugano*. p1-6.

[46] Sharma, V.S,. Jalote, P and Trivedi, K.S. (2006). A Performance Engineering Tool for Tiered Software Systems. *IEEE*. p1-8.

[47] Sharma, V.S,. Jalote, P and Trivedi, K.S. (2006). A Performance Engineering Tool for Tiered Software Systems. *IEEE*. p1-8.

[48] Gao, F,. and Sair, S. (2006). Long-term Performance Bottleneck Analysis and Prediction. *IEEE*. p1-7.

[49] Oktay, M,. Gülbağcı,A.B,. and Sarıöz, M. (2007). Architectural, Technological and Performance Issues in Enterprise Applications. *World Academy of Science, Engineering and Technology*. p1-6.

[50] Hsu, C.C,. and Devetsikiotis, M. (2007). An Automatic Framework for Efficient Software Performance Evaluation and Optimization. *IEEE*. p1-7.

[51] Smith, C.U,. Lladó, C.M,. Puigjaner, R and Williams, L.G. (2007). Interchange Formats for Performance Models: Experimentation and Ouput¤. *Smith, C.U,. Lladó, C.M and Puigjaner,(* 1Copyright 2007 by the Authors. All rights reserved*) R*. p1-10.

[52] Woodside, M,. Franks, G and Petriu, D.C. (2007). The Future of Software Performance Engineering. *IEEE*. P1-17.

[53] Cortellessa, V,. Pierini, P and Rossi, D. (2007). Integrating Software Models and Platform Models for Performance Analysis. *IEEE*. 33 (6), p1-17.

[54] Smith, C.U. (2007). Introduction to Software Performance Engineering: Origins and Outstanding Problems. *Springer*. p1-34.

[55] Sankarasetty, J,. Mobley, K,. Foster, L,. Foster, T and Calderone, T. (2007). Software Performance in the Real World: Personal Lessons from the Performance Trauma Team. *ACM*. p1-8.

[56] Menasc´e, D.A. (2008). Computing Missing Service Demand Parameters for Performance Models. *George Mason University*. p1-7.

[57] Thakkar, D,. Hassan, A.E,. Hamann,G and Flora P. (2008). A Framework for Measurement Based Performance Modelling. *ACM*. p1-11.

[58] Bertoli, M,. Casale, G,. and Serazzi, G. (2007). The JMT Simulator for Performance Evaluation of Non-Product-Form Queueing Networks.*IEEE*. p1-8.

[59] FILIPOWICZ, B,. and KWIECIEN, J. (2008). Queueing systems and networks. Models and applications. *OF SCIENCES TECHNICAL SCIENCES*. 56 (4), p1-12.

[60] Woodside, M. (2008). Performance Data and Performance Models.*Carleton University Ottawa, Canada*. P1-70.

[61] Franks, G. (2009). Enhanced Modelling and Solution of Layered Queueing Networks. *IEEE*. 35 (2), p1-14.

[62] Me´szo¨ly, A.B,. Levendovszky, T,. and Szeghegyi, A. (2009). Improved Performance Models of Web-Based Software Systems. *IEEE*. p1-6.

[63] Xu, J. (2009). Rule-based automatic software performance diagnosis and improvement. *Elsevier*. p1-26.

[64] Pelliccione, P,. Inverardi, P and Muccini, H. (2009). CHARMY: A Framework for Designing and Verifying Architectural Specifications.*IEEE*. 35 (3), p1-35.

[65] Giachetti, G,. Marín, B,. and Pastor, O. (2009). Using UML Profiles to Interchange DSML and UML Models. *IEEE*. p1-10.

[66] Bendraou, R,. and Je´ze´ quel, J. M. (2010). A Comparison of Six UML-Based Languages for Software Process Modelling. *IEEE*. 36 (5), p1-14.

[67] Distefano, S,. Puliafito, A and Scarpa, M . (2010). Implementation of the Software Performance Engineering Development Process. *ACADEMY PUBLISHER* . 5 (8), p1-11.

[68] Mészöly, A.B and Levendovszky, T. (2010). A novel algorithm for performance prediction of web-based software systems. *Elsevier*. p45-57.

[69] Mansharamani, R and Nambiar, M. (2010). Performance Engineering of a Trading Exchange's Risk Management System. *CMG*. p1-9.

[70] DISTEFANO, S,. PULIAFITO, A,. and SCARPA, M. (2010). DESIGN AND MODELING IN THE SOFTWARE PERFORMANCE ENGINEERING DEVELOPMENT PROCESS. *World Scienti¯c Publishing Company*. 19 (1), p307-323.

[71] Happe, J,. Westermann, D,. Sachs, K and Kapov´a, L. (2010). Statistical Inference of Software Performance Models for Parametric Performance Completions. *Springer*. p20-35.

[72] Traore, I,. Woungang, I,. Ahmed, A.A.E.S,. and Obaidat, M.S. (2010). UML-Based Performance Modelling of Distributed Software Systems.*ACADEMY PUBLISHER* . p1-10.

[73] Balsamo, S,. Harrison, P.G and Marin, A. (2010). A Unifying Approach to Product-forms in Networks with Finite Capacity Constraints. *ACM*. p1-11.

[74] Mansharamani, R. (2011). Auto Analysis for IT Systems Performance Management. *Rajesh Mansharamani*. p1-8.

[75] Tawhid, R,. and Petriu, D.C. (2011). Automatic Derivation of a Product Performance Model from a Software Product Line Model. *IEEE*. p1-10.

[76] Egyed, A. (2011). Automatically Detecting and Tracking Inconsistencies in Software Design Models. *IEEE*. 37 (2), p1-17.

[77] Trubiani, C and Koziolek, A. (2011). Detection and Solution of Software Performance Antipatterns in Palladio Architectural Models. *ACM*. p1-9.

[78] Mohan Reddy, R,CH,. Geetha, E, D and Srinivasa, KG. (2011). EARLY PERFORMANCE PREDICTION OF WEB SERVICES. *IJWSC*. 2 (3), p1-11.

[79] Song, Q,. Jia, Z,. Shepperd, M,. Ying, S and Liu, J . (2011). A General Software Defect-Proneness Prediction Framework. *IEEE*. 37 (3), p1-15.

[80] Balsamo, S,. and Marin, A. (2011). Performance Engineering with Product-form Models: Efficient Solutions and Applications. *ACM*. p1-12.

[81] Mansharamani, R. (2011). Performance Requirements Gathering & Analysis. *Rajesh Mansharamani*. p1-24.

[82] Mansharamani, R. (2011). Performance Targets for Developers. *Rajesh Mansharamani*. p1-14.

[83] Mansharamani, R. (2011). High Level Requirements for Capacity Management. *Rajesh Mansharamani*. p1-5.

[84] Abdullatif, A.AL,. and Pooley, R.J. (2010). UML-JMT: A Tool for Evaluating Performance Requirements. *IEEE*. p4-5.

[85] Distefano, S. (2011). From UML to Petri Nets: The PCM-Based Methodology. *IEEE*. 37 (1), p1-5.

[86] DR Naamad, A. (2012). NEW CHALLENGES IN PERFORMANCE ENGINEERING. *EMC Corporation*. p1-24.

[87] Gooijer, T.D,. Jansen, A,. Koziolek, H,. and Koziolek, A. (2012). An Industrial Case Study of Performance and Cost Design Space Exploration. *ACM*. p4-5.

[88] Arcelli, D. (2012). Model-based software refactoring driven by performance analysis. *Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica Universit`a degli Studi di L'Aquila*. p1-8.

[89] Aziz, M.W,. Mohamad, R and Jawawi, D.N.A. (2012). SOA4DERTS: A Service-Oriented UML profile for Distributed Embedded Real-Time Systems. *IEEE*. p4-5.

[90] Traore, I,. Woungang, I,. Ahmed, A.A.E.S,. and Obaidat, M.S. (2012). Software Performance Modelling using the UML: a Case Study.*ACADEMY PUBLISHER*. 7 (1), p1-17.

[91] Kounev, S,. Spinner, S and Meier, P. (2012). Introduction to Queueing Petri Nets: Modelling Formalism, Tool Support and Case Studies. *IEEE*. p1-8.

[92] Arcelli, D and Cortellessa, V. (2013). Software model refactoring based on performance analysis: better working on software or performance side?. *ACM*. p1-33.

[93] Abebe, M and Yoo, C.J. (2014). Classification and Summarization of Software Refactoring Researches: A Literature Review Approach. *ISSN*. 46,p1-6.

[94] Etxeberria, L,. Trubiani, C,. Cortellessa, V,. and Cortellessa G . (2014). Performance-based Selection of Software and Hardware Features under Parameter Uncertainty. *ACM*. p4-5.

[95] Arcelli, D and Cortellessa, V. (2015). Assisting Software Designers to Identify and Solve Performance Problems. ACM. p1-6.

[96] Steven H. Lett. (n.d). Using Peer Review Data to Manage Software Defects. *Software Engineering*. p1-6.

[97] Kasi Reddy, G,. SambasivaRao, B,. Kumar, S.D,. and Rani, P.B. (2013). Software Performance Evaluation of a Polar Satellite Antenna Control Embedded System. ISSN. 2 (1), p1-8.

[98] Schwetmen, H. (1982). Implementing the mean value Algorithm for the Solution of Queueing Network Models. Purdue University. p1-31.

[99] Woodside, M,. Zink, K and Petriu, D . (2005). Issues in Performance Annotations for UML. Carleton University. p1-35.

[100] Woodside, C.M. (2006). The Use of Optimal Filters to Track Parameters of Performance Models. Carleton University. p1-22.

[101] Ruhroth, T,. Voigt, H and Wehrheim, H. (2009). Measure, diagnose, refactor: A formal quality cycle for software models. IEEE. p1-8.

[102] L. Tahvildari and K. Kontogiannis. Requirementsdriven software re-engineering framework. In WCRE 2001, pages 71–80, 2001.

[103] M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In Proc. QEST'06, 119–120, IEEE Press.

[104] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A processoriented approach. IEEE Transactions on Software Engineering, 18(6):483–497, Jun 1992. Special Issue on Knowledge Representation and Reasoning in Software Engineering.

[105] Hon Hopkins. Component Premier. Communications of the ACM (2000): 27-30.

[106] Wojtek Kozaczynski and Grady Booch. Component Based Software Engineering. IEEE (1998): 34-37.