Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 14 Issue 6 Version 1.0 Year 2014
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Measurement and Prediction of Software Performance by Models

By G. Kasi Reddy & Dr. D Sravan Kumar
JNTU, India

Abstract- Software Performance Engineering (SPE) provides a systematic, quantitative approach to constructing software systems that meet performance objectives. It prescribes ways to build performance into new systems rather than try to fix them later. Performance is a pervasive quality of software systems; everything affects it, from the software itself to all underlying layers, such as operating system, middleware, hardware, communication networks, etc. Software Perfor -mance Engineering encompasses efforts to describe and improve performance, with two distinct approaches: an early-cycle predictive model-based approach, and a late-cycle measurement-based approach. Current progress and future trends within these two approaches are described, with a tendency (and a need) for them to converge, in order to cover the entire development cycle.

Keywords : *SPE, performance prediction, performance measurement, UML, debugging.*

GJCST-C Classification : *H.1*

Measurement and Prediction of Software Performance by Models

G.Kasi Reddy [α] & Dr. D Sravan Kumar [σ]

Abstract- Software Performance Engineering (SPE) provides a systematic, quantitative approach to constructing software systems that meet performance objectives. It prescribes ways to build performance into new systems rather than try to fix them later. Performance is a pervasive quality of software systems; everything affects it, from the software itself to all underlying layers, such as operating system, middleware, hardware, communication networks, etc. Software Perfor - mance Engineering encompasses efforts to describe and improve performance, with two distinct approaches: an early-cycle predictive model-based approach, and a late-cycle measurement-based approach. Current progress and future trends within these two approaches are described, with a tendency (and a need) for them to converge, in order to cover the entire development cycle.

Keywords: SPE, performance prediction, performance measurement, UML, debugging.

I. Introduction

Despite rapidly improving hardware, many recent software systems are still suffering from performance problems, such as high response times or low throughputs [1]. Hardware is often not the limiting factor as powerful multi-core and many core processors are readily available on the market and modern software systems may run in huge data centers with virtually unlimited resources. Performance problems often stem from software architectures that are not designed to exploit the available hardware. Instead, these software architectures ignore the advances of distributed computing and multi-core and many core processors.

Systematic approaches for engineering softw-are systems to achieve desired performance properties have been proposed [2, 3]. They advocate modeling software systems during early development stages, so that performance simulations can validate design decisions before investing implementation effort.

The advent of multi-core processors results in new challenges for these systematic software perfor-mance engineering (SPE) methods. Modeling software running on thousands of cores requires rethinking of existing approaches [4]. While techniques and tools for parallelizing software are evolving [5], novel methods and tools need to be created to assist software in designing systems that can exploit the capabilities for parallel execution but do not overburden software developers during implementation [6].

Author α : Research Scholar, CSE, JNTU Hyderabad, India.
e-mail : gkreddy@mgit.ac.in
Author σ : Principal and Professor of CSE, KITE women's college of Professional Engineering Sciences, Hyderabad, India.

II. Software Performance Engineering

SPE is a software-oriented approach; it focuses on architecture, design, and implementation choices. It uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements. The models assist developers in controlling resource requirements by enabling them to select architecture and design alternatives with accep-table performance characteristics. The models aid in tracking performance throughout the development process and prevent problems from surfacing late in the life cycle (typically during final testing).[7] SPE also prescribes principles and performance patterns for cre - ating responsive software, performance anti-patterns for recognizing and correcting common problems, the data required for evaluation, procedures for obtaining perfor - mance specifications, and guidelines for the types of evaluation to be conducted at each development stage. It incorporates models for representing and predicting performance as well as a set of analysis methods.[8]

III. Progress in Measurement, Debugging and Testing

Measurement is used by verification teams to ensure that the system under test meets its specifications, by performance modelers to build and validate models, and by designers to find and fix hot-spots in the code. Interest in the measurement of the performance of a computer system ranges back to the development of the very first systems, described in an early survey paper by Lucas [9]. Today, the state of industrial performance measurement and testing techn - iques is captured in a series of articles by Scott Barber [7] including the problems of planning, execution, instrumentation and interpretation. For performance test design, an important issue is to determine the workload under which the testing is done. An approach is to run the performance tests under similar conditions with the expected operational profile of the application in the field [9]. Briand and co-workers have pioneered the use of models to create stress tests for time-critical systems, by triggering stimuli at strategic instants [10].

Performance models are often difficult to construct, even with a live system, despite the presence of tools to actually measure performance. In the future, model building will become much more automated, and output becomes standardized, and the conversion process between measurement information and performance model becomes more practical. Ultimately, the model and measurement information will be fed back into design tools, so that performance issues are brought to the forefront early in the design process.

a) Performance Measurement- Best practices

These are practices for those responsible for measuring software performance and for performance testing. [11]

i. *Plan Measurement Experiments to Ensure That Results Are Both Representative And Reproducible*

There are two key considerations in planning performance measurements: They must be repre - sentative and reproducible. To be representative, meas - urement results must accurately reflect each of the factors that affect performance: workload, software, and computer system environment. The goal is to design your measurement experiment in a way that balances the effort required to construct and execute the measurement experiment against the level of detail in the resultant data. When unimportant details are omitted, both the design effort and the overhead required to collect the data are reduced.

Reproducibility gives you confidence in the results. In order for a measurement to be reproducible, the workload, software, and computer system environment must be controlled so that you can repeat the measurement and get the same (or very similar) results each time.

ii. *Instrument Software to Facilitate SPE Data Collection*

You instrument software by inserting code (probes) at key points to measure pertinent execution characteristics. For example, you might insert code that records the time at the start and end of a business task to measure the end-to-end time for that task. There are at least three reasons for supplementing the standard tools with instrumentation: convenience, data gran - ularity, and control.

iii. *Measure Critical Components Early and Often to Validate Models and Verify Their Predictions*

Measurements substantiate model predictions, and confirm that key performance factors have not been omitted from the models. Occasionally, software exec - ution characteristics are omitted from a model because their effects are thought to be negligible. Later, you may discover that they in fact have a significant impact on performance, as illustrated in the following anecdote: An early life cycle model specified a transaction with five database “Selects.” During detailed design, “Order by” clauses were added to three of the “Selects.” The developers viewed the additional clause as “insign-ificant” because only one to five records would be sorted for each “Select.” Upon investigation, though, the performance analyst discovered that over 50,000 instructions were executed for each sort!

The way to detect these omissions is to measure critical components as early as possible and continue measuring them, to ensure that changes do not invalidate the models.

IV. Prediction of Performance by Models

The special capability of a model is prediction of properties of a system before it is built, or the effect of a change before it is carried out. This gives a special “early warning” role to early-cycle modeling during requirements analysis. However as implementation proceeds, better models can be created by other means, and may have additional uses, in particular

- design of performance tests
- configuration of products for delivery
- evaluation of planned evolutions of the design, recognizing that no system is ever final.

a) Performance models from scenarios

Early performance models are usually created from the intended behaviour of the system, expressed as scenarios which are realizations of Use Cases. The term “scenario” here denotes a complex behavior including alternative paths as well as parallel paths and repetition. The performance model is created by extra - cting the demands for resource services. Annotated UML specifications are a promising development.

The annotations include:

- the workload for each scenario, given by an arrival rate or by a population with a think time between requests,
- the CPU demand of steps,
- the probabilities of alternative paths, and loop counts,
- the association of resources to the steps either impl -icitly (by the processes and processors) or explicitly.

As an illustration, Figure 1 shows a set of applications requesting service from a pool of server threads running on a multiprocessor (deployment not shown). Part (a) shows the scenario modeled as a UML sequence diagram with SPT annotations, (b) shows a graph representing the scenario steps, and (c) shows the corresponding layered queueing network (LQN) model. Studies in [12] [13] use such models.

At a later stage, scenarios may be traced from execution of prototypes or full deployments, giving accurate behaviour. Models can be rebuilt based on these experimental scenarios [14], combined with measured values of CPU demands.

b) Performance models from objects and components

A performance model can be built based on the software objects viewed from a performance perspective. A pioneering contribution in this direction defined a "performance abstract data type" for an object [13], based on the machine cycles executed by its methods.

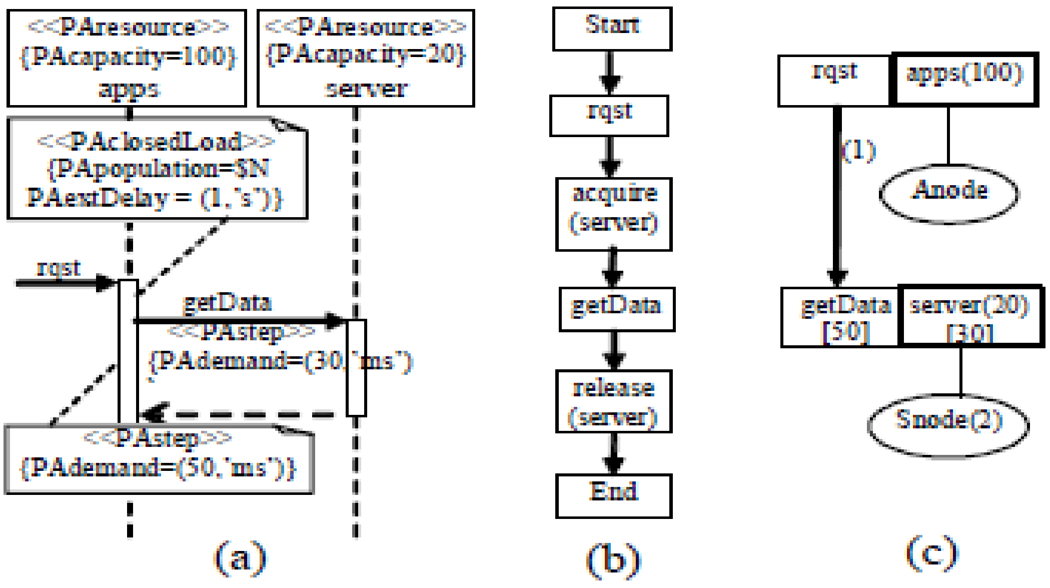

Figure 1 : Annotated UML, Scenario Model, and Performance Model

To create a performance model, one traces a response from initiation at a root object to all the interfaces it calls, proceeding recursively for each call. Queueing and layered queueing models were derived based on objects and calls in [14] and [15]. Model parameters (CPU, call frequencies) were estimated by measurement or were based on the documentation plus expertise. Object-based modeling is inherently compositional, based on the call frequencies between objects. This extends to subsystems composed of objects, with calls between subsystems. In [2] an existing application is described in terms of UNIX calls, and its migration to a new platform is evaluated by a synthetic benchmark with these calls, on the new platform. This study created a kind of object model, but then carried out composition and evaluation in the measurement domain. The convergence of models and measurements is an important direction for SPE.

The object-based approach to performance modeling can be extended to systems built with reusable components. Composition of sub-models for Component-Based Software Engineering [16] was described in [17]. Issues regarding performance contracts between components are discussed in [18]. Components or platform layers can be modeled separately, and composed by specifying the calls between them. For example, in [18] a model of a J2EE application server is created as a component that offers a large set of operations; then an application is modeled (by a scenario analysis) in terms of the number of calls it made to each operation.

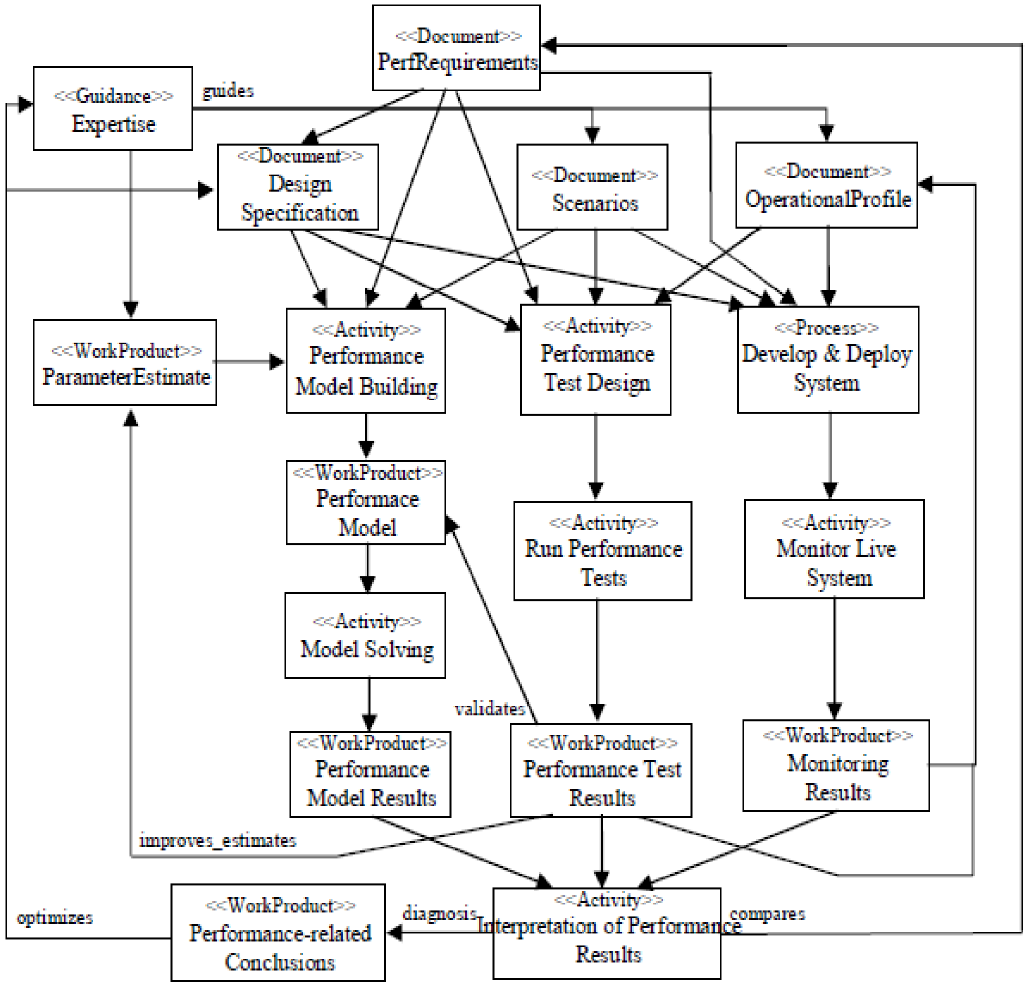
<<Document>> PerfRequirements
<<Guidance>> Expertise
guides
<<Document>> Design Specification
<<Document>> Scenarios
<<Document>> OperationalProfile
<<WorkProduct>> ParameterEstimate
<<Activity>> Performance Model Building
<<Activity>> Performance Test Design
<<Process>> Develop & Deploy System
<<WorkProduct>> Performace Model
<<Activity>> Run Performance Tests
<<Activity>> Monitor Live System
<<Activity>> Model Solving
validates
<<WorkProduct>> Performance Model Results
<<WorkProduct>> Performance Test Results
<<WorkProduct>> Monitoring Results
improves_estimates
optimizes
<<WorkProduct>> Performance-related Conclusions
diagnosis
<<Activity>> Interpretation of Performance Results
compares

Figure 2 : Simplified domain model for a converged SPE process

The quantitative parameters of the performance model for the J2EE server - and the underlying operating system and hardware platform -were obtained by measurements for two different implementations. The main challenge regarding performance characterization of reusable components stem from the fact that the offered performance depends not only on the component per se, but also on its context, deployment, usage and load. It seems obvious that such approaches apply similarly to Generative techniques [17] and to Model-Driven Development. The *completion* of perfo - rmance models made from a software design, by adding components that make up its environment but are outside the design, is also largely based on composition of sub-models [19]. This is an aspect of Model-Driven Development.

V. Convergence of the Measurement and Modeling Approaches

The present state of performance engineering is not very satisfactory, and better methods would be welcome to all. One way forward is to combine knowledge of different kinds and from different sources into a converged process. Figure 2 outlines such a process, with the main concepts and their relationships. The notation is based on the newly adopted OMG standard Software Process Engineering Meta model (SPEM) [20]. At the core of SPEM is the idea that a software process is a collaboration between abstract active entities called *ProcessRoles* (e.g., usecase act-ors) that perform operations called *Activities* on conc - rete entities called *WorkProducts.* Documents, models, and data are examples of WorkProduct specializations. Guidance elements may be associated to different model elements to provide extra information.

Figure 2 uses stereotypes defined in [20]. Concepts related to the model-based approach appear on the left of Figure 2, and to the measurement-based approach on the right. A distinction is made between performance testing measurements (which may take place in a laboratory setting, with more sophisticated measurement tools and special code instrumentation) and measurements for monitoring live production systems that are deployed on the intended target system and used by the intended customers. The domain model from Figure 3 is very generic. For instance, there is no indication whether different activities (such as Performance Model Building) are done automatically through model transformations or "by hand" by a performance analyst.

In a convergence of data-centric and model-centric methods, data (including prior estimates) provides the facts and models provide structure to organize and to extract significance from the facts. Our exploration of the future will examine aspects of this convergence. Models have a key role. They integrate data and convert it from a set of snapshots into a process capable of extrapolation. To achieve this potential we must develop robust and usable means to go from data to model (i.e., model-building) and from model to "data" (solving to obtain predictions). We must also learn how to combine measurement data interpretation with model interpretation, and to get the most out of both. Capabilities supported by convergence include:

- efficient testing, through model-assisted test design and evaluation
- search for performance-related bugs,
- performance optimization of the design
- scalability analysis
- reduced performance risk when adding new features,
- aids to marketing and deployment of products.

The future developments that will provide these capabilities are addressed in the remainder of this section. A future tool suite is sketched in Figure 3.

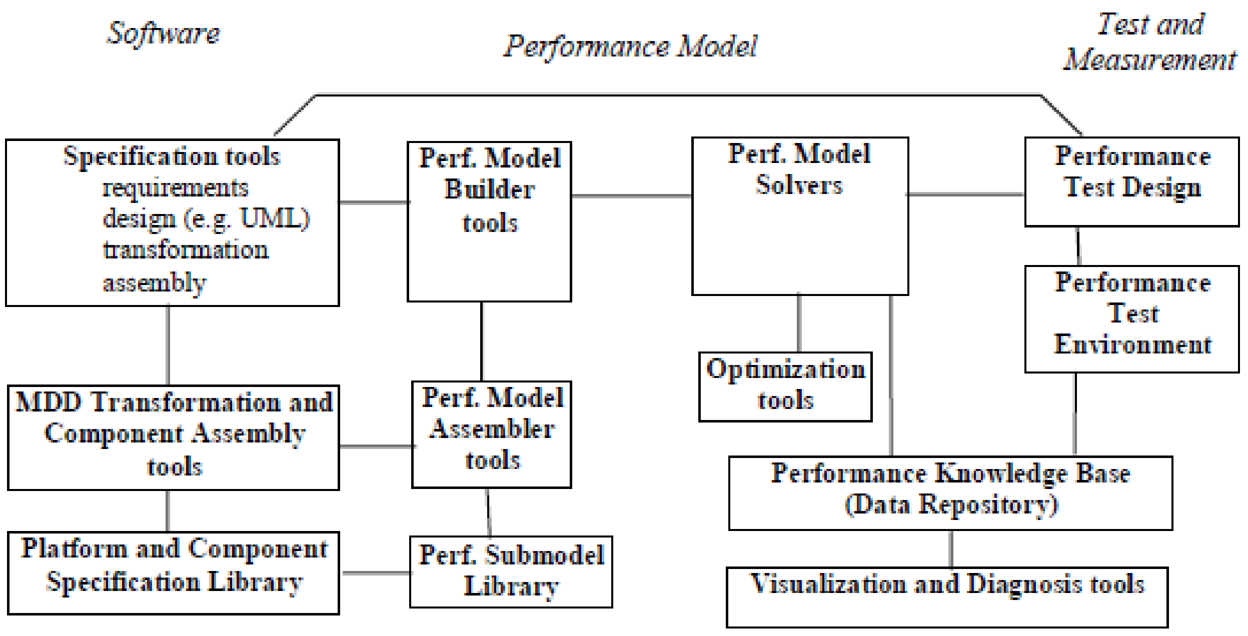
Software
Performance Model
Test and Measurement
Specification tools requirements design (e.g. UML) transformation assembly
Perf. Model Builder tools
Perf. Model Solvers
Performance Test Design
Performance Test Environment
Optimization tools
MDD Transformation and Component Assembly tools
Perf. Model Assembler tools
Performance Knowledge Base (Data Repository)
Platform and Component Specification Library
Perf. Submodel Library
Visualization and Diagnosis tools

Figure 3 : Tools for a Future Converged SPE Process, linked to Software Development Tools

VI. Efficient Model-Building Tools

The abstractions provided by performance models are valuable, but some way must be found to create the models more easily and more quickly. For performance models made early in the lifecycle from specified scenarios, automated model-building has be - en demonstrated [6] and is supported by the UML profiles [21]. The future challenge is to handle every scenario that a software engineer may need to describe, and every way that the engineer can express them (including the implied scenario behaviour of object call hierarchies, and the composition of models from component designs).

The multiplicity of model formats hinders tool development, and would be aided by standards for performance model representations, perhaps building on [22]. Interoperability of performance building tools with standard UML tools is also helpful. For instance, the PUMA architecture[23] shown in Figure 6 supports the generation of different kinds of performance models (queueing networks, layered queueing networks, Petri nets, etc.) from different versions of UML (e.g., 1.4 and 2.0) and different behavioural representations (sequ - ence and activity diagrams). PUMA also provides a feedback path for design analysis and optimization. Mid and late-cycle performance models should be extracted from prototypes and implementations. Trace based automated modeling has been described in [23], including calibrated CPU demands for operations. Fut - ure research can enhance this with use of additional instrumentation (e.g. CPU demands, code context), efficient processing, and perhaps exploit different levels of abstraction. Abstraction from traces exploits certain patterns in the trace, and domain-based assumptions; these can be extended in future research.

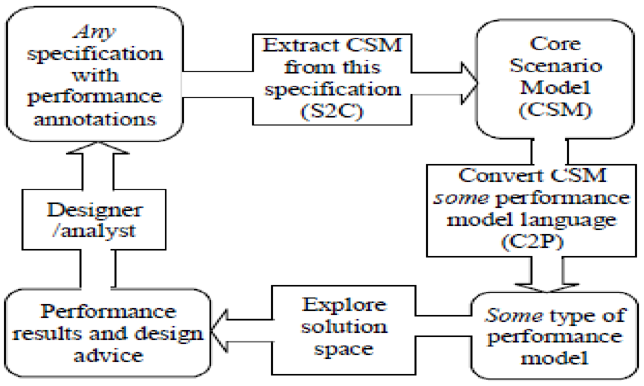
Any specification with performance annotations
Extract CSM from this specification (S2C)
Core Scenario Model (CSM)
Convert CSM some performance model language (C2P)
Designer /analyst
Performance results and design advice
Explore solution space
Some type of performance model

Figure 4 : Architecture of the PUMA toolset [23]

VII. Conclusion

Software Performance Engineering needs further development in order to cope with market requirements and with changes in software technology. It needs strengthening in prediction, testing and measurement technology, and in higher-level techni - ques for reasoning and for optimization.

References Références Referencias

1. M.R. Moghal, N. Hussain, M.S. Mirza, M.W. Mirza, M.S. Choudry, "Performance Evaluation and Modeling of Web Server Systems",WSEAS *Transa - ctions on Information Science and Applications* 1(1), July 2004, pp. 658-663.
2. C. U. Smith and L. G. Williams, Performance Solutions: A Practical Guide to Creating Res - ponsive, Scalable Software. Addison-Wesley, 2002.
3. M. Woodside, G. Franks, and D. C. Petriu, \The Future of Software Performance Engineering," in Proc. Future of Software Engineering (FOSE'07). IEEE Computer Society, 2007, pp. 171-187.
4. W. Hwu, S. Ryoo, S.-Z. Ueng, J. Kelm, I. Gelado, S. Stone, R. Kidd, S. Baghsorkhi, A. Mahesri, S. Tsao, N. Navarro, S. Lumetta, M. Frank, and S. Patel, \Implicitly parallel programming models for thousand-core microprocessors," in Proc. 44th ACM/IEEE Design Automation Conference (DAC '07), june 2007, pp. 754-759.
5. H. Vandierendonck and T. Mens, \Techniques and tools for parallelizing software," IEEE Softw., vol. 29, no. 2, pp. 22-25, 2012.
6. C. A. Schaefer, V. Pankratius, and W. F. Tichy, \Engineering parallel applications with tunable architectures," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Volume 1, ser. ICSE '10. ACM, 2010, pp. 405-414.
7. S. Barber, "Creating Effective Load Models for Performance Testing with Incomplete Empirical Data", in *Proc. 6 th IEEE Int. Workshop on Web Site Evolution,* 2004, pp. 51-59.
8. Bertolino, R. Mirandola. "Software performance engineering of component-based systems", *Proc. Workshop on Software and Performance,* WO - SP'2004, pp 238-242.
9. H. Lucas Jr, "Performance evaluation and monit - oring", *ACM Computing Surveys,* 3(3)*,* Sept. 1971, pp 79-91.
10. V. Garousi, L. Briand, Y. Labiche, "Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models", *Proc. Int. Conference on Software Engineering,* Shanghai, China, 2006, pp. 391-400.
11. V. Cortellessa, A. Di Marco, P. Inverardi, "Nonfun - ctional Modeling and Validation in Model-Driven Architecture", *Proc 6 th Working IEEE/IFIP Confer - ence on Software Architecture* (WICSA 2007), Mumbai, India, 2007.
12. C. Jittawiriyanukoon, Performance Evaluation of Parallel Processing Systems Using Queueing Network Model *WSEAS Transactions on Computers* 3(5), March 2006, pp. 612-620.
13. D.B. Petriu, C.M. Woodside, "Analysing Software Requirements Specifications for Performance", *Proc. 3rd Int. Workshop on Software and Perform - ance,* Rome, 2002.
14. M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov, "Automated Performance Modeling of Software Generated by a Design Environment", *Performance Evaluation*, vol. 45, no. 2-3 pp. 107-124, 2001.
15. J. Xu, M. Woodside, and D.C. Petriu, "Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time", in Proc. *13 th Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation*, Urbana, USA, Sept. 2003
16. C. Szypersky, D. Gruntz, S. Murer, *Component Software: Beyond Object Oriented Programming*, Addison- Wesley, 2002.

17. X. Wu and M. Woodside, “Performance Modeling from Software Components”, in *Proc.WOSP’2004,* Redwood Shores, Calif., 2004, pp. 290-301.
18. R. H. Reussner, V. Firus, S. Becker, “Parametric Performance Contracts for Software Components and their Compositionality”, in *9th Int. Workshop on Component- Oriented Programming,* Oslo, June 2004.
19. M. Woodside, D.B. Petriu, K. Siddiqui, “Performan - cerelated Completions for Software Specifications”, *Proc 24th Int. Conf. on Software Engineering*, Orlando, 2002.
20. Object Management Group, *Software Process Engineering Metamodel Specification*, formal/05-01-06, 2006.
21. Object Management Group, *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP,* OMG doc. realtime/05-02-06, 2005.
22. C.U. Smith, C. M. Lladó, V. Cortellessa, A. diMarco, L. Williams, “From UML models to software performance results: an SPE process based on XML interchange formats”, in *Proc WOSP’2005,* Palma de Mallorca, 2005, pp. 87-98.
23. M. Woodside, D.C. Petriu, D.B. Petriu, H. Shen, T. Israr, J. Merseguer, “Performance by Unified Model Analysis (PUMA)”, *Proc. WOSP’2005,* Mallorca, pp 1-12.
24. Erika Corona, Filippo Eros Pani, “A Review of Lean-Kanban Approaches in the Software Development”, *WSEAS Transactions on Information Science and Applications,* Vol. 10, Issue 1, pp. 1-13, January 2013.
25. Manoj Kumar Tyagi, Srinivasan M., L. S. S. Reddy, “Design of Traditional/Hybrid Software Project Tracking Technique: State Space Approach”, *WSEAS Transactions on Information Science and Applications,* Vol. 11, Issue 11, pp. 345-355, November 2013.

This page is intentionally left blank