

---

---

# METHODS & TOOLS

---

---

Global knowledge source for software development professionals

ISSN 1023-4918

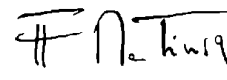
Summer 1999 (Volume 7 - number 2)

## Managing Change to Manage Stability

We have to admit that, and this is even truer in the MIS world, there is often little of engineering in our software engineering activities. In complex projects involving hundred of modules and parameter data, the integration testing phase is a difficult and stressing moment for the project members. Functions can go well through certain test scripts and fail the next time. The usual question is then: what has been changed? This question is even more crucial when this kind of problems appears in a "live" application.

Configuration management is a technique used to answer these questions. In its simple form, configuration management consists simply to track who made changes to certain parts of an information systems. In a more elaborate approach, the change management activity is centralized and certain criteria have to be respected so that an item can go from one environment to the other, from maintenance to production for instance. For medium to large projects, the elaborate form is certainly more efficient than the simple tracking of changes, but it also imposes more discipline to the developers (they will call it constraints). The difficulty lies in achieving a balance between contradictory goals. If there is a bug or if a user wants to modify something during the test phase of a new application, the natural answer of the average developer is to correct the problem as soon as possible. This is even truer if the developer has retained the "ownership" of the faulty modules. On the other side, the configuration manager wants to be sure that the problem has been correctly analyzed, that all impacts have been examined and that an adequate testing process, called regression testing, has been performed to prove that the requested modification really corrects the problem and does not have negative side effects. In the heat of the action, and even cool-headed, all those activities that seem to postpone the end of the issue are disliked by developers.

Elaborate software configuration management is an efficient tool to manage the stability of information systems. IS managers must use this technique. They have also to educate the developers about the positive results of adopting such a discipline and make sure that the software development team and the configuration management team are collaborating for the success and the quality of the project and not fighting against each other.



### Inside

Quality: Configuration Management and ISO 9001- .....	page 2
Testing: Introducing Automated Testing .....	page 11
Databases: Choosing Object Databases.....	page 16
Facts, News & Comments .....	page 21

## Configuration Management and ISO 9001

Robert Bamford, William J. Deibler II, Software Systems Quality Consulting  
ssqc@concectic.net, www.ssqc.com  
2269 Sunny Vista Drive, San Jose CA 95128, USA

---

**Configuration management is about managing change of the multiple items composing an information system. In this article, Robert Bamford and William Deibler put in reference the configuration management function and the ISO 9001 standard. This standard offers a wide range of advice on how to deal with this important, but often neglected, aspect of software engineering.**

The software engineering practices associated with software configuration management (SCM or CM) offer a number of opportunities to address requirements found in the International Standard, ISO 9001. From a management perspective, the principles and practices of CM represent an accepted and understood foundation for implementing ISO-compliant processes in software engineering organizations. In addition, the growing number of tools for automating CM practices is avenues for improving the efficiency and effectiveness of these processes.

This article begins with brief, general definitions of configuration management and of ISO 9001.

### Configuration Management

While there is no single definition of CM, there are three widely disseminated views from three different sources: the Institute of Electrical and Electronics Engineers (IEEE), The International Organisation for Standardisation (ISO), and the Software Engineering Institute (SEI) at Carnegie Mellon University.

### The IEEE perspective on CM

A most widely understood description of the practices associated with configuration management is found in the IEEE Standard 828-1990, *Software Configuration Management Plans*.

[Numbers in brackets are added]

"SCM activities are traditionally grouped into four functions: [1] configuration identification, [2] configuration control, [3] status accounting, and [4] configuration audits and reviews."

IEEE Standard 828-1990 goes on to list specific activities associated with each of the four functions (the number of the paragraph containing the reference appears in parentheses):

- ◆ **Identification:** identify, name, and describe the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled for the project. (Paragraph 2.3.1)
- ◆ **Control:** request, evaluate, approve or disapprove, and implement changes (Paragraph 2.3.2)
- ◆ **Status accounting:** record and report the status of project configuration items [initial approved version. status of requested changes, implementation status of approved changes] (Paragraph 2.3.3)
- ◆ **Audits and reviews:** determine to what extent the actual configuration item reflects the required physical and functional characteristics (Paragraph 2.3.4)

This list is similar to the set of activities noted by Pressman:

"Software configuration management is an umbrella activity ... developed to (1) identify change, (2) control change, (3) ensure that change is being properly implemented, and (4) report change to others who may have an interest."

### The ISO perspective on CM

In the guideline document, ISO 9000-3:1991 *Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*, the International Organisation for Standardisation identifies a similar set of practices as CM:

"Configuration management provides a mechanism for identifying, controlling and tracking the versions of each software item. In many cases earlier versions still in use must also be maintained and controlled. "The [CM] system should

- "a) identify uniquely the versions of each software item;
- "b) identify the versions of each software item which together constitute a specific version of a complete product;
- "c) identify the build status of software products in development or delivered and installed;
- "d) control simultaneous updating of a given software item by more than one person;
- "e) provide coordination for the updating of multiple products in one or more locations as required;
- "f) identify and track all actions and changes resulting from a change request, from initiation ... to release."

### The SEI perspective on CM

Based on a review of currently available tools and an evolving understanding of the organizational role of CM, the SEI advocates a broader definition of CM in SEI-92-TR-8:

"The standard definition for CM taken from IEEE standard 729-1983 [updated as IEEE Std 610.12-1990] includes:

*"Identification:* identifying the structure of the product, its components and their type, and making them unique and accessible in some form

*"Control:* controlling the release of product and changes to it throughout the life cycle ...

*"Status Accounting:* recording and reporting the status of components and change requests, and gathering vital statistics about components in the product

*"Audit and review:* validating the completeness of a product and maintaining consistency among the components ...

"[The IEEE] definition of CM ... needs to be broadened to encompass ... :

*"Manufacturing:* managing the construction and building of the product

*"Process management:* ensuring the correct execution of the organization's procedures, policies, and life-cycle model

*"Team work:* controlling the work and interactions between multiple developers on a product."

### ISO 9001

In 1987, the International Organisation for Standardisation in Geneva Switzerland published **ISO 9001, Quality Systems - Model for quality assurance in design / development, production, installation, and servicing.**

ISO 9001 is the most comprehensive model in the ISO 9000 series of standards. It describes a minimum set of activities found in companies and organizations that consistently produce products that satisfy customer requirements. The policies, procedures, standards, records, and associated business activities are *the quality system*. While ISO 9001 is written to describe any company providing any product or service, it tends to employ manufacturing terminology, which must be interpreted for non-manufacturing

environments, including service and software providers.

To ensure a uniform interpretation of ISO 9001 for software engineering organizations, ISO published **ISO 9000-3, Guidelines for the Application of ISO 9001 to the development, supply and maintenance of software.**

The key issues ISO 9000-3 addresses are those:

- ◆ Product exists earlier in software (during design and development)
- ◆ Software product can be proliferated easily

Focusing on these issues mirrors the guidance in Clause 7.4 of ISO 9000-1:1994:

The process of development, supply, and maintenance of software is different from that of most other types of industrial products in that there is no distinct manufacturing phase. Software does not “wear out” and, consequently, quality activities during the design phase are of paramount importance to the final quality of the product.

Note that ISO 9000-1 and ISO 9000-3 provides guidance. ISO 9001 is the only source of the requirements against which compliance in software engineering practices is assessed.

### ISO 9001 and Configuration Management

Tracing the relationship between ISO 9001's requirements and CM practices begins with an examination of the guidance in ISO 9000-3.

### ISO 9000-3 and configuration management

ISO 9000-3 contains two appendices, Annex A and Annex B, that provide cross-references between ISO 9001 and ISO 9000-3. According to Annex A, five sections of ISO 9001 correlate to ISO 9000-3, Paragraph 6.1, Configuration Management:

- ◆ 4.4 Design control
- ◆ 4.5 Document data control
- ◆ 4.8 Product identification and traceability
- ◆ 4.12 Inspection and test status
- ◆ 4.13 Control of nonconforming product

Each of these sections of ISO 9001 contains a portion of the traditional CM process.

*4.4 Design control* addresses all of the steps in the software development life cycle: planning, specification, design, coding, testing

Section 4.4 requires that design inputs and outputs be documented, reviewed, verified, controlled, approved, and modified according to documented procedures. *Design inputs* and *outputs* include plans (project life cycle definition), specifications, prototypes, requirements documents, progress reports, review results, test plans, test cases/scripts, development tools, code, and test reports.

ISO 9001 *4.4.9 Design changes*, in conjunction with ISO 9001 *4.14.2 Corrective action*, and *4.13 Control of nonconforming product*, requires that each change be traceable to an appropriate source and approval.

For software product there should be a clear path between a change request spawned by a fault report or enhancement request and a change in a specific product component to correct the fault or to implement the enhancement.

An interested party should be able to pick up the path at any point and follow it forward to the released change and backward to the change request or to the fault report.

*4.5 Document and data control* addresses the identification, protection, approval, and availability of current issues of all pertinent product- and project-related documents, including designs, specifications, plans, and schedules.

Because a fundamental function of CM is making current configuration items available, the CM practices and tools can be applied to the control of product- and process-related documentation and data.

*4.8 Product identification and traceability* requires that each version of a configuration item be identified by some appropriate means.

*4.12 Inspection and Test Status* requires procedures to identify what verification steps and tests have been completed and what results have been achieved by the product or product components at each phase in the defined development life cycle.

*4.13 Control of Nonconforming Product* requires procedures to ensure that untested, defective, or incorrect versions (e.g., down level) of the product are not *inadvertently* used. This paragraph of ISO 9001 also requires a procedure to determine the disposition of nonconforming product at all stages.

For software, the bulk of the activity related to non-conforming product is in the correction of faults identified during all phases of development (e.g., during requirements definition, prototyping, integration testing, and beta testing) and after the product has been released (e.g., customer reported faults).

### **Beyond ISO 9000-3**

There are a significant number of additional areas of ISO 9001 that can be addressed through CM-related activities.

- ◆ 4.1 Management responsibility
- ◆ 4.2 Quality system
- ◆ 4.4.2.2 Organizational and technical interfaces
- ◆ 4.6 Purchasing
- ◆ 4.7 Control of customer-supplied product
- ◆ 4.9 Process control
- ◆ 4.14 Corrective and preventive action

- ◆ 4.15 Handling, storage, packaging, preservation, and delivery
- ◆ 4.16 Control of quality records
- ◆ 4.19 Servicing
- ◆ 4.20 Statistical techniques

### *4.1 Management responsibility*

Reports produced by the CM system on progress and exceptions support management review of the suitability and effectiveness of the development practices, as part of the quality system (ISO 9001 4.1.3).

### *4.2 Quality system*

For a software engineering organization, the CM policies, procedures, and standards represent a significant portion of the quality system. Tools to support and automate the CM process support and enforce adherence to policies, procedures, and standards.

These procedures can also be automated through integrated workflow and groupware tools that increase the effectiveness and efficiency of the information exchange.

### *4.4 Design control*

CM practices can go beyond control of configuration items to ensure that necessary information regarding status and change is communicated to appropriate individuals and organizations (ISO 9001 4.4.2.2).

Standard distribution lists and notification procedures linked to specific activities prevent significant missed communication.

### *4.6 Purchasing*

The primary application of this paragraph of ISO 9001 in software engineering environments is to third party development. When an organization subcontracts software development to a third party, evaluation of the subcontractor's CM practices is a critical component of the vendor selection and approval process.

If appropriate, the subcontractor can be required to follow or implement specific CM practices.

Intermediate or final product and all related documentation (e.g., specifications, plans, progress reports, test reports) received from the subcontractor can all be treated as if they were in-house developed configuration items. Applying CM practices to third party development is of particular benefit for coordinating in-house integration and verification activities and for fault resolution.

#### *4.7 Control of customer-supplied product*

In software development, customer-supplied product includes software that is used in the development process or that is included in the product to be delivered to the customer. This software is specified by the customer and supplied by the customer or by a third party; the software can be a standard, off-the-shelf (shrink-wrapped) product or one that is custom developed.

Depending on how the included software is packaged and distributed, ISO requirements to verify, store, and maintain this software appropriately may met by considering the included software as a configuration item.

Requirements for the verification of customer-supplied product apply only to those portions of the customer-supplied product that are used in conjunction with the supplier's product. For example, if the customer specifies that the supplier's product is to run under UNIX System 5, the supplier's responsibility is to verify that the developed product works as specified under UNIX System 5. The supplier must identify and report any errors in UNIX System 5 that impact the operation of the supplier's product.

#### *4.9 Process control*

*Process control*, in conjunction with 4.4.2 *Design and development planning*, 4.2.2 *Quality system procedures*, and 4.2.3 *Quality*

*planning* clarifies ISO 9001's implicit requirements for documented procedures, suitable production equipment, monitoring and control of process and

product characteristics, and approval of processes and equipment. In software engineering environments, the project management and CM processes combine to address the majority of these requirements.

By automating the product build process, a CM system contributes significantly to the effectiveness and efficiency of the software production process, both for intermediate versions of the product and for a released version.

This becomes particularly significant, when multiple versions of the product are being developed or maintained in parallel.

#### *4.14 Corrective and preventive action*

A significant portion of corrective action is creating the mechanism to ensure that customer-reported problems are resolved in an appropriate manner. The same requirements pertain to problems identified in the development process, starting from the point at which the software product or item comes under CM control.

Incidents must be tracked from report, through classification, and, if appropriate, to resulting changes in the product. CM practices, particularly those related to change management, product maintenance, and status accounting, ensure that incidents that result in product change are always handled properly.

There is significant opportunity for improving the efficiency of product support and software engineering organizations by minimizing the amount of manual intervention and effort in moving information between the problem tracking and the CM systems.

#### *4.15 Handling, storage, packaging, preservation, and delivery*

For software product, the CM practices address all of the handling, storage, packaging, preservation, and delivery requirements at least to the point where responsibility for the product is turned over to software production. ISO 9001 4.15.2 *Handling* specifies “methods of handling product that prevent damage or deterioration”. For software product, this requirement is interpreted to include activities like virus checking if an outside replication vendor is used and off-site storage of product masters as a minimum level of disaster recovery. Automated support for the build process, included in most CM tools, reduces opportunities for error and can increase confidence in intermediate test results and in final product integrity.

#### **4.16 Control of quality records**

In ISO 9000, quality records are the records that establish that processes were followed and that quality requirements were met. By definition, *quality records* includes records of:

- ◆ Product identification
- ◆ Non-conformity review and disposition
- ◆ All verification and validation activities, including: design review minutes, test logs and records, and fault reports

While these records are not documents (and are not subject to the requirements of ISO 9001 Paragraph 4.5), requirements for *identification, collection, indexing, filing, storage, maintenance, and disposition* of quality records can be addressed through procedures implemented as part of the CM system.

#### **4.19 Servicing**

ISO 9000-3 ties *servicing* to all aspects of software maintenance, including problem resolution, interface modification (e.g., support for additional or modified hardware components), functional expansion or

performance improvement. CM practices ensure that the product is maintained in an orderly manner and that each approved change can be prioritized, tracked, and managed to completion.

Analysis of the data in the CM system related to all aspects of product maintenance can support systematic prioritization and planning for product and process enhancement (e.g., *What modules change most often? What modules cause the most problems? Is the effectiveness of testing continuing to improve?*)

#### **4.20 Statistical techniques**

While ISO 9001 contains no specific requirements for statistical process control, as noted above, CM-related activities generate a wealth of process and product data for analysis and comparison to plan: delivery dates, resources, benchmarking (e.g., lines of source code, executable size, performance), time to correct defects, etc.

Even if no *modern statistical methods* are implemented (e.g., Statistical Process Control, Design of Experiments, as suggested in Clause 20 in ISO 9004-1:1994), this data is considered input for ISO 9001 4.9d, which requires “monitoring and control of suitable process parameters and product characteristics”.

The data in the CM system is a primary input for problem analysis and the identification of root causes in products and processes.

Summary - ISO 9001 and configuration management

As described in the preceding paragraphs, of the 20 sections of ISO 9001 that define a supplier's capability to meet customer requirements, CM practices directly impact the following. Check marks in the following table indicate clauses of ISO 9001 that are addressed by CM practices.

<b>Section of ISO 9001</b>		
4.1	Management responsibility	☑
4.2	Quality system	☑
4.3	Contract review	
4.4	Design control	☑
4.5	Document and data control	☑
4.6	Purchasing	☑
4.7	Control of customer-supplied product	☑
4.8	Product identification and traceability	☑
4.9	Process control	☑
4.10	Inspection and testing	
4.11	Control of inspection, measuring and test equipment	
4.12	Inspection and test status	☑
4.13	Control of nonconforming product	☑
4.14	Corrective and preventive action	☑
4.15	Handling, storage, packaging, preservation and delivery	☑
4.16	Control of quality records	☑
4.17	Internal quality audits	
4.18	Training	
4.19	Servicing	☑
4.20	Statistical techniques	☑

In terms of improved efficiency, major opportunities exist in ensuring that the CM, project management, customer technical support, build management, and problem reporting and tracking systems are as tightly coupled as possible.

**Beyond CM - Product Attributes and Tool Selection**

Nothing in ISO 9001 requires that a specific tool or technology be employed. ISO's sole concern is that the implemented systems be *effective* in delivering the promised product or service. Tools that automate CM practices

may improve the effectiveness and will improve the *efficiency* of systems that require significant manual intervention.

From an ISO implementation perspective, automation represents an opportunity to:

- ◆ Reduce process and project documentation
- ◆ Reduce requirements for training
- ◆ Ensure that required steps are completed
- ◆ Record progress and activity

The tool selection process should ensure that the selected tools support the current or planned software engineering practices. While some minor changes to engineering practices may be required (especially if standard tools with minimum customization are selected), the tools cannot be the basis for formulating engineering practices.

**Considerations in tool selection**

While a number of factors determine the appropriateness of a particular tool, the following is an initial list of information that is required to evaluate a tool for suitability.

- ◆ Product
  - Number of products
  - Shared/common components
  - Application complexity
  - Number of platforms supported; number of platforms on which development is performed
  - Maintenance of multiple versions (multiple platforms, application variants)
- ◆ Project
  - Size of project
  - Need to maintain, control, adapt, extend, etc. - project or product
  - Risks associated with the product, project, and related commitments
- ◆ Process
  - Code structure and techniques
  - Concurrent development
  - Paradigm (configuration items, developers' requirements for access, managers' requirements for control and information)
  - Stability and flexibility

- ◆ People
  - Organization size and experience
  - Capacity of the organization to adapt
- ◆ Existing tools that will be retained
  - In the engineering organization
  - In organizations that interface with engineering
- ◆ Integration with other tools under consideration
  - Project management
  - Build management
  - Customer technical support
  - Problem reporting and tracking

Based on Feiler's characterization of current CM tools, certain aspects of tool functionality and process automation emerge as key differentiators among the competing models and tools.

- ◆ Merging and propagation of changes (to support multiple versions, concurrent development, shared/common components)
  - Among parallel versions
  - Between branches and mainstream or shared core
- ◆ Restoration of previous versions
- ◆ Identification and control of changes
- ◆ Identification and control of product and product components
- ◆ Automated build management

With this information, the ability of a tool to support a particular organization and its development practices can be evaluated objectively and any requirements for immediate or future customization can be defined.

---

### Bibliography and Recommended Reading

[Ba1] Wayne A. Babich, **Software Configuration Management**, Addison Wesley Publishing Company, Reading MA, 1986

[Bb1] R.C. Bamford and W.J. Deibler,

*Comparing, contrasting ISO 9001 and the SEI capability maturity model*, **IEEE Computer**, October 1993, Vol 26, No. 10, IEEE Computer Society, page 68

[Bb2] R.C. Bamford and W.J. Deibler, **A Detailed Comparison of the SEI Software Maturity Levels and Technology Stages to the Requirements for ISO 9001 Registration**, Software Systems Quality Consulting, San Jose, Calif., 1993.

[Bb3] R.C. Bamford and W.J. Deibler, **MKS RCS Version 6.2 and ISO 9001**, Software Systems Quality Consulting, San Jose, Calif., 1993.

[Bb4] R.C. Bamford and W.J. Deibler, **ISO Implementation as a Managed Process - A Software Perspective**, Software Systems Quality Consulting, San Jose, Calif., 1993.

[Da1] Susan A. Dart, **The Past, Present, and Future of Configuration Management**, CMU/SEI-92-TR-8, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, July 1992 (available by anonymous ftp from ftp.sei.cmu.edu)

[Fe1] Peter H. Feiler, **Configuration Management Models in Commercial Environments**, CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, March 1991 (available by anonymous ftp from ftp.sei.cmu.edu)

[Hu1] Watts S. Humphrey, **Managing the Software Process**, Addison Wesley Publishing Company, New York, August 1990 (Reprinted with corrections)

[Ie1] IEEE Std 828-1990, **IEEE Standard for Software Configuration Management Plans**, IEEE, 345 E. 47th St., New York, NY (from the Spring 1991 Software Engineering Standards Collection, April 5, 1991)

- [Ie2] IEEE Std 610.12-1990, **IEEE Standard Glossary of Software Engineering Terminology**, , IEEE, 345 E. 47th St., New York, NY (from the Spring 1991 Software Engineering Standards Collection, April 5, 1991)
- [In1] ISO 9001:1987 **Model for design/development, production, installation and servicing**, International Organisation for Standardisation, Geneva, Switzerland, 1987
- [In2] ISO 9000-3:1991 **Guidelines for the application of ISO 9001 to the development, supply and maintenance of software**, International Organisation for Standardisation, Geneva, Switzerland, 1991
- [In3] ISO 9000-1:1994 **Quality management and quality assurance standards - Guidelines for selection and use**, International Organisation for Standardisation, Geneva, Switzerland, 1994
- [In4] ISO 9004-1:1994 **Quality management and quality system elements - Guidelines**, International Organisation for Standardisation, Geneva, Switzerland, 1991
- [Pa1] Mark C. Paulk et al., **Capability Maturity Model for Software, Version 1.1**, CMU/SEI-93-TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA 15213, February 1993
- [Pa2] Mark C. Paulk et al., **Key Practices of the Capability Maturity Model, Version 1.1**, CMU/SEI-93-TR-25, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA 15213, March 1993
- [Pr1] Roger S. Pressman, **A Manager's Guide to Software Engineering**, McGraw-Hill, Inc., New York, 1993
- [Wh1] David Whitgift, **Methods and Tools for Software Configuration Management**, John Wiley & Sons, New York, 1992

## **Introduction of Automated Test to A Project**

Elfriede Dustin, Jeff Rashka and John Paul  
<http://www.autotestco.com>

---

At the start, a new candidate for (a) paradigm (change) may have few supporters, and on occasions the supporters' motives may be suspect. Nevertheless, if they are competent, they will improve it, explore its possibilities, and show what it would be like to belong to the community guided by it. And as that goes on, if the paradigm is one destined to win its fight, the number and strength of the persuasive arguments in its favor will increase. (Thomas Kuhn, "The Structure of Scientific Revolution")

### **Introduction**

Today's software managers and developers are being asked to turn around their products within ever-shrinking schedules and with minimal resources. This is due to a push from federal government and commercial industry toward streamlining the product acquisition life cycle as well as the survival necessity within the commercial software product industry of getting a product to market early. Coupled with these pressures, the Year 2000 problem has placed an additional burden on information technology organizations within companies. Again, most Information Resource Management (IRM) organizations are struggling to handle this challenge within the constraints of existing information management budgets and while in the middle of implementing other competing information management initiatives.

In an attempt to do more with less, organizations want to test their software adequately, but as quickly as possible. Faced with this reality, software managers and developers have little choice but to introduce automated testing to their projects. While needing to introduce automated testing, software professionals may not know what's

involved in introducing an automated test tool to a software project, and may not be familiar with the breadth of application that automated test tools have today.

Manual testing is labor intensive and error prone, and does not support the same kind of quality checks that are possible through the use of an automated test tool. Humans make mistakes, especially when mundane, repetitive tasks are involved. Computers are especially effective in performing these mundane and repetitious tasks, without the mistakes. The introduction of automated test tools helps to replace archaic and mundane manual test processes with a more professional and repeatable automated test environment, that promotes test engineer retention and improves test engineer morale.

### **Introducing Automated Test**

Yet, a new technology is often met with skepticism and software test automation is no exception. How test teams introduce an automated software test tool on a new project is nearly as important as the selection of the most appropriate test tool for the project. A tool is only as good as the process being used to implement the tool.

Over the last several years test teams have largely implemented automated testing tools on projects, without having a process or strategy in place describing in detail the steps involved in using the test tool productively. This approach commonly results in the development of test scripts that are not reusable, meaning that the test script serves a single test string but cannot be applied to a subsequent release of the software application. In the case of incremental software builds and as a result of software

changes, these test scripts need to be recreated repeatedly and must be adjusted multiple times to accommodate minor software changes. This approach increases the testing effort and brings subsequent schedule increases and cost overruns.

Perhaps the most dreaded consequence of an unstructured test program is the need for extending the period of actual testing. Test efforts that drag out unexpectedly tend to receive a significant amount of criticism and unwanted management attention. Unplanned extensions to the test schedule may have several undesirable consequences to the organization, including loss of product market share or loss of customer or client confidence and satisfaction with the product.

On other occasions, the test team may attempt to implement a test tool too late in the development life cycle to adequately accommodate the test team's learning curve for the test tool. The test team may find that the time lost while learning to work with the test tool or ramping up on tool features and capabilities has put the test effort behind schedule. In such situations, the team may become frustrated with the use of the tool and even abandon it so as to achieve short-term gains in test progress. The test team may be able to make up some time and meet an initial test execution date, but these gains are soon forfeited during regression testing and subsequent performance of test.

In the preceding scenarios, the test team may have had the best intentions in mind, but unfortunately was simply unprepared to exercise the best course of action. The test engineer did not have the requisite experience with the tool or had not defined a way of successfully introducing the test tool. What happens in these cases? The test tool itself usually absorbs most of the blame for the schedule slip or the poor test performance. In fact, the real underlying cause for the test failure pertained to the absence of a defined test process, or where one was defined, failure to adhere to that process.

The fallout from a bad experience with a test tool on a project can have a ripple effect throughout an organization. The experience may tarnish the reputation of the test group. Confidence in the tool by product and project managers may have been shaken to the point where the test team may have difficulty obtaining approval for use of a test tool on future efforts. Likewise, when budget pressures materialize, planned expenditures for test tool licenses and related tool support may be scratched.

By developing and following a strategy for rolling out an automated test tool, the test team can avoid having to make major unplanned adjustments throughout the test process. Such adjustments often prove nerve-racking for the entire test team. Likewise, projects that require test engineers to perform hundreds of mundane tests manually may experience significant turnover of test personnel.

It is worth the effort to invest adequate time in the analysis and definition of a suitable test tool introduction process. This process is essential to the long-term success of an automated test program. Test teams need to view the introduction of an automated test tool into a new project as a process, not an event. The test tool needs to complement the process and not the reverse.

### **Test Process Analysis**

The test team initiates the test tool introduction process by analyzing the organization's current test process. Generally, some method of performing test is in place, and therefore the exercise of process definition itself may actually result in process improvement. In any case, process improvement begins with process definition. The test process must be documented in such a way that it can be communicated to others. If the test process is not documented then it cannot be communicated or executed in a repeatable fashion. If it cannot be communicated or is not documented then a process is often not implemented.

In addition, if the process is not documented then it cannot be consciously and uniformly improved. On the other hand, if a process is documented it can be measured and therefore be improved.

If the organization's overall test process is not yet documented, or it is documented but outdated or inadequate, the test team may wish to adopt an existing test process or adopt an existing test process in part. The test team may wish to adopt the Automated Test Life-cycle Methodology (ATLM), outlined in the book "Automated Software Testing," as the organization's test process. When defining or tailoring a test process, it may prove useful for the test engineer to review the organization's product-development or software-development process document, when available.

### Process Review

The test engineer needs to analyze the existing development and test process. During this analytical phase, the test engineer determines whether the current testing process meets the defined prerequisites listed below.

- Testing goals and objectives have been defined
- Testing strategies have been defined
- Tools needed are available to implement planned strategies
- A testing methodology has been defined
- The testing process is communicated and documented
- The testing process is being measured
- The testing process implementation is audited
- Users are involved throughout the test program
- Test team is involved from the beginning of the system development lifecycle
- Testing is conducted in parallel to the system development lifecycle

- Schedule allows for process implementation
- Budget allows for process implementation
- Understand whether the organization is seeking to comply with industry quality and process maturity guidelines (i.e. CMM, ISO)

The purpose of analyzing the organization's test process is to identify the test goals, objectives and strategies, which may be inherent in the test process. These top-level elements of test planning are the cornerstones for which a project's test program develops. The purpose of documenting the test tool introduction process is to ensure that the test team has a clearly defined way of implementing automated testing, so that the team can fully leverage the functionality and time saving features of the automated test tool.

The additional time and cost associated with the documentation and implementation of a test tool introduction process is sometimes an issue. A well-planned and well-executed process will pay for itself many times over by ensuring a higher level of defect detection and fielded software fixes, shortening product development cycles, and providing labor savings. A test team, which can be disciplined in defining test goals, and can reflect the test goals within defined processes, the skills of test team staff, and the selection of a test tool, will perform well. It is this kind of discipline, exercised incrementally, which supports the test team's (and the entire organization's) advancement in quality and maturity from one level to the next.

### **Safeguard Integrity of the Automated Test Process**

To safeguard the integrity of the automated test process the test team needs to exercise new releases of an automated test tool in an isolated environment; it can then validate that the tool performs up to product specifications and marketing claims. The test team should verify that the upgrades would run in the organization's current environment. Although, the previous version of the tool may have performed correctly and a new upgrade may perform well in other environments, the upgrade might adversely affect the team's particular environment. Therefore, the test team needs to make sure that the test of the test tool upgrade is performed in an isolated environment. Additionally, using a configuration management tool to baseline the test repository will help safeguard the integrity of the automated testing process.

Although process definition, metric gathering and process improvement activities can be expensive and time-consuming, the good news is that creating and documenting standards and procedures for an automated test program is no more expensive than the same activity for a manual test program. In fact, use of an automated test tool with scripting, test identification, and automatic documentation capabilities can reduce costs by providing some of the framework and content required.

### **Considering the use of a test tool on a particular project**

Once the test engineer has reviewed the test process and has defined test goals, objectives and strategies, the test engineer can decide whether to continue the consideration of using an automated test tool. Specifically, the test engineer seeks to verify that the previously identified automated test tools will actually work in the environment and effectively meet the system requirements.

### **➤ Review System Requirements**

The first step in test tool consideration is to review the system requirements. The test team needs to verify that the automated test tool can support the user environment, computing platform, and product features. If a prototype or part of the system-under-test already exists the test team should ask for an overview of the system. An initial determination of the specific sections of the application that can be supported with automated testing can be made.

### **➤ Review Project Schedule**

Next, the test schedule needs to be reviewed. Is there sufficient time left in the schedule or allocated within the schedule to support the introduction of the test tool? Remember that automated testing should ideally be incorporated at the beginning of the development life cycle. The project schedule may need to be adjusted to include enough time to introduce an automated testing tool.

### **➤ Manage Expectations**

During test tool consideration, the automated test tool should be demonstrated to the new project team enabling all pertinent individuals to gain an understanding of the tool's capability. Project team personnel in this case should include application developers, test engineers, quality assurance specialists and configuration management specialists. Remember that software professionals on the project may have a preconceived notion of the capabilities of the test tool, which may not match the tool's actual application on the project.

### **➤ Test Tool Compatibility**

If part of the application exists at the time of test tool consideration, conduct a test tool compatibility check. Install the testing tool in conjunction with the application and determine whether the two are compatible. One special concern is the availability of memory to support both the application and

the automated test tool. Another concern is the compatibility of 3<sup>rd</sup> party controls (widgets) used in the application. Once a compatibility check has been performed and a few problems arise, the test team will need to investigate whether work-around solutions are possible.

### ➤ **Define Roles and Responsibilities**

The use of automated test tools with a particular application requires the services of a test team that has the appropriate blend of skills to support the entire scope of the test effort. Roles and responsibilities need to be clearly defined and the skills and skill levels of test team personnel need to be considered carefully.

### ➤ **Technical Expertise**

Another element of test tool consideration relates to the need to determine whether the test team has sufficient technical expertise to take advantage of the tool's capabilities. If this technical expertise is not resident within the test team, individuals who can mentor the test team on the advanced features of the test tool might be applied to the project on a

short-term basis. Another possibility is test tool training for all test team personnel.

After completing the test tool consideration phase, the test team can perform the final analysis necessary to support a decision of whether to commit to the use of an automated test tool for a given project effort.

Once the test team has concluded that the test tool is appropriate for the current project, it continues with the ATLM by performing *Test Planning*, *Test Analysis & Design*, and *Test Development*. The outcomes of activities performed in this discussion need to be recorded as part of the test planning activities that are documented within the test plan.

---

*This text is adapted from "Automated Software Testing" (sections of Chapter 4), copyright AWL, all rights reserved, ISBN 0-20-43287*

## **ORDBMS or ODBMS? Performance on Complex Data is the Key**

Douglas K. Barry  
Barry & Associates, Inc  
www.barryandassociates.com

---

In the whirlwind of fast-paced technological change that we live with every day, it's sometimes hard to make choices. One of the choices that seems to be difficult is the choice between an object database management system (ODBMS) and an object-relational database management system (ORDBMS). I believe that the choice becomes easier if you learn about the technologies and then focus on the business needs that exist. In particular, you must understand the complexity of your data and the impact this complexity could have on application performance. In this article, I will compare the characteristics of the two systems, define scenarios that highlight when one choice might be better than the other and finally outline the impact that such a decision might make on performance.

Before we can answer the ODBMS vs. ORDBMS question, some basic information must be covered. To get to an answer, you must first understand the nature of complex data, the issues raised by impedance mismatch, and the implications that SQL-99 raises. Here is the short course on these topics.

### **Identifying Complex Data**

Whether or not your data is complex is your first consideration. A careful analysis and a deep understanding of your data are necessary to determine its complexity. This is essential to an appropriate database choice. Although complex data is everywhere, it can be hard to identify in some situations. Data that appears to be simple can be complex. The opposite can occur too. It's important to know what the attributes of complex data are. They include lack of unique

identification, many-to-many relationships, access using navigation, and frequent use of type codes.

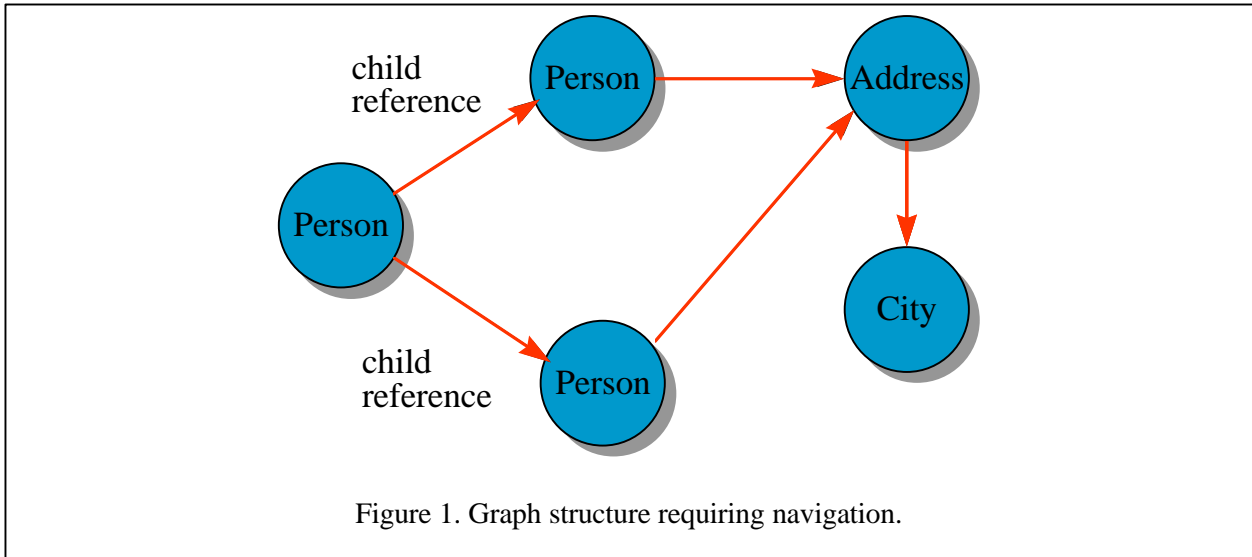
The first characteristic of complex data is the lack of an intrinsic, unique identification. Many items lack this unique identification. For example, what is the unique identification of a street address? The answer is the entire address. Addresses do not have intrinsic, unique identification.

Data with many-to-many relationships is a second sign of complex data. An example of a many-to-many relationship is a high school where each student has many teachers and each teacher has many students.

Access using navigation is yet another sign of complexity. A graph structure often indicates complex data because it shows that you must navigate a relationship between data instances. One example of a graph structure is provided in Figure 1. As is indicated in this figure, a child reference is used to navigate from a Parent Person to a Child Person. In a relational database management system (RDBMS), this navigation is often done by successive index searches or joins.

The frequent use of type codes is a final sign of complexity. Commonly, type codes are used to classify data in a relational schema. They often indicate that different processing must be done for each type within a hierarchy of types. An example of type code processing would be a database storing different types of employees, each requiring specific processing based on type codes.

These signs taken together indicate data complexity. As the complexity becomes greater, it becomes more and more difficult



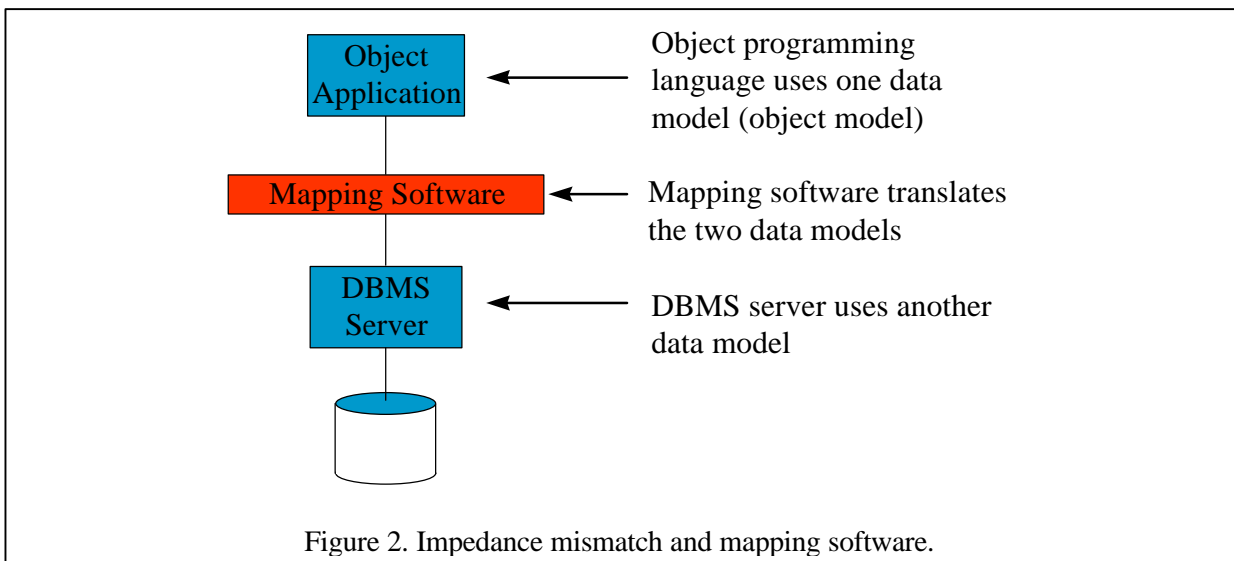
to translate the data from a relational to an object schema. The term for this difficulty is impedance mismatch. As the impedance mismatch becomes greater, the performance of an application falls lower and lower.

**The Problem of Impedance Mismatch**

As mentioned above, impedance mismatch occurs when the data is used in one format in the application and stored in another format by the DBMS. This can have a huge impact on performance. If programming done in an object programming languages such as C++ or Java, the application is using the object model of the programming language. On the other hand, if the DBMS does not support

stored in a different format by the DBMS. In that situation, mapping software is used to convert from one format to another. The size of the problem that this creates is dependent upon the amount of data and its complexity. Figure 2 illustrates impedance mismatch and the function of the mapping software.

Not only does impedance mismatch decrease performance on complex data; it also increases development costs. Because there is one model in the application and another in the DBMS, both models must be developed. In addition, you must decide how they will be mapped from one to the other. This can mean you need one or more additional people to develop the DBMS model, in addition to the application model.



the same object model, the data must be

**Issues about SQL-99**

The SQL-99 standard is an extension of SQL-92 with object characteristics. Because all RDBMSs will be supporting SQL-99 to some degree, this makes them all ORDBMSs. This is why, for the purposes of this article, I include products commonly considered RDBMSs as ORDBMSs.

Although ORDBMSs universally will be supporting SQL-99, there is one significant issue with SQL-99. That is that the SQL-99 object model does not match the object model of languages such as C++ and Java. The reason for this mismatch is that an overriding requirement in the development of SQL-99 was that it be compatible with all of SQL-92. The difficulty here is that SQL-92 uses a relational model, which means that the SQL-99 object model is distorted. In other words, the object model for SQL-99 will never perfectly match the object model used by C++ or Java. The bottom line here is that, although SQL-99 does have object characteristics, impedance mismatch is still present.

In short, using SQL-99 means you will have impedance mismatch with object programming languages. Your application will have one object model and your DBMS will have a different object model.

**A Comparison of RDBMSs, ORDBMSs and ODBMSs**

Impedance mismatch can have a significant effect on performance when using complex data. In the private benchmarks of RDBMSs versus ODBMSs that I have seen, the ODBMSs have been found to run 100 to 1000 times faster than the RDBMSs.

Will the difference be as great with ORDBMSs? At this point it's hard to say. I have not seen any benchmarks. Might the difference in performance drop an order of magnitude and the ODBMSs will “only” be 10 to 100 times faster? Or the ORDBMSs might reduce the disparity even further. What is certain is that there will still be some impedance mismatch. This is because of the adherence to the SQL-99 model. ODBMSs, on the other hand, do not have any impedance mismatch with languages such as C++ and Java. Figure 3 illustrates, at a very high level, the differences in the architectures and the amount mapping software needed because of impedance mismatch.

What might this mean for your application? If you are navigating graph structures such as the one shown in Figure 1, an ODBMS will give you much better performance. Don't just take my word for it. Michael Stonebraker, when he was the Chief Technical Officer at Informix Software, seemed to agree. During a question and answer session at the

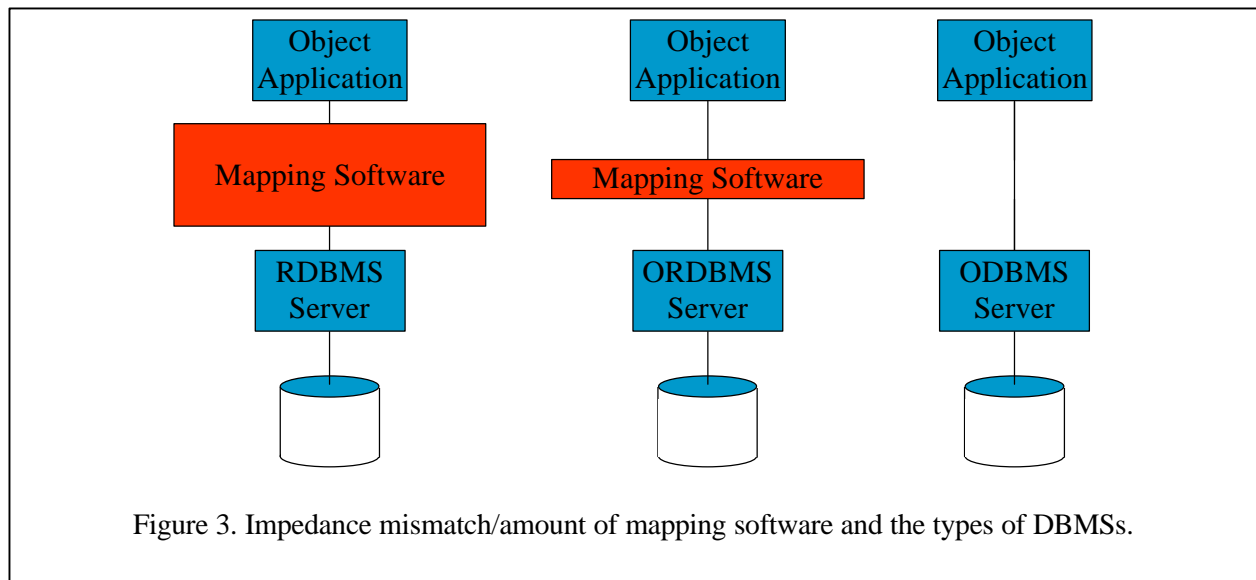


Figure 3. Impedance mismatch/amount of mapping software and the types of DBMSs.

Object/Relational Summit in Boston in August of 1997, he stated that, "My point of view is that if you want to navigate your objects in C++, object databases do that blindingly fast – way faster than a relational or an object-relational engine." He went on to say that if you want to use objects and SQL, you need an ORDBMS. Let's examine this assertion.

My understanding of Dr. Stonebraker's comment is that he believes you need SQL-99 or some subset of SQL-99 to do queries. In point of fact, however, SQL-99 is not the only way to perform an object query. Another method is to use the Object Query Language (OQL) developed by the Object Data Management Group (ODMG). OQL is based on SQL-92. But when the ODMG was developing OQL, the primary goal was not compatibility with all of SQL-92. This is the difference between OQL and the query portion of SQL-99. OQL follows SQL-92 until the relational model gets in the way of the object model and then the object model takes precedence in OQL. The ODMG worked with the SQL-99 committee to create one query language and changes were made to bring the two query languages together. As SQL-99 has moved through its adoption process, however, it was not possible to create the desired single query language and therefore SQL-99 and OQL remain different.

Here's an example of what an ODMG OQL query looks like:

```
select c.address
from Persons p,
     p.children c
where p.address.street="Main
Street"
```

In this query, all children of all "Persons" are inspected. The navigation proceeds from the Person class, using the child reference, to another instance of the Person class and then to the Address class and returns addresses on Main Street. This query uses the structure shown in Figure 1.

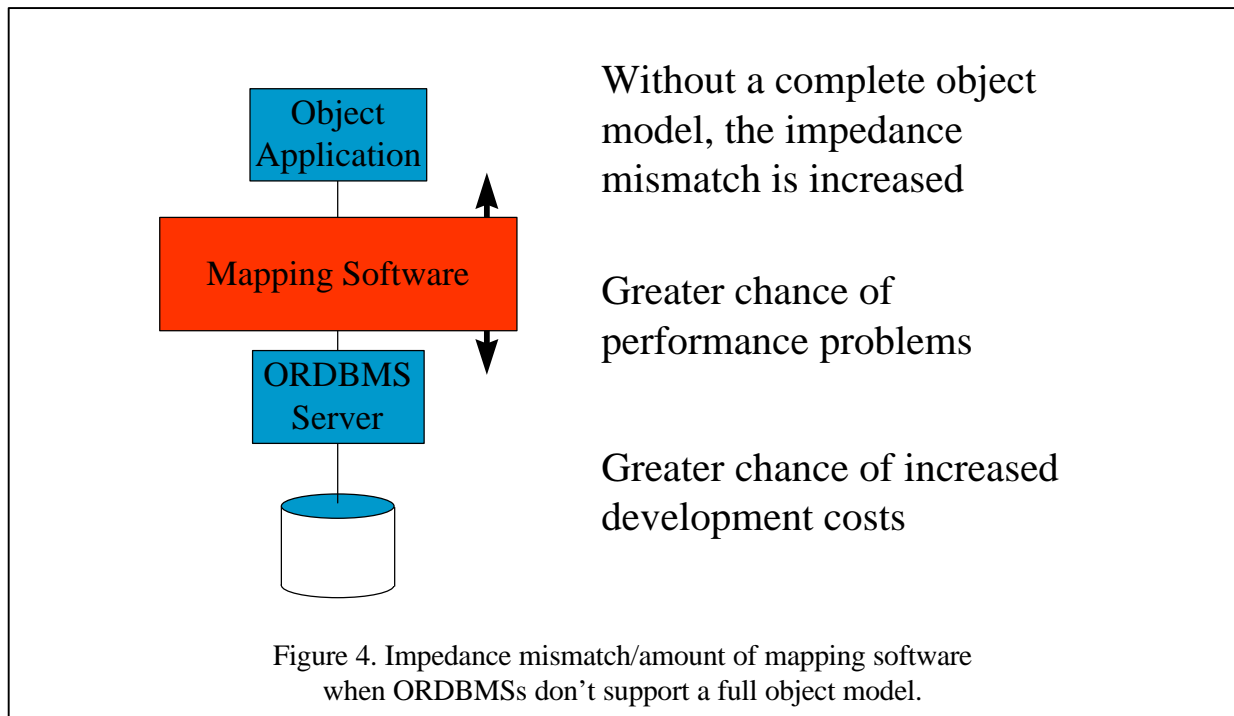
In comparing SQL-99 and OQL, it's important to remember that SQL-99 is more than a query language whereas OQL is only a query language. SQL-99 includes data definition and data manipulation languages. So how are data definition and data manipulation languages handled in the ODMG standard? The ODMG specification includes an Object Definition Language that is a superset of the Object Management Group's IDL for CORBA. Object programming languages are used for data manipulation through the use of tight language bindings in the ODMG specification. So, the ODMG standard builds on standards provided by others. The ODMG data definition language is from the OMG and the ODMG data manipulation is from the object-programming languages.

### **Current ORDBMS Products**

How well do the ORDBMS products that are currently on the market support object models? Some products have incomplete object models; this increases the impedance mismatch and potential performance problems. One example is Oracle8i, which supports references but not inheritance in its model. Another example is The Informix Dynamic Server, which supports inheritance but not references. This leads to performance problems because of the impedance mismatch. Figure 4 illustrates the effect of incomplete object models. As the ORDBMS products mature, however, this impedance mismatch likely will be reduced. Adherence to the SQL-99 model means, however, that it will not go away.

### **Choosing an ORDBMS**

An ORDBMS can be a good choice. One situation where this is true is if you have an existing application or applications and want to add some object characteristics. If your data is not very complex and impedance mismatch won't be a big problem, you have another situation. You might also decide to take a different approach and solve your impedance mismatch problem with new hardware. The additional hardware may allow



you to continue using an ORDBMS that you have in-house.

**Performance and Complex Data**

What type of DBMS is needed when data is complex and high performance is needed? An ODBMS is the performance choice in that situation because impedance mismatch won't be a consideration. You can get the maximum performance with object-programming languages such as C++ and Java. New development with complex data is a good situation for use of an ODBMS.

**A "Safer" Choice?**

Although the ORDBMS choice may seem safer to some, it might not be if you are looking for good performance on complex data. The object-relational name even seems to imply a simpler transition. A good decision, however, can be found in your data and application(s). If you merely want to add some object characteristics to a current application in order to improve performance or if your data is not too complex, ORDBMSs might be a good choice.

If you are talking about a new application that needs high performance and if your data

is complex, the ORDBMS choice probably is not the safer one. Impedance mismatch can wreak havoc with performance in those situations. High performance on complex data for most of us is becoming a basic business need. So to obtain a business solution that meets the needs of your data and application, get to know your data well. Use that knowledge, not the perceived safety of one type of product over another, as the basis for your decision.



## Companies

### Wang Global Bought by Getronics

Consolidation continues in the field of IT services, that is the place where the real money is made (it is not selling boxes...). But for once, the eater belongs to Europe and the (consenting) victim to the US. Even if Wang Global is the result of the marriage of Wang and Olsy, the service arm of Olivetti, and makes the most of its sales in Europe. The new Getronics will have around 33'000 employees in 40 countries with revenues of about 5 billion US dollars. Who remembers that only 6 years ago Wang was a computer maker?



### In Others' Words

"Most Web sites are built on top of spit and chicken wire, and we spend an inordinate amount of time and energy keeping them up. [...] Organizations set out to build a doghouse, but wind up building a skyscraper made of 10,000 doghouses. It's the organizations that set out to build a skyscraper that will ultimately succeed."

Source: Grady Booch, [www.infoworld.com](http://www.infoworld.com), May 11, 1999

Grady Booch thinks that web developers should design their sites around a component architecture based on the UML. This is another piece added to the debate about whether Web development is different from "traditional" software development. On one hand, people are saying that software development will always remain the same and that there is a need to "engineer" software. On the other hand, opponents declare that with the time constraint of the Internet, time to market is the key factor and that there is not enough time to stabilize and architecture applications. It could be true, the loss of image resulting from a Web site "shut-down" caused by software problems is also to be considered. We end up with the usual question: when is faster too fast? The answer is simple: when you fail, idiot!



### Coming next in Methods & Tools

- Software process improvement towards business improvement
- Components Software Enters The Mainstream
- 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 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](mailto:franco@martinig.ch)

---

**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](http://www.martinig.ch)  
Editor: Franco Martinig; e-mail [franco@martinig.ch](mailto:franco@martinig.ch)  
Free subscription: [www.martinig.ch](http://www.martinig.ch)  
The content of this publication cannot be reproduced without prior written consent of the publisher  
**Copyright © 1999, Martinig & Associates**

---