
METHODS & TOOLS

Practical knowledge source for software development professionals

ISSN 1023-4918

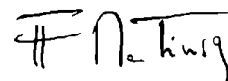
Spring 1999 (Volume 7 - number 1)

Real World, Real Issues, Real People

The first article of this issue is the kind of knowledge Methods & Tools is looking for. Mike Lee provides practical experience. As he says, "Practice, as opposed to theory, is typically neither pretty nor clean". Many texts on software development are written from a theoretical perspective, but there is less material on the "ugly and dirty" points and how to deal with them. You will be able to apply the knowledge contained in this article outside the domain of object oriented analysis to many aspects of the software development management area. There are no fixed rules to produce the perfect project or software, without thinking first. This is also one of the reason why Booch & Friends switched from the Unified Method to the Unified Modeling Language before reaching version 1.0 of their effort. We can only track the advancement of the activities and try to make the best decisions taking into account not only technical factors but also the human dimension of software development.

The human factor is the element I would like to put in evidence from the Mike Lee paper. As he declares, "Successful software engineering requires a number of very different technical skills. [...] Very few individuals are equally talented in all of these areas." So you must identify the strengths and weaknesses of the member of the IS staff. These elements have then to be considered when project teams are built. As we are currently talking about theory and practice, this is naturally easier said than done! Usually only experiments can show how somebody performs a task. It is then important to have some small projects where you can "test" people. They can obviously not be "tested" without having a chance to achieve positive results. Sadly, most of the techniques needed for development activities outside coding are not learned in computer curriculum. IS staff should have the opportunity to be trained in aspects that are very important for software development like negotiation, communication, presentation and meeting management. You should not forget that IS people are mostly introverts (3/4 of the developers are introverts versus only 1/3 of the global population), so they need more training on the communication aspects of their work. Assigning people to activities where they perform the best is then a matter of psychology. The more important thing is not to prevent people to perform some tasks (because you need multi-talented people and teams), but to be sure that you can put your best people for tasks that are critical for the success of important projects.

I hope that Mike Lee experience will help your decision making process and that you will enjoy reading his text, and the other contributions of this issue, as much as I did.



METHODS & TOOLS is published by **Martinig & Associates**, Avenue Nestlé 28, CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch ISSN 1023-4918
Editor: Franco Martinig; e-mail franco@martinig.ch
Free subscription form can be found on www.martinig.ch
Advertisements contact: franco@martinig.ch
The content of this publication cannot be reproduced without prior written consent of the publisher
Copyright © 1999, Martinig & Associates

Object-Oriented Analysis in the Real World

Michael M. Lee, Project Technology Inc., <http://www.projtech.com>
2560 Ninth Street - Suite 214, Berkeley, CA 94710

1. Introduction

Theory and Practice

There is theory, and there is practice. There is the plan, and there is its execution. This paper concerns itself with the practice and execution of using Shlaer-Mellor object-oriented analysis (OOA) [1, 2] on real-time control systems. The task of the project manager is to make the theory and the plan work together in the context of a given organization, project, and engineering team. This paper describes experiences in this world and presents an approach, evolved over time and with the help of other colleagues working in this area, for succeeding with this task.

The Projects

The experiences described in this paper were gained on five separate projects. They involved medium to large real-time systems. The organizations typically had little or no prior experience with formal analysis methods. The motivations for using a more formal approach included efforts to

- reuse all or significant parts of the system,
- manage the size and complexity of the system,
- develop a long-term, product platform and
- improve the software development process.

The first two projects were on distributed minicomputer platforms. The last three were on embedded, distributed, microcomputer platforms. The following table provides a summary of these projects.

Object-Oriented Analysis

Shlaer-Mellor object-oriented analysis uses an integrated set of models to identify the conceptual entities (objects) in the system, their dynamic behavior, and the required processing. The reasons for using this approach include

- its explicit attention to real-time issues,
- the ability to verify the analysis by simulating the models,
- the availability of textbooks, training and technical support for it, and
- its prescribed approach for transitioning from analysis to design.

The reader is encouraged to read the first two references for more details on this approach.

2. The Real World

Practice, as opposed to theory, is typically neither pretty nor clean. In this section, those aspects of projects which *are* sometimes ugly and dirty are considered. The issue here is not to point a self-righteous finger at these areas, but to recognize that they do exist and will have a negative impact on the project if not effectively addressed. The areas in which OOA can offer assistance are noted.

Requirements

The problems that arise when requirements for a software system are poorly understood are widely recognized and well documented in a number of books and articles [3, 4, 5]. Still, the problems persist, and, in my estimation, will never totally disappear. Here are a few reasons for this.

- *Pushing the Technology Envelope.* When systems are expanding into areas of new technology, the "requirements" are frequently being discovered as development proceeds.
- *Building Product Platforms.* When it is the objective of a project to develop a platform to support a family of products over an extended period of time, seldom are all the members of this family defined yet. Many are little more than a glimmer in some marketing VP's eye. In situations like this, requirements are never well understood at the beginning, nor do they remain static over time.

Business	Application	Size (SW staff)	Duration	Result
Aluminum Manufacturing	Rolling Mills	14	3.5 years	Factory Installation
Communications	Material Handling	12	2.5 years	Operational Prototype
Medical Instruments	Imaging	22	5.0 years	In Progress (design/code)
Computer Peripherals	Tape System	15	3.0 years	In Progress (design/code)
Computer Peripherals	Disk Array System	95	4.0 years	In Progress (analysis/design)

Table 1. Projects concerned

What is needed is an analysis method that allows the engineer to quantify, in some manner, the tradeoffs between different sets of requirements, and to assess the ability to meet multiple sets of requirements. The OOA models, through their ability to simulate execution, provide excellent support for this.

Schedules

Producing relevant schedules for software development is even more problematic than doing the development itself. This is especially true for the initial analysis phase when we don't know what we don't know yet. Add to that a learning curve of uncertain slope for a new analysis method and it truly becomes a Herculean task. No analysis method, it would seem can offer the clairvoyant assistance needed here. What we can ask of a good method, however, is to provide relevant feedback for adjusting and calibrating the schedule. Suggestions for how to do this with OOA are described in the next section.

Size/Complexity

Of issue here is not that systems are large or complex, but that their true size and complexity are often poorly understood. As long as this is the case, schedule projections suffer and a mild sense of being out of control persists. It is in the analysis phase that the size/complexity question must be answered, and the analysis method should have two important characteristics to deal with this. First is the ability to scale up for large and complex problems if necessary. Second is the ability to factor and reduce the size and complexity to their smallest possible dimensions. The Domain Chart of OOA is a valuable tool for doing this and its use is described in the next section.

Engineering Skills

Successful software engineering requires a number of very different technical skills. These include the ability to do analysis, system design, program design, coding, integration, and testing. Very few individuals are equally talented in all of these areas.

My informal sampling of software engineers produces a distribution that peaks in the middle with program design and coding, and falls off in both directions with good analysts and testers being the most rare. This creates two challenges during the analysis phase. One, good analysts must be identified and assigned judiciously. Two, productive work must be found for members with other skills. OOA offers some help in this area by having different roles within the analysis phase.

The design approach associated with OOA (Recursive Design [6]) also offers opportunities to begin some design and implementation tasks early in some areas, often while analysis is still proceeding in other areas.

Silver Bullet Syndrome

Perhaps the most insidious real-world problem when introducing new software methods is the over zealous and unrealistic expectation that the next new method (object-oriented anything, these days) will prove to be the long awaited "silver bullet" with which the dreaded software development dragon can be slayed.

It is insidious because it carries enthusiasm that is short lived and a commitment that is only slightly longer lived. Addressing this problem is outside the scope of *any* particular method, though it must be done if a method is to be assessed on its true merits. The best policy here is complete frankness on the benefits and liabilities of a method, and an ability to adjust for these.

3. Practical Approach to Applying OOA

Analysis Steps

Halfway through the third OOA project, when some of the difficulties encountered began to feel familiar, an attempt was made to identify and categorize these difficulties. (Note that the difficulties referred to here are not with the method, but those arising from "making the theory and plan work together in the context of a given organization..." as described in the first section).

The strongest pattern that arose when categorizing these difficulties was that of time and sequence. Different problems came up at different times in the analysis, and different problems were dealt with more or less effectively depending on the sequence in which they were resolved. The best way to address this was to partition the analysis process into a set of distinct steps based on the sequence in which the issues were best addressed. This way each step builds on the work of the preceding steps. These steps are to:

- assess applicability of OOA,
- gain management support,
- initiate analysis, and
- sustain analysis.



■ ■ ■ ■ ■ ■ ■ ■

BUILDING THE FUTURE

Do you want to develop smarter and more efficient applications? TBUILDER is a complete visual modeling tool supporting the UML notation and extensions.

TBUILDER FEATURES



- Modeling of major UML diagrams (includes Activity Diagram)
- Consolidation management in the repository
- Automated document generation
- Management of the modeling standards
- Administration of components
- Code generation for 3GL, 4GL, and Datawarehouse
- Automatic generation of client/server applications
- Automatic prototyping by interpreting class diagrams
- Re-use of objects

CONCIS
TECHNOLOGIES

CONCIS Technologies, 37bis rue du Prébeard 95814 Argenteuil Cedex - France
Tel: 33 (0)1 39 98 46 00 Fax: 33 (0)1 34 10 88 86 Email: concis@concis.com

Concis is an ARIANE II company
Visit our web site www.concis.com

This simple sequence of steps performs the age-old function of breaking a problem down into smaller pieces to reduce the difficulty, and to guide the process. The following subsections address the issues and proposed approach for each of these steps.

Assess Applicability of OOA

As basic as this step may seem, it is seldom done in as thorough a manner as it should be before the other steps are started. This can cause difficulties in the later steps when it continues to be assessed, multiple times, by different groups, possibly with different results! The basic assessment which must be made is the *match between the projects objectives and the method's capabilities*.

There are three basic areas that need to be considered in making this assessment

- *Technical Considerations*. Is OOA the right tool for the job? Does it address the difficult issues that the project expects to encounter? Does it help meet the project's technical objectives?
- *Management Considerations*. Can the project lifecycle be "front loaded"? Will there be support for training? Is there a perceived need for this technology? Does it help meet the project's business objectives?
- *Staff Considerations*. Is there an existing analysis method? Does it work? Is there a perceived need for this technology? Does it help the staff meet their professional objectives?

When assessing these issues, one needs to be aware of project objectives that may and may not be addressed by OOA. Examples of project characteristics that can be addressed extremely well by using OOA include analyzing large and/or complex systems; developing configurable, data-

driven systems; preparing for object-oriented design and programming; designing for reuse and maintenance; and automating the implementation.

Examples of project characteristics that may not be addressed particularly well by using OOA include the "n-th iteration" on a well understood problem; a project which is already absorbing a number of other new and different technologies; projects with extremely aggressive schedules; and systems where "quick and dirty" is an acceptable approach.

Meilir Page-Jones has recently compiled an interesting list of motivations for taking, and not taking, an object-oriented approach which expands on this topic [7].

Gain Management Support

Having determined that there is a good match between the project's objectives and the capabilities of OOA, the next step is to gain management support for the introduction of this new analysis method. The support that will be needed will include funding for such things as training and CASE tools, commitment to a schedule that will spend more time in analysis and design and correspondingly less in implementation, organizational changes to accommodate the different skill profiles required for this work, and cultural changes in the way work gets documented and reviewed.

It's important to gain this support based on *realistic expectations* of how the work will proceed. It will do more harm than good to gain support based on unrealistic expectations like: "it's object-oriented, so we'll see a 10 fold increase in productivity immediately", or "the learning curve impact is just the training time". Such support is very likely to dissipate quickly as these unrealistic expectations are not met, and the project will suffer accordingly.

At the heart of gaining management support is convincing the appropriate managers that there is a good match between the project's objectives and OOA's capabilities. The following tasks are key for doing this.

Present Assessment of OOA's Applicability. Having established that there is a good match between the project's objectives and OOA's capabilities, this must be communicated to the management team. They should have an opportunity to probe the assessment, have their questions answered and hopefully come to the same conclusion.

Demonstrate Benefits of Good Match. It's important at this point to recognize that you are involved in making a *sale*. The best way to succeed with this is to go beyond just establishing the good match and elaborate on other benefits to be gained on future projects when OOA, and hopefully a significant amounts of code, can be reused.

Establish Realistic Expectations. It's extremely important at this point, for reasons cited earlier, to ensure that the above benefits are presented in a realistic light, and that the costs acknowledged. Typical costs are training, learning curve, CASE tools, and possibly technical support. Typical changes in "business as usual" are some organizational shuffling, new documentation and review procedures, and more time spent at the front end of the project.

Present a Plan. A good management team will not commit to a process without at least a preliminary plan of execution. Be prepared to provide this. Base it on the prescribed approach of constructing a Domain Chart, Subsystem Diagrams, and Project Matrix. More will be said in later subsections about how to manage this planning.

Initiate Analysis

The task at this step is to overcome the inertia of initiating activity and get the team moving in a coherent manner in roughly the same direction. To do this, the following tasks must be accomplished.

- Create a productive analysis environment.

- Train the technical and managerial staff.
- Create an executable, short-term plan.
- Initiate the analysis activities.

Productive Analysis Environment. The software profession has made great strides over the past few decades in improving the implementation environment. The programmers' workbench, coding standards, configuration management systems, and interactive debugging environments are a few good examples of this.

By comparison, analysis environments are typically primitive, not that they need be, however. The tools do exist. It just seems that this receives little attention. If your plan is to spend a significant part of your schedule in this phase, it behooves you to give this more consideration. The essentials for a productive environment include the following.

- *Document Library.* Establish a central, public repository for project documentation. This is *the* deliverable during the analysis phase. It's important to manage it and ensure its common and convenient accessibility. This is most conveniently done "on line", though a manual system can also work.
- *Document Standards.* It's a small thing, keep it simple, but do it! Designate a document and drawing tool. Ensure that title, author, version, and date information are uniformly included on every document.
- *Meeting Guidelines.* Unproductive meetings place unnecessarily large overheads on the analysis process. For productive meetings, clarify objectives, limit allocated time, and stick to agendas. Distinguish between the following three types of meetings. A working meeting for getting technical work done. A presentation meeting for informal communication of the results of a piece of work. A review meeting for formally reviewing a work product.
- *CASE Tool.* OOA generates a set of integrated models which all have graphic representations. This is not something that can productively be done manually or with a simple drawing tool. The minimum requirement is a professional drafting tool and a draftsman. A more desirable alternative is a networked CASE tool to which all the analysts have access.

Train Staff. Three courses exist for OOA/RD and the team should attend them all. The best thing to do is to train the team in OOA as a group immediately before starting the analysis effort. Send both technical staff and their managers. The technical staff for obvious reasons. The managers so they understand the process they'll be attempting to schedule and control. An important point to understand about the training is what it can and cannot accomplish. One, it will not create analysts out of people without these skills. Two, it will only impart knowledge not experience. Three, it's folly not to do it.

Create a Plan. The plan created in the previous step of gaining management support was really a strategic statement outlining a general approach. Now a tactical statement is needed to clarify the week-to-week activities and focus the effort. The best guidance that can be offered is to initially target those areas of the application which are expected to provide additional insights. This may mean different things on different projects. Sometimes doing a well-understood area to gain experience with OOA method helps. Sometimes doing a particularly difficult area to demonstrate the capability of the OOA method helps.

Sometimes the structure of the problem suggests an advantageous starting point. The important point is to target your efforts for a defined objective as a means of providing a focus for your activities. Taking an undirected, breadth first approach to the problem is very likely to stall out without producing anything of significance.

Initiate Analysis. Two things are important as you begin the analysis activities. One is organizing the analysis teams, and the other is recognizing that the analysis process is frequently akin to prospecting.

When organizing the analysis team, both size and skill are critical. Small teams of two to four people seem to work best. At least two people are needed to provide second opinions, and more than four tend to either leave some members out of the process or slow down as the diversity of opinions are dealt with. The skill issue is both obvious and yet difficult to address. The obvious part is ensuring that each team has at least one skilled analyst. The difficult part is identifying the good analysts, and then keeping them evenly distributed among the different analysis areas.

A productive mind set as analysis begins is that of an oil prospector. It's impossible to accurately predict where the oil will be found. Rather than agonize over exactly where to drill, or continue to drill in areas which have not yielded anything, the best thing to do is make an educated guess, proceed with sinking some holes, and use the results to guide your future work.

From an OOA perspective, this means identifying potentially fruitful areas and proceeding to build the information models in these areas. The results of this analysis will help guide the direction of future work.

Sustain Analysis

As the different analysis teams begin to develop and review their OOA models, new issues can arise which affect the ability to successfully sustain the analysis effort. These include confusion with details of the OOA formalisms, OOA models that are only understood by their developers, models which can never be completed, confusion with what to model and what not to model, difficulties with integrating the models from different teams, the analysis plans becoming obsolete, and management support wavering as schedule pressures increase. A number of things can be done to address, and in many cases prevent, these problems.

- Augment the formal models with informal notes and diagrams.
- Review the models.
- Leverage available analysis experience.
- Distinguish between analysis and design issues.
- Maintain the domain chart and subsystem interfaces documents.
- Demonstrate tangible progress on a regular basis.
- Use results to calibrate and maintain the plan.

Informal Notes and Diagrams. The OOA formalisms (information model, state model, and process model) are an excellent means to model the conceptual entities and behavior of a system precisely. They are not always sufficient, however, for understanding the underlying abstractions upon which the models are based.

What we want to capture are the white board pictures and explanations that initially produced the "Yes, that's the way to think about this!" insights. What works best in situations like this is to produce an informal document that explains the thinking and underlying abstractions behind the models as a prelude to doing the formal models. Using diagrams and pictures in this type of document greatly enhances its ability to communicate complex ideas. With these informal documents (commonly called "technical notes"), it is possible to capture and communicate some of the most important work that goes on during the analysis phase.

It provides a good foundation for developing the OOA models. It makes them more comprehensible after they have been developed. It allows conceptual approaches to be assessed

before investing time in modeling them. And it provides a convenient mini-step that aids in tracking progress.

Review Models. A discipline of reviewing analysis models when they reach an acceptable level of maturity must be established on the project. This monitors quality, measures progress, provides interactions between different analysis teams, guards against unnecessary "polishing", and produces a sense of progress as tasks are completed. Each model should be reviewed as it is completed. For each subsystem, this means a review for the information model, state model set, object communication model, and process model set. For the system this means reviews for the domain chart and the subsystem information, communication and access models. Reviews can also be helpful prior to "shelving" a piece of work for some time, and when a piece of work is "stuck". These reviews should be formal assessments of the work product, documented with meeting minutes, and generally follow good software inspection procedures.

Leverage Available Analysis Experience. This experience typically comes from two sources. First, from other projects within the organization that have used OOA, and second from within the project, as some team members pick up the analysis technique more quickly than others. These individuals need to be identified and effectively assigned to the important analysis problems to leverage their skills. This requires a certain degree of flexibility in being able to assign and reassign team members to tasks. This flexibility can be problematic in certain organizations and with particular individuals. It helps if the dynamic nature of assignments during the analysis phase can be established early.

If the availability of in-house experience is insufficient, outside help from experienced OOA practitioners should be sought to bootstrap the project. This will significantly enhance the speed of the technology transfer and early self-sufficiency of the organization.

Distinguish between Analysis and Design. Analysis should focus on the application. Design should focus on the implementation. When implementation issues begin to creep into the analysis, the job gets bigger and the issues get cloudier. The result is that the analysis process can bog down. One needs to be on constant guard against this. The OOA courses are very explicit about this distinction, and the message needs to be reinforced throughout the analysis phase. The Recursive Design course can also help by clarifying how the analysis models are transformed in design.

Maintain Domain Chart and Subsystem Interfaces. On the one hand, the partitioning of the system into domains and subsystems is essential. On the other hand, it cannot effectively be done in one pass. Not enough is known at the start of the project to do this correctly. Given this dilemma, how should the project proceed? A best guess, based on good engineering judgment, must be made at the start of the project.

Thereafter, it is important to revisit this choice periodically and make adjustments as more is learned through the analysis. The process then becomes one of refining the system vision as more is learned about the individual components. One wants to avoid wasting time at the start of the project agonizing over the perfect partitioning. One also wants to ensure that the insights gained through analysis are factored back into the partitioning and an integrated system vision.

Demonstrate Tangible Progress. This should be done on a regular basis for two important reasons. First, it encourages continued management support for the project and its team. Very few management processes accept the Big Bang (results will suddenly appear overnight) theory of development and hence want to see incremental progress.

Second, it has a positive impact on the morale of the team to recognize their progress. It's easy for them to lose sight of this progress when the daily focus is on the work that remains to be done.

There are a number of analysis activities that can be used to demonstrate tangible progress: a revised domain chart which reflects a better understanding of the different subject matters in the system; new conceptual insights documented in technical notes; completion of reviews; successful subsystem partitioning of a domain allowing parallel work to proceed. The analysts should take the initiative on documenting and communicating these achievements. The best defense is a good offense!

Calibrate and Maintain the Plan. As the analysis effort proceeds, more is learned about the scope of the work and the rate at which it can be done. This information needs to be factored back into the analysis plan on a regular basis to understand its impact and allow for timely adjustments. The situation here is very analogous to creating and maintaining the domain chart. Good engineering judgment is used to initialize the plan and adjustments should be made as new information is available. Adjustments to scope should track changes to the domain chart. Adjustments to efficiency need to be made far enough apart to ensure significant data and close enough to recognize problems. My rule of thumb is to do it between every 3 to 6 months. The guiding principle is that, to be useful, the plan must reflect a reasonable approximation of what the team is capable.

4. Summary

There is a lot more to succeeding with OOA than mastering the modeling techniques. Some of it is just good project management practices. Some of it is just good analysis practices. Some of it has to do with the process of introducing a new engineering discipline into an organization. Some of it has to do with the general, object-oriented approach of OOA. And some of it has to do with application of the specific OOA modeling techniques. If the issues in all of these areas are actively tracked and addressed, the chances of success are greatly enhanced. The following checklist summarizes critical activities.

- Establish utility of OOA.
- Create productive analysis environment.
- Train the analysis team.
- Use experienced OOA analysts.
- Augment formal models with informal technical notes.
- Factor analysis results back into the maintenance of the domain chart and subsystem interfaces, and into the project plan.

References

- [1] Sally Shlaer, Stephen J. Mellor, "Object-Oriented Systems Analysis -- Modeling the World in Data", Prentice Hall, 1988.
- [2] Sally Shlaer, Stephen J. Mellor, "Object Lifecycles -- Modeling the World in States", Prentice Hall, 1991.
- [3] Tom DeMarco, "Structured Analysis and System Specification", Yourdon Press, 1978.
- [4] T.E. Bell, T.A. Thayer, "Software Requirements: Are They Really a Problem?", Proceedings, 2nd International Conference on Software Engineering, 1976.
- [5] Harlan D. Mills, "Software Engineering", IEEE Transactions on Software Engineering, Volume SE-2, No. 4, December 1976.
- [6] Sally Shlaer, Stephen J. Mellor, "Recursive Design", Computer Language, March 1990.
- [7] Meilir Page-Jones, "Object-Orientation: The importance of being earnest", Object Magazine, July-August, 1992.

Understanding the Unified Modeling Language (UML)

Sinan Si Alhir, salhir@earthlink.net, <http://home.earthlink.net/~salhir>.

Introduction

Organizations compete in a global market that is characterized by opportunities and risks where ongoing business and technological change fuel ever-increasing competition. Organizations must not only manage change and the complexity that results from adapting to change, but capitalize on the lessons learned, best practices, and knowledge gained through this evolutionary process. The importance and criticality of knowledge has given way to the Knowledge Revolution. This revolution can be characterized by the radical and fundamental paradigm shift that is occurring within the business and technology industries where an organization's application of knowledge defines its competitive advantage. Organizations can no longer only rely on physical muscle and size, but must harness intellectual capital and creativity to be successful. Knowledge takes on many forms; however, its value is demonstrated through its application and the realization of solutions to problems. If knowledge is captured and reapplied, an organization is enabled to become even more competitive and proactive, rather than reactive, to change and complexity; thus, increasing an organization's probability of success. This raises the fundamental question of how does an organization best capture, communicate, and leverage knowledge in order to gain a competitive advantage.

Historically, organizations have attempted various methods for procuring intellectual capital. Within the information system and technology industry, we have encountered the use of structured techniques to minimize the impacts of change and complexity, the use of Computer Assisted Software Engineering (CASE) tools to automate the development process, the use of business reengineering techniques to optimize organizational processes, the use of object-oriented techniques to facilitate reuse, the use of patterns to capture solutions to recurring problems, and the use of components to actualize reusable parts. Inherent to these techniques is the encapsulation of knowledge.

With the emergence of the Unified Modeling Language (UML) from Rational Software Corporation and the Object Management Group (OMG), it is very conceivable that such a language that unifies the many threads and incarnations of the Knowledge Revolution is the most viable means for organizations to best realize a competitive advantage via capturing, communicating, and leveraging knowledge. Rational Software Corporation and three of the most prominent methodologists in the information systems and technology industry, Grady Booch, James Rumbaugh, and Ivar Jacobson (the Three Amigos) originally conceived the UML. The UML emerged from the unification that occurred in the 1990s following the "method wars" of the 1970s and 1980s to gain significant industry support from various organizations via the UML Partners Consortium and be submitted to and adopted by the OMG as a standard (November 17, 1997).

The UML is a modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process.

- Within a system-intensive process, a method is applied as a process to derive or evolve a system.

- As a language, it is used for communication. That is, a means to capture knowledge (semantics) about a subject and express knowledge (syntax) regarding the subject for the purpose of communication. The subject is the system under discussion.
- As a modeling language, it focuses on understanding a subject via the formulation of a model of the subject (and its related context). The model embodies knowledge regarding the subject, and the appropriate application of this knowledge constitutes intelligence.
- Regarding unification, it unifies the information systems and technology industry's best engineering practices across types of systems (software and non-software), domains (business versus software), and life-cycle processes.
- As it applies to specifying systems, it can be used to communicate "what" is required of a system, and "how" a system may be realized or implemented.
- As it applies to visualizing systems, it can be used to visually depict a system before it is realized.
- As it applies to constructing systems, it can be used to guide the realization of a system similar to a "blueprint".
- As it applies to documenting systems, it can be used for capturing knowledge about a system throughout its life cycle.

The UML is not:

- A visual programming language, but a visual modeling language.
- A tool or repository specification, but a modeling language specification.
- A process, but enables processes.

The UML is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language.

- As a general-purpose modeling language, it focuses on a set of concepts for acquiring, sharing, and using knowledge coupled with extensibility mechanisms.
- As a broadly applicable modeling language, it may be applied to different types of systems (software and non-software), domains (business versus software), and methods or processes.
- As a tool-supported modeling language, tools are readily available to support the application of the language to specify, visualize, construct, and document systems.
- As an industry-standardized modeling language, it is not a proprietary and closed language but an open and fully extensible industry-recognized language.

The UML enables the capturing, communicating, and leveraging of strategic, tactical, and operational knowledge to facilitate increasing value by increasing quality, reducing costs, and reducing time-to-market while managing risks and being proactive in regard to ever-increasing change and complexity.

The Big Picture

To successfully leverage the UML, we must first understand the overall context in which the UML applies.

Problems, Solutions, and Problem Solving

Organizations produce and deliver products and services that address customer needs and requirements. Requirements may be characterized as problems (often referred to as as-is situations). Products and services that address requirements are characterized as solutions (often referred to as to-be situations). To deliver valued solutions, organizations must apply knowledge in problem-solving efforts; therefore, knowledge and the ability to apply it are the determining factor of success.

- Projects are problem-solving efforts that involve stakeholders and deliverables or work products in order to formalize the "work hard and hope for the best" approach to problem solving.
- Programs are collections of problem-solving efforts.
- Methods specify how to conduct problem-solving efforts.
- Processes are realizations of methods.
- Methodologies are taxonomies, or well-organized collections, of related methods.

The role of the UML is to enable and facilitate the following:

- Specifying, visualizing, understanding, and documenting problems.
- Capturing, communicating, and leveraging knowledge in problem solving.
- Specifying, visualizing, constructing, and documenting solutions.

However, the UML does not prescribe any particular problem-solving approach, but is very flexible and customizable to fit any approach. It enables and promotes (but does not require nor mandate) a use-case-driven, architecture-centric, iterative, and incremental process that is object oriented and component based.

- Use cases are used to manage and provide focus for a problem-solving effort.
- Architecture is used to manage complexity and maintain integrity and focus as a solution to a problem evolves.
- Iterations and increments are used to repeatedly apply a process to evolve a solution to a problem.

Fundamentally, the UML provides a means for addressing issues and risks concerning problems, solutions, and problem solving.

Problems and Solutions

Problems and solutions occur within a context (domain or space). The problem (system) must be understood in order to be solved. The solution (system) to a problem must be understood in order to be constructed and used. The solution must be organized (architecture) in order to facilitate its realization and adhere to the various constraints of the context in which it will be realized. To solve the problem, appropriate knowledge about the problem and solution must be captured (models), organized around decisions regarding the problem and solution (architectural views), and depicted (diagrams) using some language that enables it to be communicated and leveraged in the problem-solving process.

Therefore, the following concepts are critical to problems and solutions:

- Domains or spaces are organized collections of related elements in a self-contained situation or area of interest.
- Systems are organized collections of interacting and connected elements cooperating to accomplish a purpose.
- Architectures are schemes involving the structural and behavioral organization of systems within contexts (domains or spaces).
- Models are complete abstractions of systems or contexts. Abstraction involves focusing on those things that are relevant (essential) while avoiding those things that are irrelevant (incidental) to understanding something.
- Architectural views are abstractions of models.
- Diagrams are graphical projections of sets of model elements.

Problem Solving

Problem-solving approaches are organized (life cycles) to offer a management perspective and a development perspective. The two perspectives enable the effort to be managed and performed.

Problem solving requires being able to view the problem (paradigm) for the purpose of understanding it, and being able to view the solution (paradigm) for the purpose of realizing it. The problem-solving process involves leveraging knowledge to derive the solution (artifacts) to the problem through a series of (possibly concurrent) steps (activities) in which knowledge and rules of thumb (heuristics) gained from other problem-solving efforts may be used.

Therefore, the following concepts are critical to problems solving:

- Life cycles are collections of phases that divide efforts into several more manageable and controllable subordinate efforts.
- Paradigms are organized, self-contained collections of related components that form the basis for models.
- Artifacts are work products or deliverables resulting from efforts.
- Activities are efforts or collections of tasks directed at producing or developing artifacts. Workflows are collections of activities that are performed by specific roles.
- Heuristics are empirical or experience-based guidelines or rules of thumb.

System development may be characterized as problem solving, including understanding or conceptualize a problem, solving the problem, and implementing or realizing the solution. Conceptualizing a problem involves representing the problem using representational constructs (mental notions or ideas). Solving the problem involves manipulating representational constructs from the problem domain and the solution domain to derive a representation of the desired solution. Realizing a solution involves mapping those representational constructs of the solution unto the solution world, that is, constructing the solution. The use of representational constructs is a very natural process that often occurs subtly and sometimes unconsciously in problem solving. Underlying this scheme is the use of a paradigm in determining the possible types of representations utilized in problem-solving efforts.

- The object-oriented paradigm focuses on constructing reusable units and encompasses the conceptualization and specification principles of abstraction, encapsulation, inheritance, and polymorphism.
- The component-oriented paradigm focuses on the assembly of reusable units and encompasses the specification and realization principles of components, interfaces, and infrastructure.

The Unified Modeling Language

To successfully apply the UML, we must understand how the UML is holistically and cohesively organized to facilitate problem solving.

A language consists of a collection of concepts (semantics) with a notation (syntax) and rules (guidelines) governing the concepts and notation. Underlying a language and the methods that use a language is a foundation consisting of fundamental principles (axioms). These fundamental principles involve essential and "universally" accepted elements or constituents that define the constructs upon which a language is established (means) and facilitate some goals and scope to which the language applies (ends). The ends of the UML encompass models, architectural views, and diagrams to address why the UML exists. The means of the UML encompass the object-oriented paradigm and component-based development to address how the UML facilitates satisfying its ends.

The Architecture of the UML

To understand the architecture of the UML, consider how computer programs and programming languages are related. There are many different programming languages (C, C++, Java, Smalltalk, etc.), and each particular program is developed using a specific programming language. All of these languages support various declarative constructs for declaring data, and procedural constructs for defining the logic that manipulates data. Because a model is an abstraction, each of these concepts may be captured in set of related models. Programming language concepts are defined in a model called a metamodel. Each particular programming language is defined in a model that uses and specializes the concepts within the metamodel. Each program implemented in a programming language may be defined in a model called a user model that uses and instantiates the concepts within the model of the appropriate language. This scheme of a metamodel representing computer programming constructs, models representing computer programming languages, and user models representing computer programs exemplifies the architecture of the UML.

The UML is defined within a conceptual framework for modeling that consists of the following four distinct layers or levels of abstraction:

- The meta-metamodel layer consists of the most basic elements on which the UML is based -- the concept of a "Thing", representing anything that may be defined. This level of abstraction is used to formalize the notion of a concept and define a language for specifying metamodels.
- The metamodel layer consists of those elements that constitute the UML, including concepts from the object-oriented and component-oriented paradigms. Each concept within this level is an instance (via stereotyping) of the meta-metamodel concept "Thing". This level of abstraction is used to formalize paradigm concepts and define a language for specifying models.

- The model layer consists of UML models. This is the level at which modeling of problems, solutions, or systems occur. Each concept within this level is an instance (via stereotyping) of a concept within the metamodel layer. This level of abstraction is used to formalize concepts and define a language for communicating expressions regarding a give subject. Models in this layer are often called class or type models.
- The user model layer consists of those elements that exemplify UML models. Each concept within this level is an instance (via classifying) of a concept within the model layer and an instance (via stereotyping) of a concept within the metamodel layer. This level of abstraction is used to formalize specific expressions regarding a give subject. Models in this layer are often called object or instance models.

Within the fundamental UML notation, concepts are depicted as symbols and relationships among concepts are depicted as paths (lines) connecting symbols.

Models

Models capture the structural, or static, features of systems and the behavioral, or dynamic, features of systems. Models may be viewed via a small set of holistic but nearly independent and non-overlapping dimensions (aspects) that emphasize particular qualities of a model. The structural model dimension emphasizes the static features of the modeled system, and the behavioral model dimension emphasizes the dynamic features of the modeled system. Fundamentally, models capture knowledge (semantics).

Architectural Views

Architectural views organize models and knowledge around specific sets of concerns (architectural focus). The UML provides the following architectural views regarding models of problems and solutions:

- The user model view encompasses a problem and solution as understood by those individuals whose problem the solution addresses. This view is also known as the use case or scenario view.
- The structural model view encompasses the structural dimension of a problem and solution. This view is also known as the static or logical view.
- The behavioral model view encompasses the behavioral dimension of a problem and solution. This view is also known as the dynamic, process, concurrent, or collaborative view.
- The implementation model view encompasses the structural and behavioral dimensions of the solution's realization. This view is also known as the component or development view.
- The environment model view encompasses the structural and behavioral dimensions of the domain in which the solution is realized. This view is also known as the deployment or physical view.
- Other model views may be defined and used as necessary. An architectural focus is defined by a set of concerns (particular to stakeholders). An architectural view is defined by the set of elements from a model that address an architectural focus. For example, security issues may define an architectural focus. A security architectural view includes the set of elements from a model that address security issues.

Fundamentally, architectural views organize knowledge in accordance with guidelines expressing idioms of usage.

Diagrams

Diagrams depict knowledge in a communicable form. The UML provides the following diagrams, organized around architectural views, regarding models of problems and solutions:

- The User Model View
 - Use case diagrams depict the functionality of a system.
- The Structural Model View
 - Class diagrams depicts the static structure of a system.
 - Object diagrams depict the static structure of a system at a particular time.
- The Behavioral Model View
 - Sequence diagrams depict an interaction among elements of a system organized in time sequence.
 - Collaboration diagrams depict an interaction among elements of a system and their relationships organized in time and space.
 - State diagrams depict the status conditions and responses of elements of a system.
 - Activity diagrams depict the activities of elements of a system.
- The Implementation Model View
 - Component diagrams depict the organization of elements realizing a system.
- The Environment Model View
 - Deployment diagrams depict the configuration of environment elements and the mapping of elements realizing a system onto them.
- Other diagrams may be defined and used as necessary.

Fundamentally, diagrams depict knowledge (syntax).

Modeling Mechanisms

Mechanisms are practices for approaching modeling and diagramming that enable the creation of more precise and communicable models.

- Perspectives define a particular point of view from which to draw or read a diagram. They enable clear viewpoints to be associated with diagrams, and are used to heighten the effectiveness of the communication.
- Dichotomies define how something may be viewed from two different perspectives. They enable something to be viewed from multiple perspectives, and are used to discover inconsistencies within models.
- Layers or levels of abstraction define a particular level of abstraction and establish a level of detail at which attention and concentration are focused regarding a subject (problem or solution). They enable focused communication, and are used for organizing all of the diagrams pertaining to a single model into a coherent body of knowledge.

- Extension mechanisms define the means for customizing and extending the UML. They enable the UML to be flexible and adaptive, and used to ensure that the UML will evolve rather than be redefined to meet changing needs and requirements.

Within a problem-solving process, knowledge regarding a problem and solution is captured, organized around decisions, and depicted using the UML so that it can be communicated and leveraged. When deciding what diagram to use for communicating, consider the question or questions the communication is addressing, and what diagram or set of diagrams most effectively communicates the response. This decision centers on what dimensions of a model are to be emphasized in the response. Fundamentally, each diagram type emphasizes different dimensions of a model.

And within a problem-solving process, it is the underlying method that suggests how knowledge is utilized to realize a solution to a problem. This includes suggesting which diagrams to use and the perspective and the level of abstraction used to render and interpret these diagrams. Methods should be considered as suggestions and recommendations that organize and facilitate the problem-solving process rather than being considered rigid and inflexible rules that restrict the art of problem solving.

Conclusion

Conclusively, the Unified Modeling Language is an evolutionary general-purpose, broadly applicable, tool-supported, and industry standardized language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process. It is a fundamental communication mechanism that empowers organizations to capture, communicate, and leverage strategic, tactical, and operational business and technological knowledge on an enterprise-wide scale. Such knowledge can be applied to improve value by increasing quality, reducing costs, and reducing time-to-market while managing risks and being proactive to ever-increasing change and complexity.

By understanding the overall context in which the UML applies and how the UML is holistically and cohesively organized to facilitate solving problems, we have reviewed the required foundation for strategically determining how to successfully apply the language to maximize its benefits. However, caution should be emphasized. Simply because the UML evolved primarily from various second-generation object-oriented methods, the UML is not simply a third generation object-oriented modeling language. Its scope extends its usability far beyond its predecessors. And it is experience, experimentation, and gradual adoption of the standard that will reveal its true potential and enable organizations to realize its benefits.

Sinan Si Alhir is a consultant and author of “UML in a Nutshell” (O’Reilly and Associates, Inc., 1998).

Extreme Y2K Contingency Planning

Richard T. Dué, rtdue@ccinet.ab.ca, <http://ourworld.compuserve.com:80/homepages/rtdue>

Any organization that is now claiming Y2K compliance is simply unaware of what is the real Y2K Crisis. Y2K problems are unique in their essence, their scope, and their unremitting deadlines. There now are simply too many unknowns about the Y2K crisis for any organization to realistically claim 100% remediation.

What we don't know now about Y2K remediation

- No one can now say how big the Y2K problems will be, nor how long they will last.
- No organization now knows the actual state of their operating environment subsequent to January 1, 2000. The status of the telecommunications, utility, transportation, and government infrastructure on January 1, 2000 is unknown.
- The impact of infrastructure failure on the organization, on its employees and their families, suppliers, and customers is unknown.
- No organization now knows the impact of Y2K associated panic and panic reactions on their organizations.
- Few organizations even seem to know the size of their mainframe code remediation projects, let alone the size of their distributed processing, stand-alone computer, and embedded systems code remediation projects. In the past few days, the 15 largest banks in the United States reported that their Y2K budgets had increased by 10 to 20 percent in just the last 3 months. Supposedly, banks in the United States are the furthest advanced in their Y2K projects of any organizations in the world, yet they are apparently still unsure of what work has to be done and how much it will cost to do it. AT&T has revised its Y2K budget from \$500 million to \$800 million in just the last 3 months. Supposedly, telecommunications companies in the United States are the furthest advanced in their Y2K projects of any other telecommunications companies in the world, yet AT&T is apparently still unsure of what work has to be done and how much it will cost to do it. Organizations are now finding that the more work they do on Y2K remediation, and the deeper their understanding of the Y2K crisis, the larger the project becomes.
- The status of supply chain systems is unknown. Will organizations be able to request and receive the required supplies and services needed to stay in operation?
- The status of customer systems is unknown. Will customers be able to request, receive, and pay for supplies and services from the organization?

What do we know about Y2K remediation?

- We now know that Y2K remediation will be the largest information technology project in history. One financial company alone, Chase Manhattan / Travelers, is now budgeting close to \$1 billion for Y2K. World-wide, the estimates for Y2K have already reached \$850 billion with more than 1 year of budget revisions to go.
- We do know that the existing software project track record is abysmal. Most software projects are delivered late and over budget. The larger the project, the larger the proportional increase in defects in the delivered system. Most large software projects (in excess of 1 million lines of third generation language code) are complete failures. After all of the allocated time and budgets have been consumed, the projects are abandoned. The larger the project, the greater the rate of failure.
- We do know that we will have to wait until January 1, 2000 to adequately test if all remediation efforts have been successful. Critical infrastructure systems, for example, can not

be shut down and tested in advance. Interconnected systems can only be tested after all remediation work has been accomplished and only within the actual operating environment of January 1, 2000 (it doesn't matter if the code works if the infrastructure has shut down).

- We do know now that the only prudent approach must be to establish contingency plans for the most likely disaster scenarios facing us at work and at home.

Extreme Contingency Planning

So far, Y2K contingency planning has focused on three approaches:

1. Triage. Identify the critical systems that can be saved and work on them to the exclusion of other projects. The problems with triage include unidentified system dependencies and differing assessments of system priority. A system that has low priority could be a necessary resource for a critical system. A system with low priority within an organization could be critical to outside users.
2. Head for the hills. Things can't be fixed so prepare for personal survival. A recent and very troubling variant of this approach are informal reports of senior IT staff who are preparing to leave their organizations by mid-1999 in the face of impossible Y2K remediation requirements.
3. Don't worry, be happy. We don't know what will happen, but it won't be bad, and in any case, it will be fixed in three days.

I would like to offer a fourth approach that I call Extreme Contingency Planning. Extreme Contingency Planning is focused on the survival of the organization, not the survival of the organization's systems. The basis of this approach is to prepare for graceful business shutdown. To ultimately deal with all of the Y2K uncertainties some organizations are preparing in advance to shut down certain highly exposed parts of the business, or to shutdown certain business operations. For example, an organization may choose to stop accepting any new business until the status of its post January 1, 2000 infrastructure and information systems can be evaluated. An organization may choose to shut down or sell off parts of the business which are especially exposed to Y2K infrastructure and panic reactions or which can not be remediated cost effectively. Variants to the graceful business shutdown approach include:

1. Identification and servicing of key customers only. An organization may choose to only deal with a very small but economically important subset of customers. This subset may be as little as a few hundred customers. Key customer servicing will only require relatively simple systems to deal with low volume, but very high value transactions.
2. Quarantine and Embargo. Lockout all non-compliant or suspected non-compliant suppliers and customers from interacting with the organizations systems. In extreme cases, this means suspending transactions with certain industries, governments, and even geographical areas of the world to avoid data corruption from non-compliant systems.
3. Repeat December 1999. Keep printing December 1999 checks and invoices every month. This strategy is especially useful for buying additional remediation time for payroll, government benefits, pensions, etc. Most of the checks and invoices will be approximately correct, but organizations choosing this alternative will face a growing reconciliation problem. In effect, the repeat December 1999 tactic closes down the parts of the organization that deal with new business or changes to existing business arrangements.

Graceful business shutdowns may not last a long time. Perhaps the organization will be able to restart some functions in a few hours, weeks, or months. Some functions, however, may never be restarted. The important consideration is that the organization has survived the crisis even if its systems have failed. Some organizations are now preparing these extreme contingency plans. Its time to consider how you or your organization will respond to the extreme contingency plans of other organizations. Your own systems may be "compliant", but if your organization is embargoed, or not deemed to be a key customer, you could be out of business.

When you develop good software, you don't need to test carefully!

Franco Martinig, Martinig & Associates, franco@martinig.ch, www.martinig.ch

Well, this statement will seem exaggerated to the vast majority of the professionals of software development. However, this seems to be the conclusion that could be drawn from the evaluations performed by Martinig & Associates on software development processes worldwide.

When asked what aspect is the most important in their software development process, 53% of the evaluated organizations rank quality first, before schedule (40%) and costs (7%). However, the evidence of this importance does not appear in other parts of the evaluation. For instance, there is little usage of process quality models like the ISO 9000 series and the Capability Maturity Model. We found also that the questions concerning the test activities and tools have small percentage of positive answers.

- Are standards applied to the preparation of unit test cases? 31%
- Is test coverage measured and recorded for each phase of functional testing? 23%
- Are software trouble reports resulting from testing tracked to closure? 77%
- Is test progress tracked by deliverable software component and compared to the plan? 33%
- Is the error data from code reviews and tests analyzed to determine the likely distribution and characteristics of the errors remaining in the product? 16%
- Is a mechanism used for independently calling integration and test issues to the attention of the project manager? 51%
- Is there a mechanism for assuring that regression testing is routinely performed? 27%
- Is there a mechanism for assuring the adequacy of regression testing? 14%
- Are formal test case reviews conducted? 27%
- Are automated test input data generators used for testing? 15%
- Are computer tools used to measure test coverage? 10%
- Are computer tools used to track every required function and assure that it is tested and verified? 10%

Usage of testing practices and tools. Source: Martinig & Associates.

As you can see, there is little evidence of an infrastructure to plan and follow the testing phase of software development, even in fundamental aspects like defects tracking. It is also true that the developers have often a limited knowledge of the basic testing techniques, and even more often a very limited inclination to apply them. Developing software is fun, but testing or debugging it with care is not. These activities are often associated with the burden to build test data and to execute modules repetitively with different settings. That is not cool...

However, there has been a tendency in the recent times to take the opportunity of the special Year 2000 budgets to introduce in the organizations more rigorous testing practices and to buy automated testing tools. It is still to be verified if this spending pattern will effectively influence the testing practices of software development units in the long term or if developers will revert to their old habits after the Year 2000 deadline.

The figures above are computed from the evaluation of the software processes of 186 organizations in Europe and North America. Other results, information on our assessment reports in the banking industry and the questionnaire for a free evaluation of the software development process can be found on the web site of Martinig & Associates: www.martinig.ch

Facts, Opinions & Comments



Companies

Bargain Hunting in the Software Development Industry

As the Year 2000 deadline approaches and organizations are implementing "all budget for Y2K" and "no change unless necessary" policy to their information systems, we will see this year more acquisitions of financially endangered software development tools vendors. Just to remember, last year saw the end of Intersolv, Cayenne Software, Seer, Synon and Nat Systems as independent companies. The list should be bigger this year, as the recent Computer Associates (CA)-Platinum deal and the problems faced by Inprise (ex-Borland) show us. Look in future M&T issue to find a report on "Who eats who" in the software development tools industry and let's hope that Delphi will not be acquired by CA.

\$9%£ Numbers

Forecasting: Who Gets the Most of IT?

"Internet forecasting is a great way to make money. Last fall Forrester Research in Cambridge, Mass., issued a release entitled ONLINE GROCERY SALES TO REACH \$10.8 BILLION BY 2003. Then its 100 salespeople fanned out to sign up grocers. The ones convinced they needed help agreed to fork over annual fees ranging from \$15'000 to more than \$ 1 million to subscribe to the firm's weekly reports and monthly 16-pages analyses [...].

No one knows how much real insight online consulting firms provide, but what would-be Internet can afford not to subscribe? Planning for the future is hard in a mature industry; It's nearly impossible in one still teething. The result is that demand for forecast is insatiable. "You put those numbers out there, and it's like putting meat in a sink full of piranhas" says Scott Smith, a former Jupiter analyst [...]

So frenzied has activity in this area become that there are even consultants to offer you advice on other consultants. Outsell in Burlingame, Calif., advises clients which services to subscribe to and how much to pay. New York startup eMarketer offers glossy reports compiled from the free projections the new-media research firms. The reports sell for \$495 to \$795 apiece. [...]

The forecasts do look great. Forrester says that by 2002, North American online advertisers will spend \$7.8 billion, or 14 times the \$550 million it says they spent in 1997. Jupiter, for its part, predicts that by 2002, North American companies will spend \$7.7 billion advertising online, or eight times the \$940 million it says they spent in 1997."

Source: "My, what big Internet numbers you have! The better to hook you with, my dear", Daniel Roth, Fortune, March 15, 1999

We are familiar in our industry to look at the predictions of Gartner Group & Friends about our future: "Object oriented databases market to reach \$ 2,3 billion in 1999" or "Java is the language of the future". There is nothing wrong with trying to help companies making choices and making inaccurate predictions. But making predictions only to gain customers just does not seem to be something very ethical to me... Think about it next time you will look at one of these forecasts.



In Others' Words

Object: Solution or Problems?

"OT is tremendously helpful in achieving strategic goal and fulfilling business needs. However, because it is relatively new, it requires a transition plan, and its implementation must be closely supervised. [...] We also designed our strategy to solve the type of problems we, and many others, have encountered during OO projects:

- A lack of coherency in methodology and process across the enterprise causes backward incompatibility and product line integration problems. For customers, it reduces plug-and-play capabilities and makes it harder to upgrade to newer product lines.
- Architectural instability and incompleteness cause schedule delays and results in software components that must be redesigned for each development iteration. This also results in throwaway code and, therefore, low developer morale.
- A lack of staff familiarity with OT requires a substantial investment in training. A software developer typically needs six months of training to understand OO methodology, C++, and development tools. This must be included in the development schedule to avoid project delays.
- Staff inexperience with OT leads to an overly complex system architecture and implementation, which can degrade maintainability, performance and reliability.
- Because it is new, distributed OT software development is hard to learn and to use. Support tools are generally slow and unreliable, so their use will hurt system performance.
- Lack of a code reuse strategy leads to redundant implementation.
- Developers who inadequately evaluate off the shelf product's applicability to their projects may spend time developing capabilities when existing products already provide them (thus reinventing the wheel). They may also use inappropriate product features in system design, which may cause performance and reliability problems.
- Not having an OO test environment and strategy affects software implementation stability and reliability"

Source: Computer, "An OO Project Management Strategy", Babak Sadr & Patricia Dousette, September 1996.

I think that this text is still valid and all the problems mentioned explain why we found in our database of software process evaluations that only 50% of the organizations are using object technology.

No New Models!

"[...] What we do need is for practitioners to routinely and effectively apply the techniques defined by our existing models and frameworks. Once we've reached their practical limit, we can turn to improved models that provide guidance for working in better ways. Some current approaches may be unworkable, and projects on the bleeding edge of technology, business needs, or development approaches may find current methods inadequate. More sophisticated models may also benefit practitioners who have in fact pushed current methods to their limits.

My sampling of audiences at conferences and training seminars suggests, however, that many organizations do not consistently apply existing approaches for software development excellence.

I outline here several sets of software engineering and management practices that, in my experience, are still not being routinely applied across the industry. Far from a lack of suitable models to help us structure our thinking and practice, the problems we most often face include

- insufficient awareness of current best practices and published standards in software development, management, and quality;
- inadequate training of practitioners and managers in these established practices;
- resistance to change, expressed as the "not-invented-here" syndrome and an insistence that "our project is different and those things don't apply"; and
- a shortage of discipline, rigor, and available time for people to continuously improve their personal software process by applying a broad spectrum of superior techniques. [...]

Testing

A software quality magazine recently published a series of columns delineating an elaborate model for system testing, and leaders in the software quality industry have proposed several testing maturity models. While some very large, very regimented projects may actually apply these complex testing models, I think the real testing issues are closer to home.

Thing about the developers in your organization who also do some testing. Do they have testing books on their desk? (My informal surveys of conference attendees suggest not.) Have they been trained in testing? (Ditto.) Do they write test plans? Are their test documented and repeatable? Do they understand basic testing concepts? Can they describe the state of a program after testing is complete?

If practitioners cannot answer "yes" to such questions, a better testing model probably won't help them. Some training on testing practices and concepts, test case design and documentation, and the use of automated testing tools will do more good than a new testing model. Experienced testers can enhance their productivity by following the dictates of a rigorous testing model. However, formulating ever more elaborate models of the testing process will do little to improve how average software developers test their products. [...]

Practice what we preach

I don't think models are bad, I have found models that help structure my thinking and provide a framework for making sensible decisions to be extremely valuable. My point is that the software industry is not fully exploiting the models, standards, and frameworks we already have available. Before we invent new models, let's help developers, managers and quality professionals consistently and effectively apply the practices embodied in those that currently exist. As educators, let's incorporate into our curricula a solid foundation of software industry best practices, along with guidance on how to put them into action. As managers, let's emphasize continuous learning in our organizations and reward those who apply better way of working. As practitioners, let's read the literature and commit to improving our personal software processes through effective application of what we've learned from others.

And as industry leaders, let's not clutter the market with Yet Another Model until we're convinced the ones we have are truly not getting the job done."

Source: Karl E. Wiegers, "Read My Lips: No New Models", Head to Head, IEEE Software, September/October 1998

There is little to add to these excerpts from the thoughts of Karl Wiegers. I am sure that I am not the only one out there that wonders why we do not try to apply more common sense to software development projects and processes. Is this because this is not in human nature?

America's Second Discovery?

"Boag is worried about the loosening of immigration laws in the US, which she says will cause staff from the UK to quit Europe and drift to the other side of the Atlantic. This would sharpen the effect of the 'brain drain' – the hemorrhaging of skills in the UK IT industry. The problem stems from pressure to Congress by the US software and computer processor industry. Large IT firms that are having difficulty finding staff have asked Congress to relax immigration laws to allow more professionals into the US. If their demands are met, then the number of annually available H1-B visas, which allow professionals into the States from other countries, will increase by an amount yet to be decided.

'I'm trying to act as a bit of an early warning system because I'd heard about what was happening in the US' says Boag [...] 'many companies in this country would be concerned if their permanent and contract staff went to the US.'

Trish Boag, chair of the Human Resources Group, UK Computing Services and Software Association, cited in Application Development, July/August 1998

True or false? contact franco@martinig.ch to comment on this topic!



Coming next in Methods & Tools

- Components Software Enters The Mainstream
- Configuration Management and ISO 9001
- Data Warehouse Design
- Estimating Software Earlier and More Accurately
- How to Sponsor a Successful Project
- Object Oriented Group Work
- Testing Client/Server Applications

Classified Advertisement

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organizing software development related training? Selling your new report? This space is waiting for you at the price of US \$ 20 each line. Reach more than 2000 (end of March) web-savvy software developers and project managers worldwide with a classified advertisement in Methods & Tools. Without counting those that download the issue without being registered! To advertise in the classified section, to place a page ad or to become the distribution sponsor of the next issue, simply contact Franco Martinig at franco@martinig.ch