SIMULATION MODELING OF UML SOFTWARE ARCHITECTURES*

SIMONETTA BALSAMO MORENO MARZOLLA

Dipartimento di Informatica
Università Ca' Foscari di Venezia
via Torino 155, 30172 Mestre (VE), Italy
e-mail: {balsamo|marzolla}@dsi.unive.it

Abstract Quantitative analysis of software systems is being recognized as an important issue in the software development process. Performance analysis can help to address quantitative system analysis from the early stages of the software development life cycle, e.g., to compare design alternatives or to identify system bottlenecks. Modeling software systems by simulation allows the analyst to represent detailed characteristics of the system. We consider simulation for performance evaluation of software architectures specified by UML. We derive a simulation model for annotated UML software architectures. First we propose the annotation for some UML diagrams to describe performance parameters. Then we derive the simulation model by automatically extracting information about Use Case and Activity Diagrams from the XMI descriptions of UML diagrams. This information is used to build a discrete-event simulation model, which is finally executed. Simulation results are inserted back into the original UML diagrams as tagged values to provide feedback at the software architectural design level.

Keywords: Process-Oriented Simulation, Software Systems, Unified Modeling Language, Performance Evaluation.

1 INTRODUCTION

In recent years it has been recognized that the software development processes should be supported by a suitable mechanism for early assessment of software performance. Early identification of unsatisfactory performance of Software Architecture (SA) can greatly reduce the cost of design change [Smith, 1990; Smith and Williams, 2002]. The reason is that correcting a design flaw is more expensive the later the change is applied during the software development process. This is particularly true if the waterfall software development model is employed, as any change during the development process requires to start back from the beginning. However, this is still a relevant issue whenever a different development process is used.

Both quantitative and qualitative analysis can be performed at the software architectural design level. Qualitative analysis deals with functional properties of the software system such as for example deadlock-freedom or security. Qualitative analysis is carried out by measurement or by modeling the software system to derive quantitative figures of merit, such as, for example, the execution profile of the software, memory or network utilization. We focus on performance models of software systems at the SA level.

In this paper we consider quantitative evaluation of the performance of SA at the design level by means of simulation models. We consider SA expressed in terms of Unified Modeling Language (UML) [Object Management Group, 2001] diagrams. We propose to annotate the UML diagrams using a subset of annotations defined in the UML Profile for Schedulability, Performance and Time Specification [Object Management Group, 2002a] (referred as *UML performance profile*).

Simulation is a powerful modeling technique that allows general system models; simulation models can represent arbitrarily complex real-world situations, which can be too complex or even impossible to represent by analytical models. We define a simulation model of an UML software specification introducing an almost one-to-one correspondence between behaviors expressed in the UML model and entities or processes in the simulation model. This correspondence between system and the model helps the feedback process to report simulation results back into the original SA.

There are a few previous works dealing with simulation of UML specifications. Arief and Speirs [Arief and Speirs, 1999a,b, 2000] developed an automatic tool for deriving simulation models from UML Class and Sequence diagrams. Their approach consists in transforming the UML diagrams into a simulation model described as an XML document. This model can then be translated into different kinds of simulation programs, even written in different languages. In this way the performance model is decoupled from its actual implementation. De Miguel et al. [De Miguel et al., 2000] introduced UML extensions for the representation and automatic evaluation of temporal requirements and resource usage, particularly targeted at real-time systems. The extensions are expressed in term of stereotypes, tagged values and stereotyped constraints. These were intro-

*This work has been partially supported by MURST Research Project "Sahara" and by MIUR Research Project "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools".

duced in a commercial UML CASE tool, which has been made able to generate OPNET simulation models starting from annotated UML diagrams.

Previous simulation-based performance modeling approaches for evaluation of UML SA were developed before the UML performance profile was defined. Thus, they introduced their special extensions of UML or introduced non standard annotations to express quantitative information useful for deriving the model. In this paper we describe UML Performance Simulator (UML-PSI), a performance evaluation tool which translates UML Use Case and Activity diagrams into a discrete-event process oriented simulation model. The UML diagrams are annotated according to a subset of the UML Performance Profile. This additional information is used to define the simulation model, which is finally executed. Simulation results are inserted back into the original UML diagrams as tagged values to provide feedback at the software architectural design level.

This paper is organized as follows. In Section 2 we illustrate the proposed methodology for generating simulation models from UML SA. In Section 3 we describe UML-PSI, a tool we built to implement that methodology. In Section 4 we illustrate a simple case study, and conclusions and future works are discussed in Section 5.

2 METHODOLOGY

In order to assist the software developer during the design process, we illustrate in Fig. 1 a general framework for quantitative analysis of UML SA [Balsamo et al., 2002]. The starting point is a description of the SA. We consider a description as a set of UML diagrams annotated with quantitative information in order to derive a simulation-based performance model. The model is obtained using a suitable Modeling Algorithm. The model is then implemented in a simulation program, which is eventually executed. Simulation results are a set of performance measures that can be used to provide a feedback at the original SA design level. The feedback should pinpoint performance problems on the SA, and possibly provide suggestions to the software designer about how the problem can be solved. The modeling cycle can be iterated until a SA with satisfactory performances is developed.

Note that the modeling and performance evaluation framework of Fig. 1 is independent from the particular performance model that we apply. In this paper we consider simulation-based performance models of UML SA. We describe UML-PSI, a prototype performance evaluation tool which processes an XMI [Object Management Group, 2002b] description of UML Use Case and Activity diagrams. The UML SA has to be annotated using a simplified subset of the UML Profile for Schedulability, Performance and Time Specification [Object Management Group, 2002a]. The simulation model is process oriented and its objects are

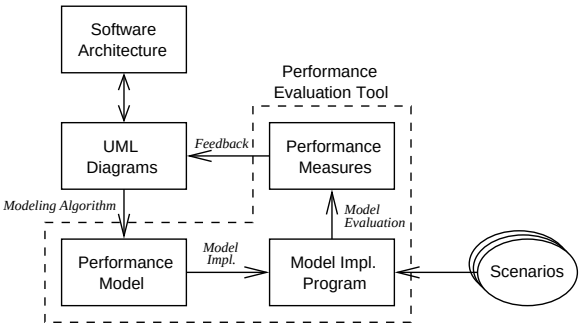


Figure 1: Framework for quantitative evaluation of UML diagrams

derived by the analysis of the UML diagrams annotated with performance specification of the software system components. The simulation model is implemented as a discrete-event simulation program written in C++, whose execution provides results for a set of performance indices. We evaluate through simulation the mean response time associated with the execution of each scenario (Use Case) and each scenario step (Activity). Simulation results, i.e., the performance measures of the software components are inserted back into the original UML SA as tagged values to provide feedback to the system designer.

Figure 2 illustrates the structure of the performance simulation model derived from the UML diagrams. The basic object of the simulation model is a `PerformanceContext`. This object contains the other elements of the model, namely Workloads and Scenarios. Workloads can be open or closed, depending on whether the number of users accessing the system is unbounded or fixed. Open workloads are characterized by the following attribute (the exact notation used to describe the attributes is given in the next section):

occurrencePattern (of type RTarrivalPattern, defined in Section 3) the pattern of interarrival times of consecutive requests

Closed workloads are characterized by two attributes:

population the total number of users in the workload

externalDelay (of type PAPerfValue, defined in Section 3) the delay between the end of a scenario execution and the beginning of the next request

Each workload actually drives one or more scenarios. Each time a new workload user requests service to the system, one of the scenarios associated with that workload is selected. Selection is done randomly, according to the probability associated to each scenario.

A scenario is a set of abstract scenario steps, represented by the `AbsStep` class. All kinds of scenario steps are characterized by the following attributes:

probability the probability to execute this step, in the case the predecessor step has multiple successors

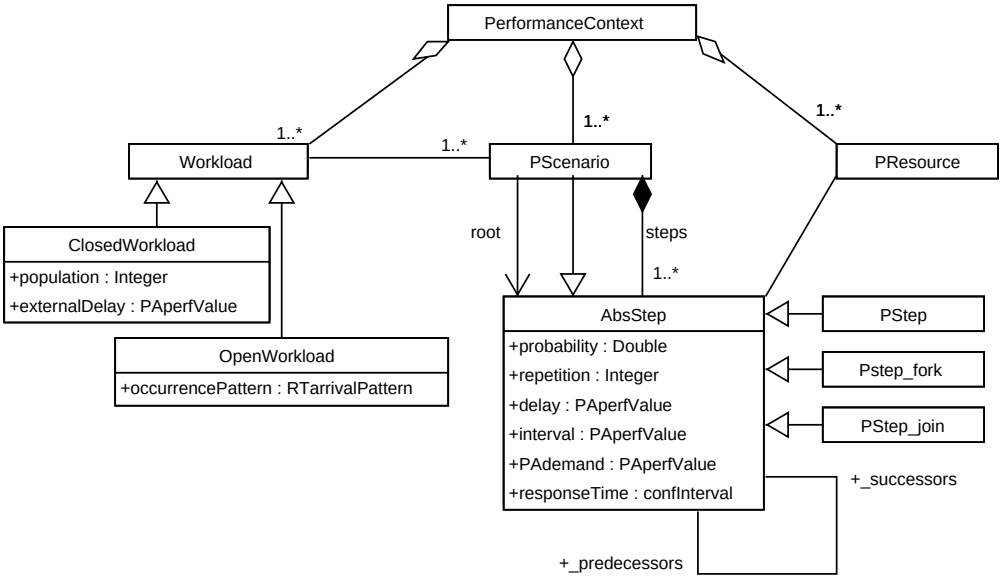


Figure 2: Structure of the simulation performance model

repetition the number of times this step has to be repeated

delay (of type PAPerfValue) an additional delay in the execution of this step, for example to model a user interaction

interval (of type PAPerfValue) the time between repetitions of this step, if it has to be repeated multiple times

PAdemand (of type PAPerfValue) the processing demand of this step

responseTime the computed delay between the starting and finishing time of this step. The estimation of this quantity is the result of the simulation model execution. This value has type confInterval, which we define as the pair of the confidence interval bounds.

Abstract scenario steps can either be composite steps (described by the `PScenario` class), or atomic steps of different kinds. Scenarios are collections of steps; exactly one of these steps is marked as the root step (starting step) of the scenario. Atomic steps can be of type `PStep_fork` for nodes representing the creation of multiple execution threads, `PStep_join` for nodes representing synchronization points between different threads, and `PStep` for normal atomic steps.

There are two main differences between the performance model depicted in Figure 2 and the one described in the UML Performance Profile. First, we assume a very simple model for resources: each Use Case has an associated computational resource; the resource is acquired when the Use Case starts, and is released at its completion. Thus, two instances of the same Use Case cannot be executed in parallel. This simplification is motivated by the fact that we evaluate SA at the architectural level, without assuming any implementation on a specific platform. Indeed we assume that the software architect ignores the specific hardware platform on which the software will be executed. Detailed resource modeling is usually done in later stages of the software development process.

Second, the structure we propose for the class hierarchy describing the processing steps is slightly different from that in [Object Management Group, 2002a]. The UML Profile defines a `PScenario` class from which a `PStep` class is derived, thus every step is a scenario. This is because each step, at a deeper level of detail, could be modeled as a whole scenario, that is, a sequence of multiple sub-steps. We choose a different structure to model the `PStep` and `PScenario` hierarchy to keep atomic steps and scenarios as separate entities. We apply the Composite Pattern [Gamma et al., 1995] to reflect the hierarchical nature of the processing steps. This choice makes the construction of the simulation model easier, because there are different kinds of step (e.g., `PStep`, `PStep_fork`, `PStep_join`) which are modeled as different simulation object types. The behavior of a fork step consists of activating all the successor steps concurrently. A join step waits for the completion of all predecessor steps before activating the successor. A normal step simulates execution according to the specified delays, and activates one of the successor steps. Finally, the behavior of a scenario is to activate its root step.

To summarize, the proposed approach to derive the simulation model to evaluate SA performance from UML Use Case and Activity diagrams is defined as follows:

1. Consider an UML representation of a software

system in terms of Use Case and Activity diagrams. Both diagrams are respectively annotated as follows:

- UML Use Case diagrams describe the interaction between the software system and one or more Actors requiring service. As proposed in [Cortellessa and Mirandola, 2002; Pooley and King, 1999] we identify Actors to represent workloads applied to the system and Use Cases to represent scenarios. Actors can be stereotyped as $\ll$ PAopenLoad $\gg$ or $\ll$ PAclosedLoad $\gg$ to represent respectively open and closed population of users accessing the system. Use Cases are tagged with `PAprob` tags, whose value indicates the probability of executing that scenario.
- Each Activity of an Activity diagram can be tagged with the following informations: the number of times the step has to be repeated (`PArep`); the delay between repetitions (`PAinterval`) of the same step; an additional delay for each step representing user "think time" (`PAdelay`); the service demand of the step (`PAdemand`).

2. The simulation model is automatically derived from the XMI description of the UML diagrams. Currently UML-PSI uses the XMI dialect of the open-source ArgoUML CASE tool [ArgoUML, 2003], The simulation model is an instance of the general class structure depicted in Figure 2, and includes a `PerformanceContext` object and a set of workloads and scenarios.

3. The simulation model is executed, optionally asking the user to specify some parameters for the simulation, such as the desired confidence level for the estimation of the performance indices, the confidence interval width and the simulation length.

4. Simulation results are inserted back into the UML model as tagged values associated with Activities and Use Cases. We consider as simulation results the average delays (`PArespTime`) of Activities and Use Cases execution.

3 UML-PSI TOOL DESCRIPTION

The steps of the proposed methodology are implemented into a simulation tool called UML-PSI. As concerns the first step of the annotation of UML diagrams, the UML Performance Profile suggests the use of TVL (Tag Value Language) to describe values of tags applied to model elements, which is a subset of the Perl language [Wall et al., 2000]. We use a freely available Perl interpreter library [CPAN, 2003] to evaluate tag values, so taking advantage of the full Perl language. Note that both the UML Performance Profile and UML-PSI do not strictly depend on the specific language used to express annotations.

The `PAperfValue` and `RTarrivalPattern` data types are expressed according to the following BNF notation, which is a simplified version of the annotations defined in the UML Performance Profile:

$$
\begin{aligned}
< \text{PAperfValue} > &:= \ '['\mathbf{assm}|\mathbf{pred}|\mathbf{msrd}, \\
&\quad \mathbf{dist}, < \text{PDFstring} >']' \\
< \text{PDFstring} > &:= \ '['< \text{constantPDF} > | \\
&\quad < \text{uniformPDF} > | \\
&\quad < \text{exponentialPDF} > | \\
&\quad < \text{normalPDF} >] \\
< \text{constantPDF} > &:= \ < \text{real} > \\
< \text{uniformPDF} > &:= \ \mathbf{uniform}, < \text{real} >, < \text{real} > \\
< \text{exponentialPDF} > &:= \ \mathbf{exponential}, < \text{real} >, < \text{real} > \\
< \text{normalPDF} > &:= \ \mathbf{normal}, < \text{real} >, < \text{real} > \\
< \text{RTarrivalPattern} > &:= \ '['< \text{bounded} > | \\
&\quad < \text{unbounded} > | \\
&\quad < \text{bursty} >']' \\
< \text{bounded} > &:= \ \mathbf{bounded}, < \text{int} >, < \text{int} > \\
< \text{bursty} > &:= \ \mathbf{bursty}, < \text{PDFstring} >, < \text{int} > \\
< \text{unbounded} > &:= \ \mathbf{unbounded}, < \text{PDFstring} >
\end{aligned}
$$

UML-PSI parses an XMI description of UML diagrams, annotated as described above. The UML model is translated into a C++ discrete-event simulation program according to step 2 of the proposed methodology. UML-PSI includes a general purpose discrete-event simulation library providing roughly the same functionality of the `Simulation` class of the SIMULA language [Dahl and Nygaard, 1966], namely pseudo-parallel process execution using coroutines and Sequencing Set scheduling facilities. We developed the simulation library for several reasons, mainly code portability, availability of compilers and the necessity to use a freely available C library for parsing XML documents [libxml, 2003].

The simulation library includes random number generators and some statistical functions. Random variates of various distributions are generated using the uniform random number generator described in [L'Ecuyer, 1999].

The library provides basic estimation functions, e.g. mean, variance and confidence interval. Since we consider steady-state simulation, we discard an appropriate initial portion of the observations to remove the initialization bias. The mean is computed using the method of independent replications [Banks, 1998]. The simulation stops when the relative width of the computed confidence intervals is smaller than a given threshold. Both the confidence level and the threshold can be defined by the user. If they are not provided, we assume default values of 90% confidence level and 5% threshold.

Each UML Actor is mapped into an appropriate `Workload` object, that can be an open or closed workload, depending on the stereotype which is applied to the UML element. Workloads are active objects in the simulation, which simply perform an endless loop in which they select and activate a Use Case, whose behavior is activating the root step of its associated Activity diagram. Each Activity in the Activity diagram is translated into the corresponding kind of `AbsStep` object. These objects simulate execution of the corresponding step or scenario according to the structure of the diagram and the value of the associated tags. At the end of the simulation, the computed average response time of each `AbsStep` object (that is, a step or scenario) is inserted into the original UML diagram the `PArespTime` tag.

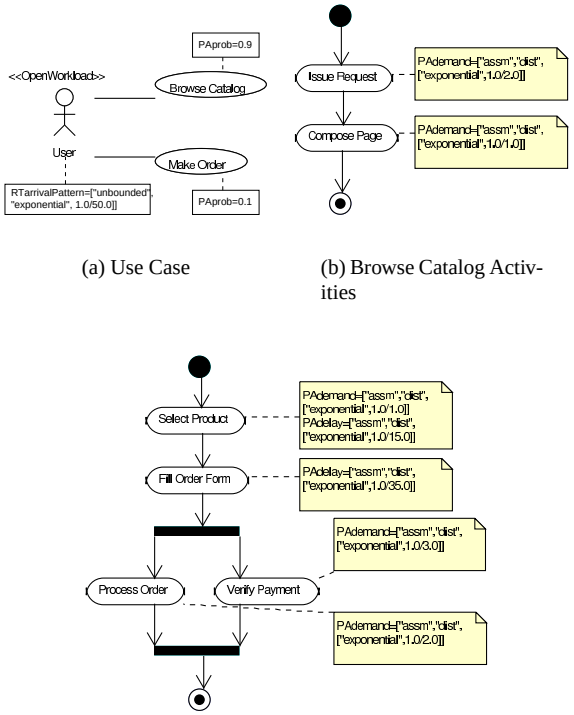
We propose to apply the UML-PSI tool as described in the modeling cycle illustrated in Figure 1. Namely, the software designer defines an UML SA with ArgoUML and specifies model parameters as tagged values associated with UML elements. Then, the user optionally selects parameters such as simulation length or confidence interval width, and runs the simulation. When the simulation finishes, the results are automatically inserted into the UML diagrams. At this point the software designer accesses the simulation performance results by opening the ArgoUML project file to access the results, and then possibly iterates the performance evaluation analysis by providing a new set of parameters in the UML model tags.

4 CASE STUDY

In this section we illustrate with a simple example how UML-PSI works. The example involves an e-commerce application in which users can browse a web catalog of products or submit purchase orders. We assume an unlimited stream of users requiring service. Users interarrival time is exponentially distributed with mean A. Users can browse the online catalog with probability p, and make an order with probability $1-p$. Browsing the catalog involves two sequential activities, which are: issuing a request to the product database and composing the web page. Making an order involves the following activities: selecting a product, filling the order form, processing the order and verifying the payment informations. Orders must be paid by credit card, whose number must be validated. Order validation and credit card checking are performed in parallel.

The UML Use Case and Activity diagrams are depicted in Figure 3. Model elements are tagged according to the notation described in Section 2.

Note that an analytical model of the system of Figure 3 can not be easily evaluated due to the fork/join component of the Make Order Activity. The simulation model which is derived from the SA is made of several active components (processes) arranged according to the structure of Figure 2. The actor is represented by



(a) Use Case

(b) Browse Catalog Activities

(c) Make Order Activities

Figure 3: UML representation of an E-commerce application.

an object of type `OpenWorkload`, which generates a stream of users. Use Cases are objects of type `PScenario` and have a queue which collects users according to their arrival order. Users select the Use Case to join according to the associated probability. Use Cases simulate one user request at a time activating the root step of the corresponding `PScenario`. When the scenario execution completes, a new user requests in the Use Case queue is started. Finally, there is one simulation process for each node in the Activity diagram. Nodes can be Activity nodes, which are modeled by processes of type `PStep`, or fork and join nodes which are modeled by processes of type `PStep_fork` and `PStep_join`, respectively.

Scenario	Average delay (seconds)
Make Order	$[2.98394, 3.00172]$
Browse Catalog	$[54.7787, 55.7254]$

Table 1: Simulation results for the e-commerce application of Figure 3.

A numerical example of the performance results, computed as steady-state average delays are reported in Table 1. The results are obtained by setting $A = 50sec$ and $p = 0.9$; other model parameters are set as shown in Figure 3. The computed intervals are at 90% confidence level. The results are inserted into the original UML diagrams as values of the `PArespTime` tag. These

tagged values are attached to each relevant model element. The software designer can now explore different situations by repeating the modeling and performance evaluation process with different tag values.

5 CONCLUSIONS

We have proposed a simulation-based performance modeling approach for UML software architectures. The notation we have used to describe performance parameters is a subset of the one defined in the UML Performance Profile. We have derived a simulation model from Use Case and Activity diagrams. The simulation model is executed and the results are inserted into the original UML diagrams as tagged values. We have presented UML-PSI, a prototype tool to implement the methodology.

The proposed approach has been defined to evaluate software performances at the SA design level. We plan to extend this approach to further steps of the software development process, by considering Deployment diagrams and resource allocation. Further research will be devoted to a more complete set of performance measures. UML-PSI will be extended accordingly.

REFERENCES

ArgoUML (2003). ArgoUML – Object-oriented design tool with cognitive support. http://www.argouml.org/.

Arief, L. B. and N. A. Speirs (1999a, June). Automatic generation of distributed system simulations from UML. In *Proceedings of ESM '99, 13th European Simulation Multiconference*, Warsaw, Poland, pp. 85–91.

Arief, L. B. and N. A. Speirs (1999b, November). Using SimML to bridge the transformation from UML to simulation. In *Proc. of One Day Workshop on Software Performance and Prediction extracted from Design*, Heriot-Watt University, Edinburgh, Scotland.

Arief, L. B. and N. A. Speirs (2000, September). A UML tool for an automatic generation of simulation programs. See [Proceedings of WOSP 2000, 2000], pp. 71–76.

Balsamo, S., A. D. Marco, P. Inverardi, and M. Simeoni (2002, December). Software performance: state of the art and perspectives. Technical Report MIUR SAHARA Project TR SAH/04.

Banks, J. (Ed.) (1998). *Handbook of Simulation*. Wiley–Interscience.

Cortellessa, V. and R. Mirandola (2002, July). PRIMA–UML: a performance validation incremental methodology on early UML diagrams. In *Proceedings of the Third International Workshop on Software and Performance (WOSP 2002)*, Rome, Italy. ACM Press.

CPAN (2003). Comprehensive Perl Archive Network (CPAN). http://www.cpan.org/.

Dahl, O.-J. and K. Nygaard (1966, September). SIMULA–an ALGOL-based simulation language. *Comm. of the ACM 9*(9), 671–678.

De Miguel, M., T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec (2000, September). UML extensions for the specifications and evaluation of latency constraints in architectural models. See [Proceedings of WOSP 2000, 2000], pp. 83–88.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns: Elements of reusable Object-Oriented programming*. Addison–Wesley.

L'Ecuyer, P. (1999). Good parameters and implementations for combined multiple resursive random number generators. *Operations Research 47*, 159–164.

libxml (2003). `libxml`: the XML C library for Gnome. http://xmlsoft.org/.

Object Management Group (2001, September). Unified modeling language (UML), version 1.4.

Object Management Group (2002a, March). UML profile for schedulability, performance and time specification. Final Adopted Specification ptc/02-03-02, OMG.

Object Management Group (2002b, January). XML Metadata Interchange (XMI) specification, version 1.2.

Pooley, R. J. and P. J. B. King (1999, February). The Unified Modeling Language and performance engineering. In *IEE Proceedings – Software*, Volume 146, pp. 2–10.

Proceedings of WOSP 2000 (2000, September). *ACM Proceedings of WOSP 2000, 2nd International Workshop on Software and Performance*, Ottawa, Canada.

Smith, C. U. (1990). *Performance Engineering of Software Systems*. Addison-Wesley.

Smith, C. U. and L. Williams (2002). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison–Wesley.

Wall, L., T. Christiansen, and J. Orwan (2000, July). *Programming Perl* (third ed.). O'Reilly & Associates.