Generalized Simulation Models: What, Why and How?

David Raffo, Greg Spehar and Umanath Nayak
School of Business Administration
Portland State University
davidr@sba.pdx.edu, spehar@pdx.edu, nayak@pdx.edu

Abstract

In this paper, we define a generalized simulation model as one that can be easily adapted to multiple process contexts using significantly less effort than would be required to develop the original model. We discuss the obstacles and issues associated with developing and maintaining simulation models. We then review the research that has been done to make software process reuse and develop software product families. Finally, we will state our position that developing generalized simulation models can offer important benefits in terms of saving time and improving the quality and usability of the model.

Key Words: Software Process Modeling, Process Reuse, Process Template, and Process Simulation.

1. Introduction

There is a great need for better project and process management. Software process simulation (SPS) modeling has shown some results in addressing the following questions as well as others [3]:

- Would it be better to perform work in-house or to out-source (subcontract) it?
- Would it be more beneficial to employ a product-line approach to developing similar systems, or would a more traditional, individual product development approach be better suited for a given project?
- What is the likely long-term impact of current or prospective policies and initiatives (e.g., hiring practices, training policies, software process improvement initiatives)?
- What would be the impact on a given project when choosing between different process alternatives?
- Would adding a selected process improvement achieve the productivity goals for a new project?
- Would limiting or removing a subprocess save cost and schedule, without serious impact to product quality?

There are a number of benefits that can be achieved by using software process simulation models by companies at all levels of the CMM [9, 10]. Software process simulation is directly linked to software project planning and process improvement activities. So organizations can use software process simulation at all levels of maturity to obtain significant benefits, which include:

- Designing and defining software processes
- Improving organizational decision making
- Providing justification for process improvement initiatives
- Predicting project level performance
- Predicting the impact of process changes before they are actually implemented
- Assessing financial measures of performance improvement
- Quantitatively assessing risk
- Supporting "what if" analyses
- Refining metrics programs.

Currently, industry is not using SPS models (SPSMs) to their full advantage. Typical reasons sited include:

- Process models tend to be difficult to build and maintain.
- The metrics data for the process models can be difficult to acquire (or may not exist).

As a result, the perception is that process simulation models are costly to build and it is difficult to obtain timely results from them.

What can be done to enable software process simulation technology to be more readily used and more widely applied in industry? One possible approach is to lower the cost of developing these models by designing a generalized software process simulation model (GPSM) that can be tailored quickly to a particular project scenario and deployed rapidly to be utilized effectively. Such a model would be designed to overcome some of the key obstacles that limit the diffusion of software process simulation technology. We believe some key aspects of SPSM technology that impact its adoption by industry include:

- Reducing model complexity
- Making the mechanics of building models easier (creating better tools)
- Making it easier to acquire model data

- Lowering the cost (i.e. the effort and expertise) associated with developing and SPSM.

2. Obstacles and Issues

In this section, we discuss some of the obstacles and issues associated with developing and maintaining SPSMs in more detail.

2.1 Building Simulation Models

The literature has indicated that building simulation models has been a challenging and time consuming task. It should be recognized that SPS models can be developed at different levels of scope and depth to suit an organization's needs. This can range from qualitative models that are used to design and define the software development process to detailed models that utilize up-to-date metrics and are used to support software project management activities [10, 12]. In general, the higher the level of detail and fidelity to the process, the greater amount of effort required to build the SPS model.

One reason is the complexity of the software development process being modeled. Kellner et al [3] suggest that certain types of complexities like system uncertainty and stochasticity, system dynamic behavior, and system feedback mechanisms are often encountered in software development systems. Kellner et al argue that simulation models are particularly well suited to capture these types of complexity and support decisions compared with other modeling paradigms. At the same time, just as system complexity is a major cost driver of software development projects, process complexity and uniqueness drive the cost in terms of effort and expertise required for all phases of SPS model development. This includes: defining model scope, defining model parameters, building and debugging the model, and exercising the model and analyzing the results.

Obtaining quantitative and qualitative data for simulation models is another significant cost associated with SPS model development. Some SPS models require hundreds of parameter values to be populated and maintained. Companies may not collect the data that are needed for the model on a regular basis or in forms that are readily usable. In addition, it may be necessary to interview staff or extract the data from written documents.

Moreover, even if the data do exist, organizational issues pertaining to company security, company confidentiality and company politics can confound data collection efforts for the SPS model. It may be that people within the organization are unable or unwilling to provide timely and accurate qualitative and quantitative data.

When developing SPS models, the tools themselves can have a large impact on the effort and expertise required to develop the model. Most simulation tools are very general and are not tailored to address the software industry. Most discrete event simulation packages were developed for the manufacturing sector. Although the constructs that have been developed for manufacturing can be applied to software development, modelers face the added burden (paid in terms of increased effort and expertise necessary) to construct an architecture and supporting blocks that are applicable to software development processes. Moreover, the traditional queuing, utilization, and bottleneck analyses which are fundamental to models of manufacturing operations may not be as relevant to software development operations where the operation times are much longer and more flexible. Individual developers may be assigned to multiple pieces of the system. As a result, the productivity associated with any one piece of the system can vary as work assignments vary. This is different from a traditional manufacturing system where a work piece is either being processed or it's not.

2.2 Expertise required to Develop and Utilize SPS Models

Three primary skills that are involved in developing SPS models are:

- Model designer/developer
- Data collector/analyst
- Model analyst/business case analyst

Model designers and developers need high-level skills for designing models of complex systems as well as detailed skills associated with a specific simulation tool or language. Experience in building models is a great help. Also access to senior modeling experts can greatly improve the quality of the model and the speed with which the model is developed and verified. We believe that all of the research describing the variation in developer productivity and capability is applicable. Novice personnel require more time to build the model, collect data and analyze and interpret complex simulations. Utilizing junior personnel exclusively to build a model can lead to serious problems. Having a tool that provides an architecture and model structure in the form of templates and basic model building blocks would significantly reduce the expertise required of the modeler.

Data collectors and analysts are responsible for developing model parameters from raw product and process data. An understanding of the model is required to define model parameters and to develop appropriate values from raw data [11]. Having a tool that pre-populates a model with industry data provides an example of the kind of data necessary and provides initial values that can be refined by a company.

Model analysts and business case analysts are responsible for examining model output and to construct a business case that can be utilized by management when making decisions. A certain understanding of basic business concepts, financial measures and statistics is required. This can be obtained through books and tutorials at major conferences [1]. Having a tool that provides analysis templates can support the analyst as they respond to management queries and assess various strategies as they exercise the SPS model.

It is also important to provide training to managers who will utilize model results to make decisions. Stochastic model results are not as simple to interpret as having a single number. Stochastic model results provide a quantitative assessment of risk. These results provide more information and insight to the manager about the situation.

3. Research Review

This section discusses research that is applicable to generalizing process simulation models (GPSMs). This work centers on three different areas: Modularization Concepts, Software Product-Line Family Concepts, and Cognitive Pattern concepts/techniques. Each of these will be discussed further in the following sub-sections and their relevance to defining GPSMs will be discussed in section 4.

3.1 Modularization

David Parnas is one of the pioneers in describing the virtues of modularization in programming code. This concept of modularization has been widely disseminated and has contributed to the development of the structured programming and the Object Oriented Programming techniques. The basic idea consists of designing logic to the appropriate level, finding and reusing logic in tasks, defining logic into modules that reference other small modules and hiding logic complexity whenever possible behind an interface.

In the paper describing the idea of levels of design logic, Parnas and Clements [8] discuss the use of the design process to find a "rational" manner in which to design software. The use of tools to create the design products helps to define a process that can be "viewed" as rational and that can closely approximate a "rational" process. Parnas and Clements suggest that the software process is most useful when it is defined to the level in which it best explains the process that produces the most necessary software design artifacts. In addition to this, Parnas [6] discusses the virtues of designing software for ease of extension and contraction. This "designing for change" enables increased levels of reuse. Within this context, Parnas indicates that the software programs are "abstract mathematical objects" where as "the software engineers' techniques for responding to anticipated changes are more subtle and more difficult to grasp than the techniques used by designers of physical objects". Thus the effort needed to perform the design process is difficult and effected by change. So looking at the software code for commonalties can have positive impacts by defining families that perform "common aspects" or common tasks. Defining these families can "...exploit the commonalties, share code, and reduce maintenance costs."

Furthermore, Parnas [4] states that the module structure "...is based on the decomposition criterion known as information hiding. According to this principle, system details that are likely to change independently should be the secrets of separate modules; the only assumptions that should appear in the interfaces between modules are those that are considered unlikely to change." Thus the definition of one module accessing another module would require that the information pass through an interface. This interface then protects the functions and information from each module and allows for the abstraction of the information into aspects or tasks that can be understood and maintained. Also, Parnas et al [7] sets the goal for modularizing software: "The primary goal of the decomposition into modules is reduction of overall software cost by allowing modules to be designed and revised independently." And finally, Parnas [5] introduces the concept of a "family" of products where the concept allows for gaining efficiencies if the logic of the program is treated as a family of programs instead of a "sequence of individual programs." This concept of family is focused more on the idea that as a program changes in time, it may have several versions that will be maintained.

Ribo et al. [13] use the information hiding concept defined within a software program context and apply this to a process definition context. According to Ribo, "There are several issues that are connected with [process] reuse: the ability to identify and generalize useful parts of already constructed process models in order to be reused later *(harvestin*g); the adaptation of these models to be effectively reused in the construction of a new one...". Thus the leap from the use of these constructs of information hiding in software can be related to defining processes that can be reused using the same methods.

3.2 Product Families

Weiss and Lai [14] take the family concept advocated by Parnas to the next level by proposing the use of software product-line engineering, a family-based software development process to build a product within

the context of a "family" of products. They use a process framework called FAST (Family-Oriented, Abstraction, Specification, and Translation), which relies heavily on abstractions. These abstractions can be used and reused in a variety of ways such as to identify and specify commonality and variability in systems. The FAST approach focuses on reuse through the lens of family-oriented development and helps us in taking advantage of the reusable aspects in producing software. Some of the stated benefits of using FAST are:

- More efficient production of family members.
- Ability to support greater variety of family members.
- Graceful aging of family members.
- Systematizing knowledge capture.
- Rapid software production.
- Improved quality.
- Competitive advantage in domains of application.
- Facilitate economic analysis.

From this "family" concept, Weiss and Lai provide an approach for developing a product line family through a defined set of analyses. FAST processes are decomposed into three sub processes: identify families worthy of Investment (qualifying the domain), identify the addition of facilities for producing family members (engineering the domain) and identify the facilities for producing family members very rapidly (engineering applications).

3.3 Cognitive Patterns

In section 3.1 we described how Parnas advocated the development of systems with modules for data hiding. In section 3.2 we described how Weiss advocated the development of Product Families within the concept of defining the complex environment that occurs with highly complex systems. In this section, we describe how Gardner closes the loop by defining processes that assist in mapping the real world into cognitive patterns.

In his book called "Cognitive Patterns" [2], Gardner presents the idea that process "frameworks" can be used to define different logical parts of a software program. He further defines the idea of domains, frameworks, cognitive maps and design patterns that are all examples of cognitive patterns. According to Gardner, "Frameworks are considered to be specific, context driven examples of cognitive patterns called domains. The term cognitive map, represents a kind of framework...". He further states Design Patterns as "...detailed contextual descriptions of object behavior and communication."

The use of the cognitive maps can be traced to what Gardner calls "Problem-Solution Templates" (PST). This concept was adopted from the research performed for the development of KADS (Knowledge Acquisition and Design Structuring) by several European groups. With KADS, process logic problems can be defined and solved through a process of defining a concept description, a process description and a strategic description. Within a KADS Object, the selection of the Problem Solving Template (PST) can be preformed as the basis for solving the logical problem. The "Problem Solving Template Component" group then consists of a set of diagrams of the context specific cognitive patterns used by a particular organization, process, system or individual. The PST will then illustrate the underlying reasoning patterns used to solve the logical problem, reach the logical conclusion or obtain the logical result.

4. Synthesis

4.1 Defining the GPSM

Given the overall obstacles and issues and the current state of the research in the area of software development and reuse, we believe that developing GPSMs can offer important benefits in terms of saving time and improving the quality and usability of software simulation models. With the simulation modeling effort defined by qualitative logic (how the model is assembled) and quantitative data (what data are represented in the model), it is important to define the GPSM component logic with the following modularization framework generated from the works by Parnas:

- Partitioning the Simulation Process Logic
  - Reducing Logic (Complexity)
  - Protecting Logic (Errors)
- Changing the Simulation Process Logic
  - Internal (High Changes)
  - Interface (Low Changes)
  - Grouping (Manageable Aggregation)
- Understanding the Simulation Process Logic
  - Document Internal
  - Document Interface
  - Relevant Access

Using the concept of "family" as defined by Weiss et al [14] within the GPSM framework will be another important consideration. The family can define the overall simulation domains by processes being modeled and by the facilities that will be required to promote the development of family members and finally the ability to rapidly create these members into useable simulation applications that perform within their context. Making use of the FAST processes, simulation development can identify families of simulation applications that are worthy of investment, identify the addition of engineering facilities to assist in producing simulation family members, and identify the engineering facilities for producing family members rapidly. So it is clear, if

applied within the proper domain, that the family product line concept could be used to create more robust and better-defined models than defining unique models for each simulation situation encountered.

Finally, we believe that the concept of "cognitive patterns" is the part that will bring the modularization concept and the family concept together. This concept is very similar to the process defined by the Product-Line family. But it is unique in that the cognitive patterns concept attempts to define the general processes that define the necessary steps or patterns, called Problem-Solution Templates (PST) by Gardner, for a consistent aspect or solution to a problem. The use of cognitive patterns, as advocated by Gardner, allows the designer of a GPSM to manage complexity, define the simulation scope and boundaries, create a consistent vocabulary, identify the necessary general simulation components, incorporate the situational knowledge rapidly and generate a consistent and valid model accessible to differing skill levels.

Since we have defined what a GPSM model might look like, it should be noted that great care is required when adapting the GPSM to a specific context. Many of the assumptions and empirical results used in one context may not be easily transferable to another context, so a new or different GPSM may be required.

4.2 GPSM Project Costs

With GPSMs, we also believe that the industry will be more interested in SPS Models since several issues pertaining to making the business case for developing SPS models can be addressed. The first issue that can be addressed with the use of GPSMs is "the cost to build the model verses the cost of not building the model". In other words, if the simulation model is less costly to build and deploy than the cost associated with making poor process change decisions, we would choose to build the simulation model. We suggest that GPSMs be employed early in the SDLC to offer the most benefit and to support process planning and design decisions.

The second issue is that managers have to consider the overall cost of the modeling effort compared to the overall cost of the project. If the overall cost of the simulation modeling is more than some small portion of the overall allocated cost of the software development project, one would not build the simulation model. We suggest that using GPSMs can reduce cost of the modeling effort enabling smaller projects to benefit from the technology. As a result, the overall number of projects that could benefit from process simulation modeling would increase.

5. References

[1] Boehm, Barry, "Value Based Software Engineering", Tutorial presented at the International Conference on Software Engineering, Held in Portland, Oregon, USA, May 5, 2003.

[2] Gardner, K., "Cognitive Patterns: Problem-Solving Frameworks for Object Technology", Addison-Wesley, 1998, 250, pgs., ISBN 0-521-64998-6

[3] M. I. Kellner, R. J. Madachy, and D. M. Raffo, "Software Process Modeling and Simulation: Why, What, How," Journal of Systems and Software, Vol. 46, No. 2/3 (15 April 1999).

[4] Parnas, D. L. "On the Criteria To Be Used in Decomposing Systems Into Modules", Communications of the ACM, Vol. 15, No. 12, pp. 1053-1058, December 1972.

[5] Parnas, D.L., "On the Design and Development of Program Families'", IEEE Transactions on Software Engineering, Vol. SE2, No. 1, March 1976, pp. 1-9.

[6] Parnas, D. L., "Designing Software for Ease of Extension and Contraction", IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 128-137.

[7] Parnas, D. L., "The Modular Structure of Complex Systems", IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, March 1985, pp. 259-266.

[8] Parnas, D. L., Clements, P.C., "A Rational Design Process: How and Why to Fake It", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp. 251-257.

[9] D. M. Raffo, "Getting the Benefits from Software Process Simulation", Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE'99), Held in Kaiserlautern, Germany, June 1999.

[10] D. M. Raffo, J. V. Vandeville, and R. H. Martin, "Software Process Simulation to Achieve Higher CMM Levels," Journal of Systems and Software, Vol. 46, No. 2/3 (15 April 1999).

[11] Raffo, D. M. and M. I. Kellner, "Empirical Analysis in Software Process Simulation Modeling" Journal of Systems and Software, Vol. 47, No. 9, 2000, pages 31-41

[12] D. Raffo, W. Harrison, J. Vandeville, "Coordinating Models and Metrics to Manage Software Projects", International Journal of Software Process Improvement and Practice, 5:2/3, June/September, 2000, pages 147-157.

[13] Ribó J. M.; Xavier Franch X. A precedence-based approach for proactive control in software process modelling, ACM International Conference Proceeding

Series, Proceedings of the 14th international conference on Software engineering and knowledge engineering, Ischia, Italy, 2002.

[14] Weiss, D. M.; Lai, C.T.R., "Software Product-Line Engineering: A Family-Based Software Development Process", Addison-Wesley, 1999, 448, pgs., ISBN 0-201-69438-7.

Biographies

David Raffo

Dr. Raffo is an Associate Professor of Technology Management and Information Systems at Portland State University. His research interests include: software process modeling and simulation, strategic software engineering, and process improvement. He has over twenty-five refereed publications in the field of software engineering and has received research grants from the National Science Foundation, the Software Engineering Research Center (SERC), NASA, IBM Corporation, Tektronix Corporation, and Northrop-Grumman Corporation. Prior professional experience includes programming as well as managing software development projects. He has also taught in Purdue University's Software Engineering Retraining Program (SERT) through the Department of Computer Science. Dr. Raffo received his Ph.D. at Carnegie Mellon University.

Greg Spehar

Mr. Spehar has over 12 years of development experience working as a software engineer and project manager. He has worked for companies such as McDonnell Douglas, Federal Express, GemStone, Nike, the State of Oregon, Brokat Technologies, and the Bonneville Power Administration. During his 4 years as a systems engineer with McDonnell Douglas, Mr. Spehar worked on both the STS-49 and the re-engineering of the International Space Station. Where the primary work was in the area of simulation of the Space Station Requirements utilizing RDD-100. Mr. Spehar received a B.S. in Aerospace Engineering from Purdue University and a MBA at the University of Texas at Austin.

Umanath Nayak

Mr. Nayak has over nine years of software development with expertise. He has worked on a wide range of systems like Carrier Access Billing, Message Processing, Charge Card, Job Management, Activity Management, Error Tracking, and Payroll Systems. His simulation experience involved using Extend software to simulate software development processes. Mr. Nayak has an MBA from Portland State University with an emphasis on Management of Innovation and Technology. He received B.E. in Electronics and Communications Engineering from Mangalore University in India.