CUTTER IT JOURNAL

Finding a Common Language for Global Software Projects

by Alexandre G. Rodrigues

The rapid globalization of the world economy is forcing companies to focus their business on international market segments. In order to reach their global target groups, many projects are now implemented and managed by international teams, thus gathering people from different cultural backgrounds. Effective cooperation among these teams is vital to success. In the software industry, technical teams in the US and European Union (EU) are increasingly incorporating programmers from other parts of the world, often through geographically dispersed groups. Technology has been helping organizations to overcome communication barriers through multimedia-based systems for cooperative work. However, at the management level, cultural differences are probably the more significant influence — language, life rhythms, perception of time urgency, values, and codes of conduct all may differ.

Project management is primarily based on effective communication and negotiation among the parties involved in a project: the client, the contractor, the subcontractors, and the various technical and management subteams. Misinterpretation of cultural differences can easily jeopardize mutual trust, which is essential for cooperation. However, procedures to cope with this problem on a more systematic basis have not yet been developed. A "culturally independent" common language for managers to share their personal views would be most valuable. System dynamics (SD) business modeling can help fill this gap. An SD model represents how managers perceive their business processes, through intuitive cause-and-effect diagrams. Once simulated, numeric results and dynamic patterns of behavior provide insight and forecasts. These models capture important dynamic issues such as feedback, human perceptions, and time delays. In this article, I will describe how SD modeling can help organizations achieve effective communication and negotiation within a global software project environment. I will also discuss a practical application of SD modeling in a large-scale international software project.

GLOBAL ENVIRONMENTS

Managing a software project is a great challenge: persistent failures suggest that the IT industry has not achieved a mature and effective management framework. Managing a global software project can be even more difficult. A global operating environment adds a variety of new issues that need to be handled effectively.

Global projects can take various forms: a "joint venture" project generally comprises a multinational consortium of partners; some projects may have only one contractor organization but the work is divided and outsourced to various subcontractors of different nationalities; in other projects, a stable and single-nationality project team may need to handle an international client; and in other cases, a project team may have its technical staff divided into multinational and geographically dispersed groups. Whatever the scenario, there are issues that make it difficult to observe the following project management "best practices":

- Keeping a project orientation
- Building a project team culture
- Keeping a customer orientation

- Maintaining effective and transparent *communication*

In multinational consortium projects, the different partners often have different objectives, both strategic and operational. For one partner, the more important objective may be to penetrate a new market, while for another partner, achieving high profits is the primary motivation. Another problem has to do with the differences in the hardware and software platforms used by geographically dispersed groups. System integration can raise serious technical challenges, yet changing the platforms requires time and money — two rare commodities in software projects. Perhaps the more critical (yet subtle) issue is the cultural differences. Cultures establish norms through which people communicate. A great deal of what happens in a software project is just about communication.

THE CHALLENGE OF COMMUNICATION

Projects almost always bring people together for the first time. Poor communication is one of the major causes of management problems in projects. Global software projects are faced with this tremendous challenge: ensuring transparent communication. While a simple concept in words, it may be very difficult to achieve in multinational groups. Problems stemming from poor communication typically have knock-on effects, and their causes are difficult to trace back.

In general, projects are characterized by intensive and difficult communication. There are many communication channels that are necessary to maintain in order to keep the project underway. Difficulties then stem from the unfamiliarity among the parties involved, from the differences in their objectives (often in conflict), and from cultural differences. The key players involved in a project are usually the contractor, the client, the technical and management subgroups, the subcontractors, and eventually the government(s), which may impose certain legislation over the contractor's work. In a global project, not all but many of the communication channels established between these players occur among parties of different nationalities. The critical issue about communication is that it is the basis of an ongoing *negotiation* process. Poor communication may prevent productive agreements that would otherwise be easy to achieve. In other cases, it may lead to situations where two parties both think they have negotiated effectively, but their understanding of the agreement can differ substantially.

Let us take the example of the client-contractor relationship. We know that the client's trust in the contractor is a key factor in project success: a client who suspects the contractor's competence is likely to take actions that have adverse impacts on the contractor's productivity and work quality. Typical examples are the reluctance to delay milestones (even when that would actually benefit the project) and a high demand for progress reports. In certain software projects, negotiation of scope changes with the client can be the single most important management policy. Requirements "errors" are persistently reported as being very expensive and the major cause of cost overruns. Estimating their full impacts is very difficult due to the indirect knock-on effects. Providing the client with convincing arguments to agree on realistic estimates becomes very difficult.

Another good example is the relationship between the management and technical teams, which is critical to internal performance. In a global software project, these teams can be multinational and geographically dispersed. Here, multimedia-

based systems for cooperative work hold the promise of bringing everyone together [4]. But the cultural barriers and the differences between the technical and the management perspectives still remain: different ways of working, different prioritization of the tasks, and different perceptions of what indicates progress may all threaten effective communication.

There are many interesting examples of differences in cultural dimensions [8]. In some cultures, people like to "atomize" time: they like to do many things at once, and having constant social contact with others is very important for them. Considerable delays in keeping appointments can be a consequence of these preferences. Interestingly, these delays work as time pressure for such people to keep up their energy and motivation. In other cultures, by contrast, people prefer to divide and organize time. Tasks are carried out one at a time and in a more isolated environment. In such cultures, delays can be seen as lack of organization and incompetence. Some cultures have a "strong context"; that is, people like to be continually well informed about everything throughout their networks. In other cultures, people have a weaker context and select just what is important for a specific task. They will seek additional information to understand things further only if necessary.

If one considers the impacts of these differences on two parties trying to negotiate, one can imagine how valuable it would be to have a "culturally independent" language.

MODELING AS A COMMON LANGUAGE

Modeling is about representing reality in a simplified and organized manner. A model makes it easier for us to understand what reality is and how it works. There are several types of models, each taking a different perspective about the world and using its own representational language. Many see modeling as a technical task, but this reality has been changing over the last decade, in great part because of computers and user-friendly software. In the business world, models have been emerging as a very powerful way of helping managers to understand their problems better. Many business consulting companies are increasingly using computer models to help their clients.

Why is business modeling so valuable? We all have our mental models of problems. However, personal mental models tend to be inconsistent, incomplete, and unstructured. But we only realize that when we are forced to represent them explicitly. And that's what we do when we develop a formal model. A model helps us to find out "what we really think" and let others know about that. When developed in a group environment, a model allows different people with different perspectives to share their personal views and develop a common one to which they all agree — this common view becomes the model itself. In this way, modeling brings *consensus*. It shifts the focus of a discussion from the individuals to the model, an object over which everyone has a sense of ownership. Practical experience shows that developing business models with a group of managers enhances communication within a team. A final advantage of a model is that it may have the ability to answer specific questions.

As I mentioned earlier, there are various types of models. System dynamics [3] is a very powerful modeling approach. SD modeling is "process oriented" (i.e., focused on "how things work") and has various powerful features:

- It provides a dynamic perspective of how things evolve over time (e.g., how productivity varies throughout the life cycle of a software project).

- It captures the time delays so important in business systems (e.g., gaps between decisions and their impacts).
- It captures explicitly the soft factors of human nature, which often dominate software development (e.g., the impact of staff experience on defect generation rates).
- It provides a cause-and-effect analysis, which is very powerful in diagnosing problems (e.g., the real cause of a problem is often indirect).
- It allows causes to be traced through cause-and-effect chains.
- SD models can be simulated, producing dynamic patterns of behavior and numerical estimates (e.g., what is likely to be the final project cost).

There have been a number of applications of SD models to projects, some of which are in the software industry [6]. We will now turn our attention to a practical application of SD modeling in a real project.

SD MODELING IN ACTION

In the course of the project discussed below, I developed a system dynamics–based project management integrated methodology (SYDPIM). It provides a method for developing and validating an SD project model for any specific software project and formally integrates its usage with the PERT/CPM networks. The example I present here took place in the course of this research. For reasons of confidentiality, the numbers are disguised and the scenarios are hypothetical, though close to reality. The structure and calibration of the SD model are based on real data.

The Software Project

The prime contractor for this software-intensive, large-scale military project was BAeSEMA Ltd. (UK), now part of BAe Defence Systems. The project was aimed at developing a destroyer command-and-fire-control system for a Far Eastern customer. The system components were being developed by various subcontractors from Europe and a Far Eastern country. The software system was being developed mostly in-house and was planned for two overlapping increments scheduled for 32 months in total. Transfer of "know-how" to the client was part of the contract, thus implying close client involvement, including participation in technical development.

The software technical team was international, including European, Far Eastern, and Indian programmers. Effective communication with the client and within the technical development team was crucial. Of particular relevance was the subjectivity involved in interpreting the several contractual agreements, a problem that could be exacerbated by cultural differences. In order to negotiate effectively, the contractor had to be able to explain to the client the full consequences of scope changes and to quantify those impacts. The contractor also had to ensure the transfer of know-how without disrupting the project. Understanding and quantifying the impacts of involving less-experienced programmers from the client organization proved most valuable to the management team, and the project was completed successfully and on schedule.

Assessing the Impacts of Client Changes

If the client were to ask for scope changes, what would be the full impacts on the project outcome, and why? How could the contractor explain to the client the full extent of these impacts? Both numerical estimates and a clear explanation were important. SD modeling could help answer both of these questions.

The problem with scope changes halfway through development is not so much the direct impacts but the knock-on effects. Because these are indirect, they are more difficult to identify and estimate. Another problem is that scope changes rarely consist of a single, uncontrolled event caused by the client — although the contractor often assumes so. Instead, they result from an ongoing relationship between the client and the contractor.

One of the techniques used in SD modeling is the development of causal maps, also called influence diagrams. These diagrams identify all the relevant factors involved in a certain dynamic phenomenon and link them through cause-and-effect relationships. These links generate closed feedback loops, which in the end are responsible for the dynamic phenomenon.

Let us consider the example of client changes. First the client requests changes, then the contractor perceives that extra time will be required. Rarely does the client agree to the full amount of extra time really needed. The client's perspective is that, while some extra time may be acceptable, it is the contractor's obligation to cope with the problem. The contractor will simply have to handle the remaining schedule pressure by increasing staff productivity and the overall project work rate (e.g., requiring overtime, doing less QA).

This view of the problem is described in the causal map shown in Figure 1. Two loops can be identified in the cause-and-effect chains: the first loop, L1, shows how part of the schedule slippage is handled by negotiating a schedule extension with the client; the second loop, L2, shows how the remaining slippage is handled through pressures to increase productivity. The individual relationships in the influence diagram can be positive, meaning that the factors change in the same direction (e.g., the higher the productivity, the higher the work rate), or they can be negative, meaning that the factors change in the opposite direction (e.g., the higher the work rate, the lower the perceived schedule slippage). The negative relationships are identified here with an "O" (meaning "opposite"). All the other relationships are positive. Feedback loops with an odd number of "Os" are called negative, or balancing, loops (here identified with a "B-"). This is because they generally show a phenomenon where a problem generates

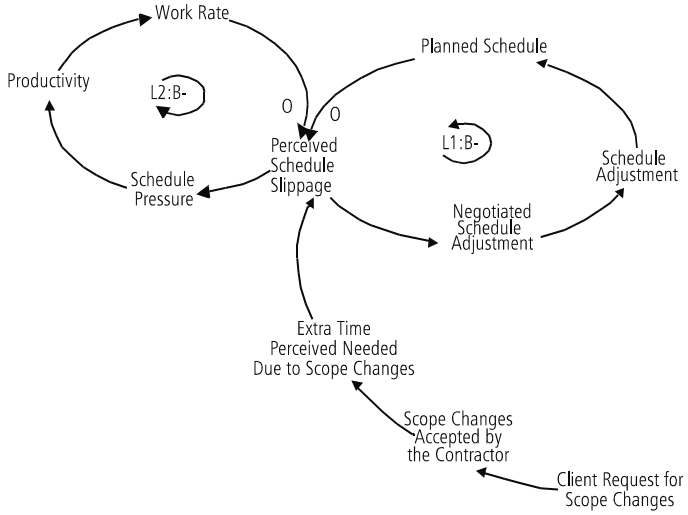

Figure 1: An SD causal map representing an incomplete view of the impacts of scope changes

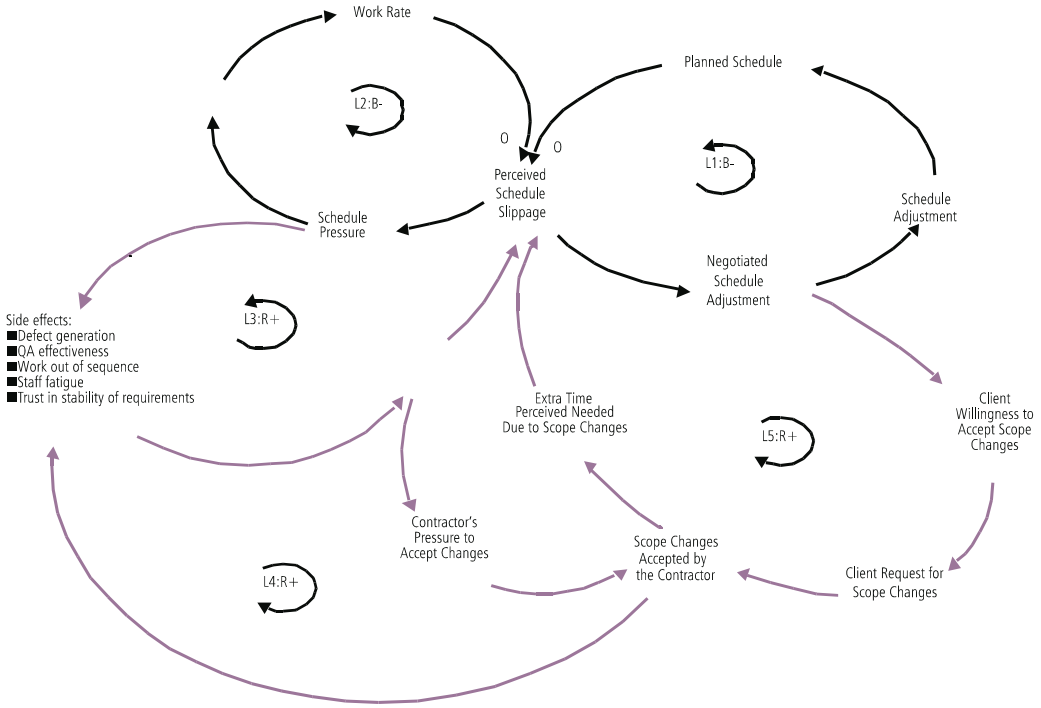

Figure 2: A complete view of the full impacts of client scope changes

actions that, over time, will attenuate that problem. The other loops are called positive, or reinforcing, loops (here identified with a "R+"), because they show dynamic events that reinforce themselves over time — like vicious circles.

The influence diagram in Figure 1 describes two true effects — neither the client nor the contractor can reject them. However, any contractor will know that this is an optimistic and incomplete view: scope changes are not just about extending the schedule a little bit and working faster. There are other serious effects:

- Scope changes typically require work to be done out of sequence, decreasing QA effectiveness and increasing defect generation. Later, rework will emerge and will cause delays. Generally, the client does not accept these delays, and if more time is to be negotiated, the contractor is now under pressure to accept further scope changes.
- As the client concedes schedule extensions to accommodate current scope changes, the more the client feels the right to ask for further changes in the future.

These effects were added to the causal map, as shown in Figure 2. Three more cause-and-effect loops can be identified. L3 shows that if the negotiated schedule extension is not long enough, excessive schedule pressure will cause negative side effects, which in the long run will delay the project even more. L4 shows that scope changes themselves may cause technical difficulties and adverse side effects, which will lead to delays. In order to get the extra time needed from the client, the contractor is now under pressure to accept further scope changes. L5 shows that as the client concedes schedule extensions, the more the client is likely to ask for further scope changes in the future.

All these loops are positive, or reinforcing, showing how problems can easily snowball

throughout the project life cycle, eventually aggravating one another. Each loop in the diagram can be seen as a "force" pushing the project outcome toward a certain direction. The final outcome will be the result of these combined forces. This is certainly a more complete and realistic view of the full impact of scope changes. The client will certainly agree that there are side effects, as shown in loops L3 and L4. The client has probably experienced situations where an aggressive posture toward the contractor has led to more and more scope changes, with the final outcome being serious overruns of schedule and cost. A caual map like this, agreed to by both parties, enhances communication and is an excellent means for negotiating scope changes.

Both parties are aware that if the right balance between a schedule extension and schedule pressure is not achieved, problems may escalate. The question now is, what is the *right* balance? In order to answer this question, one needs to move onto a quantitative analysis. In SD modeling, causal maps can be translated into simulation models that capture and quantify all the individual cause-and-effect relationships in great detail, through the use of equations. The model allows the user to test several scenarios until the desired outcome is achieved and the best solution is identified.

I will now present an example of this type of quantitative analysis using the SD model from our case-study project. The model is used to analyze the impacts of introducing scope changes during the later stages of the design phase of a certain system component.

In this case, the team knew that the requested changes could be accommodated with little or no schedule adjustment. However, while the cost impacts might be minor, what would be the impacts on the quality of the designs? A high number of undetected design defects could escape to the coding phase, and the problems in integration could then be disastrous. The question was, how much more schedule should be given to the design phase in order to accommodate the requested scope changes without damaging the quality of the designs too much?

In our example, a total of 30% of the initial requirements were to be changed, and the client started requesting these changes about 75% of the way into the design phase. Various policies for schedule adjustment were tested in the SD model. The results are shown in Figure 3, which presents the time-cost-quality impacts of different schedule adjustment policies. Each point on the x-axis shows a different policy:

- A 100% policy means that if 30% of scope changes are requested, then the schedule will also be extended by 30% (i.e., 100% of 30%) of the initial planned duration.
- A 50% policy means that the schedule will be extended by 15% (i.e., 50% of 30%).
- A 0% policy means that the schedule will be kept as originally planned.

The y-axis shows the percentage variations in cost, schedule, and defects escaped (i.e., quality of the designs) against the scenario where no scope changes were requested (called the "base case"). Each point in the graph refers to an experimental run in the model where the respective scheduling policy was tested. The results show that keeping the original schedule eventually leads to schedule and cost overruns of 18%, but at the expense of a very low design quality: about 75% more design defects escape to coding. Beyond a 20% schedule adjustment policy, considerable gains in quality start being achieved at some expense of cost and schedule. Beyond a 60% policy, the marginal gains in quality

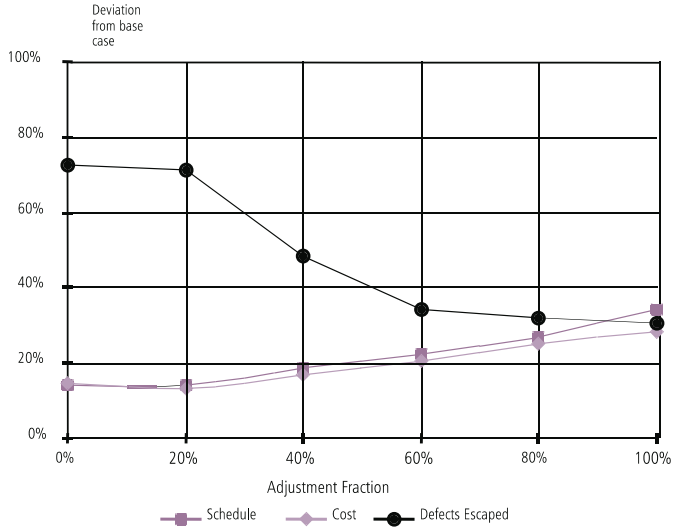

Figure 3: Impacts of different schedule adjustment policies

become minor and might not outweigh the increasing schedule and cost overruns. Therefore, this analysis suggests that the schedule should be extended by 18% (i.e., 60% of 30%) of the original duration. Figure 4 shows some characteristics of the dynamic behavior of the design phase for the two experiments of: (1) no schedule adjustment, and (2) a 60% schedule adjustment policy (see Figures 3 and 4).

With this quantitative analysis, both contractor and client know that a schedule adjustment less than 6% (i.e., 20% of 30%) will cause an unacceptable low quality of the designs. The contractor knows that asking the client for more than an 18% schedule extension is not worth it and is only likely to encourage further requests for changes. Both parties also know that, even with the best solution (i.e., a 60% policy), the design phase will inevitably cost more and last longer (about 20%), while the number of defects escaping to coding is likely to increase 38%. A more detailed description of these results and how the SD project model was developed and validated can be found in [7].

The value of an SD analysis like this is two-fold: (1) it provides numerical estimates, which are useful in assessing the impacts and developing mitigating actions, and (2) since both parties are involved in developing the causal map and both provide input to quantify the simulation model, they both have a sense of ownership of the results. This creates commitment and helps in reaching consensus. As a business modeling technique, system dynamics used in this way can play the important role of enhancing communication and negotiation between parties that may have different objectives and possibly different managerial and even national cultural backgrounds. What could be a fierce dispute for concessions may become a joint modeling effort aimed at finding the best solution for the project.

HOW CAN WE USE SD MODELING IN GLOBAL SOFTWARE PROJECTS?

Global software projects include multinational parties. Effective communication and negotiation among these parties is critical to success. Yet the potential barriers are numerous, and difficulties will always emerge. The key is not to hope to prevent all these difficulties but to handle them

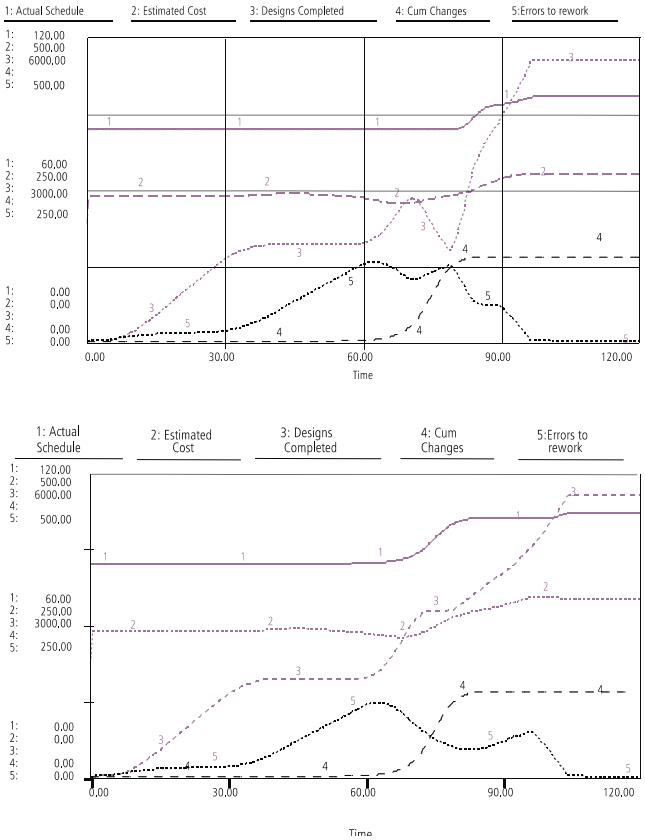

Figure 4: Dynamic behavior of the design phase produced by the model

effectively when they arise. This can only be achieved by bringing the views of the different parties together and finding solutions on this basis, while making everyone understand why — a good solution without support from all parties is doomed to failure. Everyone's point of view must be considered, and it has to be everyone's solution for the benefit of the project.

In this article, I have argued that business modeling can play an important role in facilitating effective communication and negotiation among the parties in a global software project. The various ways in which different people understand a problem can be represented as a common view, and solutions can be identified more effectively. But how should an organization involved in a global software project approach the use of SD modeling in this way?

Well, first there is some basic learning required about the methodology. Like any modeling technique, SD has its own technical side, but this does not put it out of reach. The development of causal maps or influence diagrams will require some familiarization with the feedback perspective of the world. Fortunately, most of us understand the term "vicious circle" (positive/reinforcing feedback) and know how a thermostat controls the temperature of a room (negative/balancing feedback). Causal maps are the qualitative side of SD. Problems can be diagnosed and solutions identified at this level. Using the causal map technique is a good first step. The quantification of the causal maps into simulation models requires the use of specialized software packages (e.g., iThink/Stella, Powersim, Vensim). Quantification also requires more technical

Alexandre Rodrigues is a lecturer at the University of Minho, where he also works as a consultant specializing in the use of system dynamics modeling in software development and project management issues. Mr. Rodrigues has previously worked as a software analyst and pursued the NATO Ph.D. program at the University of Strathclyde, UK. His research focused on the application of system dynamics to software project management, and he has published various articles on this topic. Mr. Rodrigues later worked as a consultant at the Pugh-Roberts Associates practice of PA Consulting Group, where he also applied part of his research to the development of software products. He received a licentiate degree in systems and computing engineering from the University of Minho, Portugal.

Mr. Rodrigues can be reached at the Department of Information Systems, The School of Engineering, University of Minho, 4800 Guimarães, Portugal. Tel.: +351 53 510 149; Fax: +351 53 510 250; E-mail: Alex.Rodrigues@dsi.uminho.pt; Web site: www.dsi.uminho.pt/~alex/

knowledge and some level of familiarity with the principles of simulation. However, the software packages available are user friendly, and the modeling language is presented in a visual fashion, making it easier for the beginner. A good practical introduction to SD modeling can be found in [2] and also in [5].

Secondly, it is important to decide who will build the model and what is the development method to be followed. My advice is that the individuals in charge of this work should belong to the software process improvement group, which usually is also in charge of metrics. Given the nature of this group, it will be easier for these individuals to grow and maintain the required technical knowledge. Furthermore, an SD project model requires metrics for calibration and is also an excellent tool for assessing the impacts of software process improvement initiatives. Regarding the model development method, there is, unfortunately, no formal method available yet that ensures the development of the right project model. For the inexperienced modeler, the development, calibration, and validation of the model will require some expert guidance.

Finally, an organization will achieve higher returns from its first effort by using the model in various projects. In order to reuse a model, my two main pieces of advice are: (1) develop and maintain a database of metrics as the model is used and calibrated to past projects, and (2) build the model as an assembly of generic substructures and maintain these in a database. Different projects will require models with different architectural structures. It will be easier and quicker to develop new models by reusing and assembling these generic substructures — you can call this the "modeling version" of the OO/component development approach to software development. A final piece of advice: start simple; get used to causal maps first, and then try to derive simple models from them. Just these small steps are likely provide you with valuable insights about your software project. Move onto more complex models after you get comfortable with the methodology. And remember: the model is there to help *you* take decisions and *not* to take the decisions for you.

REFERENCES

1. Association for Project Management. "IT Project Failure." *Project*, Vol. 11, No. 6 (November 1998), pp. 5.

2. Coyle, R.G. *System Dynamics Modeling: A Practical Approach*. Chapman & Hall, 1996.

3. Forrester, J. *Industrial Dynamics*. MIT Press, 1961.

4. Marcos, A. "Modelling Cooperative Multimedia Support for Software Development and Stand-alone Environments" (Ph.D. diss., University of Darmstadt, Germany, 1998).

5. Richardson, G.P., and A.L. Pugh. *Introduction to System Dynamics Modeling with Dynamo*. MIT Press, 1981.

6. Rodrigues, A.G., and J.A. Bowers. "The Role of System Dynamics in Project Management." *International Journal of Project Management*, Vol. 14, No. 4 (August 1996), pp. 213-220.

7. Rodrigues, A.G., and T. Williams. "System Dynamics in Project Management: Assessing the Impacts of Client Behaviour on Project Performance." *Journal of the Operational Research Society*, Vol. 49, No. 1 (January 1998), pp. 2-15.

8. Schneider, A. "Project Management in International Teams: Instruments for Improving Cooperation." *International Journal of Project Management*, Vol. 13, No. 4 (August 1995), pp. 247-251.

