To Main Proceedings Document

Options in Source Code Documentation

Authors: Donald Habib, Kevin A. Ryan, and Greg Love
Project Performance Corporation, McLean, Virginia, USA

ABSTRACT

Customers using system dynamics models for multipurpose applications require varying degrees of documentation of a model's source code and design. A broad range of factors can influence the customers' documentation requirements. Because the model developer plays an important role in determining a model's functional requirements and has often been exposed to a range of model applications and documentation requirements, the developer is in an excellent position to understand and advise the customer on what level and form of source code documentation may be appropriate for a specific application.

This paper will discuss the range of documentation alternatives considered by Raytheon Company C^3I Systems, Project Performance Corporation, and U.S. Government customers in documenting source code for various system dynamics models. The paper also discusses the factors that were considered as potentially influencing the customer's documentation requirements. The most expeditious documentation is often developed automatically when using commercially available off-the-shelf system dynamics software. More rigorous documentation can also be developed manually, either during or after development of the model. In addition, rules-of-thumb and informal professional practice standards exist for documenting source code.

Source code documentation serves two primary purposes. First, it is a quality assurance tool that helps to minimize the number of defects or bugs in the source code. Second, it can lower the life-cycle cost of the model by reducing the learning curve and costs associated with evaluating and modifying the source code. These two purposes are the dimensions of the cost-risk decision that model owners face when making documentation decisions. Factors considered in the decision include the model purpose, the expected frequency and duration of model usage, the experience and knowledge of model users, the potential costs associated with model misuse or failure, and the direct costs of preparing and maintaining documentation.

May 5, 2000

Contact Information:
Donald Habib
Project Performance Corporation
7600 Colshire Drive, Suite 500
McLean, Virginia 22102 USA
Email: dhabib@ppc.com
703-748-7117

Options in Source Code Documentation

INTRODUCTION

Raytheon Company C^3I Systems and Project Performance Corporation (PPC) (the Team) have developed a variety of system dynamics models for the U.S. Government to address a broad range of systems and problems. At the end of the model development cycle when the model is turned over to the government customer, the deliverable includes both a model documentation package as well as an electronic copy of the model. The model's source code is an important element of the documentation package, and also includes a users manual, examples of model runs, information describing the conceptual model, and model validation information.

The content of the source code documentation can vary depending on the ascribed purpose of the model, as defined by the customer. Just as a model can be developed to meet a broad range of customer needs, the content of source code documentation should also vary depending on customer needs.

During recent modeling efforts, the Team received a requirement from their government customers stating that models and simulations must be documented in a manner that are sufficient to describe what each model does, what it contains, and how it works. However, the customer did not provide specific guidance that outlined what constituted "sufficient model documentation," particularly with respect to source code documentation. Consequently, the Team developed several source code documentation options arrayed along a level of effort scale. This paper summarizes those options and describes the factors considered in determining the most appropriate option for a given model.

DETERMINING FACTORS FOR SOURCE CODE DOCUMENTATION

In models that are used for multiple purposes or used repetitively to address several similar applications over time, source code documentation serves two primary purposes. First, it is a quality assurance tool that helps to minimize the number of defects or bugs in the source code. Second, it can lower the life-cycle cost of the model by reducing the learning curve and costs associated with evaluating the model for its applicability to a specific problem and, if necessary, modifying the source code. These two purposes are the dimensions of the cost-risk decision that model owners face when making documentation decisions.

In making a documentation decision, the model owner must weigh the likelihood (and cost) of incorrect model results against the investment needed to develop source code documentation. The factors considered by both the model developer and the model owner include the overall purpose of the model, the expected frequency and duration of model usage, the profile of model users in terms of their experience and knowledge, the potential costs associated with model misuse or failure, and the cost of preparing and maintaining documentation. Each of these factors is described below.

Overall Purpose. Models can be developed for several purposes. A robust model may be developed to address a range of problems, while a highly customized model may be developed to solve a highly specific problem. The level of source code documentation should shift accordingly.

A robust model may be developed, for example, to optimize the performance of a system whose inputs change periodically but whose basic design remains essentially the same over time. These types of models can also be used in training situations where a model can be re-used many times to solve and learn about different problems. Each time, the model user (or model designer) can customize the model by varying input data. For this reason it is necessary to understand the model's original purpose, the assumptions that went into its design, and the limits under which the model can produce valid results. Much of this information can be reliably stored in and retrieved from the stock-flow diagrams, model code, or other forms of source code documentation.

In contrast, it is also common for models to be used to solve narrowly defined manufacturing or work process problems. In this case, the nature of the problem is often unique. When a new problem arises, it is often--but not always--different enough from preceding problems that a model used to address the new problem is most efficiently developed from scratch rather than by revising a previous model. In these cases, the value of source code documentation is to allow the user to more easily determine the applicability of and, if necessary, revise an existing model. If the purpose of the existing model is well documented in the source code, the user can more easily determine whether the existing model is applicable, with either no or limited revisions, to the new problem. Without appropriate documentation, the cost of evaluating and adapting the existing model may outweigh the cost of developing a new model from scratch.

Frequency and Duration of Model Usage. From a quality assurance standpoint, a key benefit of source code documentation is to ensure that the model has not been adulterated after the model developer delivers it to the user. As the time period between initial receipt of the model and its use grows, there is an increasing risk that the model may include inadvertent changes. This is particularly true for models stored electronically on erasable media (e.g., hard disks, diskettes, zipdisks®, and tapes). For this reason, source code documentation becomes very useful when the corresponding model is used over a long time period. It can be used to illuminate the differences between the model in its present state and the original version, or to simply verify that the model has not been changed.

Profile of Model Users. The single common factor to all model users is their stake in the successful understanding of the system being modeled. An understanding of the model user's experience, training, and responsibilities is needed in model development and should be reflected in model's user interface and documentation. There exists a broad spectrum of interests and responsibilities within the universe of model users.

- *Users with Limited Knowledge of System Dynamics and the Modeled System.* Model developers inevitably encounter customers with limited knowledge of system dynamics as a field. In addition, the user may be unfamiliar with the system being modeled (newly hired employee, new to the project, etc.). These limitations can be addressed in a model's source code documentation. From the model developer's standpoint, source code documentation can be used to aid the user in understanding the functions, capabilities, and limitations of the model.

Recognizing the potential involvement of new and inexperienced personnel is a critical factor in decisions about model code documentation. In the normal evolution of many organizations, new personnel arrive; existing personnel leave or acquire new responsibilities; and the organization's responsibility structure changes. As a result, the individuals identified as key personnel responsible for a system can change, and new personnel can replace the original users, owners, or developers of a model. New personnel will typically have more limited knowledge than their predecessors of a system's history and performance, including its problems and the use of system dynamics models to address them. For these individuals, source code documentation is a useful asset in deciding whether and how to use an existing model. Equally important, the documentation can be structured to serve as a training tool to educate the user on the performance drivers and limitations of the system and, in appropriate circumstances, in the teaching of system dynamics theory and fundamentals.

- *Users with system dynamics exposure*. At the other end of the spectrum, the model developers might be the same as the model users. These individuals have knowledge of the system; the problem being studied; the science of system dynamics; and the purpose, limitations, and internal design of the model. Source code documentation for these users, while still useful, may be different from that used by inexperienced users. The documentation decision should consider that some aspects of the model are so basic as to not require documentation. For example, if the model is stored on non-erasable media (e.g., a CD) and the user has the appropriate software to access the actual source code, separate paper copies of stock-flow diagrams and source code may be unnecessary. In another example, a user knowledgeable about system dynamics, software use, and the modeled system may not require source code information to be incorporated into the model's user interface.

Because characteristics of the model user are important factors in several key areas of model design, it is a good practice for a model developer to ask questions to find out as much as possible about the primary model user. Information on model user characteristics is generally useful in developing the model's user interface, which can be customized to include elements of source code documentation appropriate for the user.

In the best situations, the customer has a detailed understanding of who will use the model and what the user's skills and knowledge will be. Conversely, it is possible for a customer to be uncertain about who the model users will be. A situation where the model users are poorly identified poses difficult challenges both in determining the content and format of source code documentation, the user interface, and in developing other model documentation. Comprehensive source code documentation can allow a user to evaluate a system behavior problem, determine the applicability of the model, make appropriate changes to the model, and effectively use the model to solve the problem. It can also eliminate the need for the model user to enlist the services of an outside modeling consultant or to track down the original model developer, neither of which may be available (or within budgetary constraints). Determining the appropriate level of source code documentation can be achieved by considering future user scenarios during the initial model development period and the preparing a documentation package commensurate with identified needs.

Cost of Model Misuse or Failure. When deciding an appropriate level of source code documentation, the risk and costs associated with the decisions a model recommends should be considered. If a model is used to make decisions involving high-risk, high-cost issues or irreversible consequences, such as the operation of safety-related systems or a commitment of capital for a project, the justification is strong for thorough source code documentation. In such instances, the source code documentation may provide an analysis of every section of model code, discussing its purpose, applicability, functionality, limitations, and relationship to other sections of code. More extensive source code documentation functions to provide greater assurance the model is being used properly.

In contrast, other situations call for a more restrained approach in developing documentation. For example, less documentation may be necessary when a model is being used to make a decision that is easily reversible or has limited consequences. For example, if a model is being used to determine what settings of a continuously monitored manufacturing process will result in the most efficient performance, evidence of an incorrect setting will be apparent from system monitoring, and the setting can be adjusted without experiencing additional costs, then there is a limited penalty for misusing the model. Similarly, less rigorous documentation may be appropriate if a model is being used as only one element in a management analysis toolkit, and its results will be further scrutinized before a decision is implemented. In these situations, a large investment in source code documentation may not prove to be cost-effective.

Cost of Preparing Documentation. In examining the different ways that the source code of a model can be documented, an obvious observation is that some types of documentation can be produced at minimal cost while others require a considerable investment. In between these two extremes, there exists a spectrum of documentation options with a variety of costs. The experience of the Team has found that customers are more willing to require—and pay for—extensive source code documentation in situations where the need for extensive documentation is suggested by the factors discussed above. The Team has encountered various customers that require models that are versatile enough to address a wide range of problems, usable for several years, designed for a potentially inexperienced user group, and can be relied upon to make critically important decisions. These conditions argue in favor of a larger investment in source code documentation.

Options for Source Code Documentation

This paper identifies a series of five options for documenting source code that have been developed in consultations between the Team and its customers. These options are not the only options available, but they are representative of the broad range of available options. The documentation options vary in several ways: by their cost to develop, their level of rigor and thoroughness, and their user-friendliness (Figure 1). The options presented are cumulative. That is, each successive option builds upon the previous option by adding more detail and rigor. However, any single option or combination of options may be appropriate for a given modeling project.

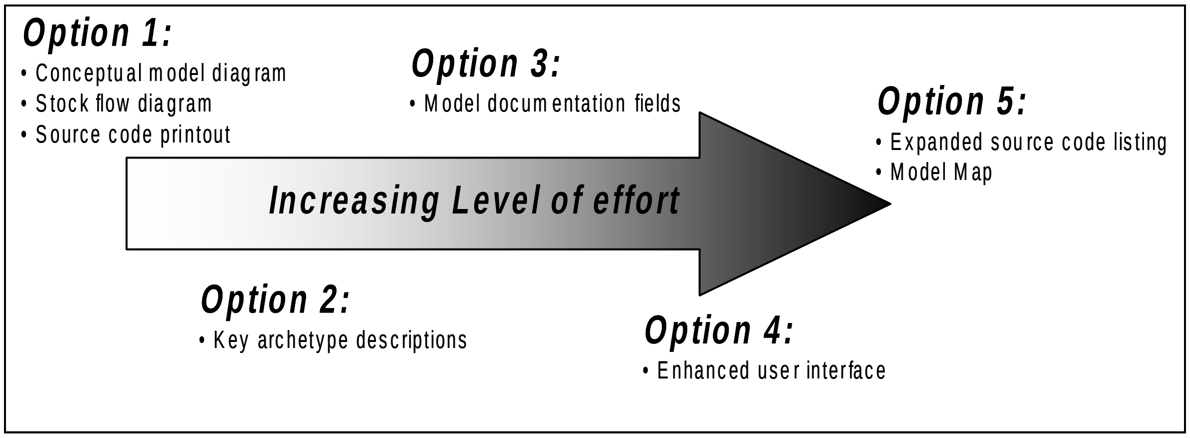
Option 1:
• Conceptual model diagram
• Stock flow diagram
• Source code printout
Option 3:
• Model documentation fields
Option 5:
• Expanded source code listing
• Model Map
Increasing Level of effort
Option 2:
• Key archetype descriptions
Option 4:
• Enhanced user interface

FIGURE 1. INCREASING LEVEL OF EFFORT OF SOURCE CODE DOCUMENTATION OPTIONS

Option 1 – Automated and Existing Documentation

This option includes those materials that the modeling software (Powersim® and ithink® are the primary software packages used by the Team) can produce automatically or that are prepared for other purposes during model development. These materials include:

- A brief introductory narrative describing what the model does and how it works.
- A printout of the conceptual model in the form of block flow or causal loop diagrams.
- A printout of the model stock-flow diagrams.
- A printout the source code associated with the model.

These materials can be prepared in paper and electronic formats and provided to the model owner either separately or as an addendum to other model documentation, for example, a users' manual or verification and validation plan. For purposes of source code documentation, the electronic versions of the materials accessible through the modeling software are copied, where possible, into more commonly used business management software (i.e., word processing and graphics presentation software), so that the information is more broadly available. That is, system dynamics software would not be needed to access the documentation.

The advantage of Option 1 is that it can be completed with the least level of effort. This option entails little additional model documentation effort beyond what is normally invested in a typical modeling project. The chief disadvantage of this option is that it provides the least complete package of information. It does not list, define, nor provide any explanatory text for the specific source code variables, constants, and equations or provide any description of the high-level architecture of the model.

Option 2 – Add Narratives and Diagrams of Key Model Archetypes

Option 2 encompasses the features of Option 1, but adds narrative descriptions and stock-flow diagrams of the key archetypes used in the model. Key archetypes include those used repetitively in the model and those involving a key area of the model. This may include, for example, the ordering and delivery of different raw materials from off-site (used repetitively for different types of raw materials) or processing bottlenecks of the existing system.

The narrative typically defines and explains—in plain English—the structure and functionality of each node (i.e., the reservoirs, flows, queues, ovens, conveyors, and converters, etc.) comprising the selected archetype. In this way, the model user is provided with direction in how to interpret these sections of the model. The narrative explains the relationship between the source code and model structure for the selected archetypes and allows the user to understand the thought behind the model. Besides its value from a source code documentation standpoint, the narratives and diagrams form an excellent training tool to educate new model users on the internal logic and functionality of the model.

Option 3 – Add Explanatory Text to Selected Documentation Fields

Most simulation software applications have accessible documentation fields for each node into which it is possible to insert an unlimited amount of explanatory text. This explanatory text can be used to describe the interrelationships between model nodes or to define each corresponding variable, constant, or formula. Whatever text is placed in each documentation field appears where the node is defined in a printout of the source code.

Depending on the nature of the model, this option may include populating the documentation fields for either all or some of the nodes. The Team has found that this information is most valuable for documenting converters and flows, and less important for documenting stocks. In populating the fields, a standardized protocol for presenting the information helps make the text easier to understand when reading it in the source code. Effective protocols attempt to reduce repetitive information, identify key functional and defining information, and present information in a consistent format. By populating the node documentation fields in this fashion, the model source code listing (which contains this information) becomes a more effective documentation tool.

The advantage of this option over Options 1 and 2 is that the user is provided with definitions of the equations, variables, and constants used in each equation associated with the converters and the flows. The principal disadvantage is that the information is not provided in a particularly user-friendly format in that there is no “map” identifying all the instances in which a particular variable or constant might is used.

Option 4 – Add Conceptual Diagrams and Node Documentation Electronically Linked to the Model's User Interface

This option includes enhancing the model's user interface, known as the management flight simulator, by including diagrams of the conceptual model. Further, the conceptual model and the stock-flow diagram are electronically linked to each other and to text boxes at key points. The content of the text boxes would be similar to the node documentation text, but would address high level model design issues, for example, the function of and relationships between various sectors or sub-modules within the model.

The primary advantage of this option is that it delivers information to the user in a highly user-friendly manner. That is, it is delivered graphically at the point of use. In addition, the type of information conveyed in this option addresses high-level model architecture issues not captured in documenting individual nodes.

Option 5 – Add Expanded and Formatted Source Code Listing and Model Map

This option builds upon Options 3 and 4 by expanding and enhancing the source code listing by presenting the node documentation information in a more user-friendly way and include high-level model architecture information within the source code listing. As a rule of thumb, approximately 20 percent of the lines of code should be documentation/comment lines.[1] The source code listing would be edited and formatted either by using the modeling software or in a word processing program in order to create a document that is better organized and easier to follow. For example, variables may be introduced and defined in a tabular format rather than in a sequential line-by-line listing of text. The Team has found that using macros (in a word processing program) simplifies the formatting effort, particularly when the source code may run for several thousand lines of code. For example, macros can be effectively used to more easily separating formulas from documentation/comment text and formatting the documentation/comment text where appropriate. When created in a word processing program, the source code document can be organized with a table of contents and indexed with dynamic links.

This option also would include a Model Map in the enhanced source code listing that cross-references each variable in the model to all the instances where it appears in the source code. Creating a "Model Map" would involve assigning reference numbers to each equation in the source code. This reference number could be sequential or could be prefixed with a "sector" letter linking it to a particular sector or sub-module of the model. The variable dictionary would then be amended to include the corresponding equation reference number(s) where each variable appears.

CASE STUDY IN SOURCE CODE DOCUMENTATION

A large manufacturing company that operates several similar plants is developing a system dynamics model. It has a particular interest in analyzing the productivity limitations and

[1] System Dynamics Modeling, A Practical Approach. R. G. Coyle. 1996. Page 145.

operational vulnerabilities at its plants. In particular, it wants to analyze the impacts on overall production related to the potential failure of key systems at each plant. The company has no existing standard for model documentation and is looking to the model developer to make a recommendation on the most appropriate level of source code documentation in which they should invest. The customer is not all that familiar with the field of system dynamics (almost a bit skeptical), but would like to publish the model on the company's Intranet as part of a manager education program. In addition, the customer has a concerns about the turnover rate and likely future re-organization of the management. As in every real-life situation, the customer has a tight budget, with needs that outweigh the available funds.

A relatively extensive source code documentation package would be appropriate in this example. Issues include the multiple uses of the model (it would be used to model several plants), the turnover and inexperience of the model users, and the high cost associated with model failure (not being able to predict a plant shutdown).

CONCLUSIONS

Source code documentation decisions should be simple rather than complex. However, making the best decision involves consideration of several key pieces of information about the nature of the model and the environment in which it will be used. This information involves the model purpose, the frequency and duration of model usage, the profile of model users, the potential risks of misusing the model, and the cost of preparing documentation. Once this information is evaluated, an informed documentation decision can be more effectively made. A range of source code documentation options are available to the modeler, including materials that can be automatically prepared by the modeling software, interactive materials incorporated into the user interface, and extensive manually prepared documentation. These materials can address the line-by-line modeling code, information on variables and nodes, as well as the high-level architecture of the model.

References

System Dynamics Modelling, A Practical Approach. R. G. Coyle. 1996.