
METHODS & TOOLS

Global knowledge source for software development professionals

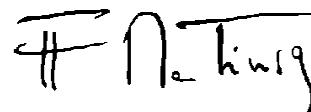
ISSN 1023-4918

Winter 1999 (Volume 7 - number 4)

People and Organization: It is all About Willing to Share

Often the initial journey as a software developer is traveled alone. At home or at school you were sitting in front of your screen, mastering the exciting project of producing an "Hello World" window or stack management procedures. Can you remember the happiness of making things happen on your own? Perhaps some of you are still operating on a stand-alone basis, but most of you are working in teams, sharing a project with other software developers. This is the moment when the individual parts of software development meet the Organization and sometimes the organization. The Organization with an upper case 'O' is what you are working for, the organization with a small 'o' is the way you collaborate with other software developers. This second item is the main subject of this issue. Collaborating in an individualistic job like programming is not easy. Despite talks about "software engineering", you have to recognize that the "design" content is strong in our profession... and design is a matter of taste. In opposition to "classical" manufacturing, most of the design-related aspects in software development are often not targeted to consumers, but they are rather only visible by co-workers, either in the development or in the maintenance phase. There are a lot of books about good design and coding styles and it is worthwhile to read them, but every developer has its own style. You have to accept this, because you know that even you, at different times, will have another solution to a specific problem or requirement.

Working together is not only about development style, it is a matter of being able to understand each other design and code, also with the help of comments, and to accept different solutions. It is to be sure that modules will communicate and that the meaning of interface variables is well defined. It is to know where to find a particular piece of code dealing with a specific subject. It is to know that you have to go beyond your individual vision of your work, because you cannot succeed alone in projects and you will be glad to have colleagues that share the same concerns than you. There is nothing better in the Organization than colleagues happy to collaborate with each-other...



Inside

Management: Object Oriented Group Work, Matti Halme.....	page 2
Web: The WebSite Quality Challenge.....	page 11
Facts, News & Comments	page 16

Object Oriented Group-Work

Matti Halme, Stonesoft Corporation, www.stone.fi
Taivalmäki 9, 02200 ESPOO, Finland

As the size of an information system project grows and the number of human resources needed increases, significance of effective and efficient group-work becomes a very essential factor in the success of the project. Does object orientation have anything to do with this?

Introduction

Have you ever thought what happens if the cooperation between the team-members more or less fails in an information technology project?

First, of course, inefficient teamwork results in inefficient total effort. Results of individual team-members cannot be adequately used by others and excessive rework occurs. Group meetings and other communication activities may then also take lots of extra time.

Second, there may be potential quality problems since technical and functional compromises are forced to be made in absence of rational agreement in important issues.

Third, ineffective group-work may set a practical limit to the size of the system that can be successfully implemented no matter how many people work in the project.

Prerequisites of successful group-work in information technology projects sure are largely same than those of any team-oriented projects.

Of course, there must exist a common goal that everybody is trying to achieve. Features and characteristics of the system to be built must be known as clearly as possible and there must be a common agreement upon them. High-level goals or visions of 'what are we ultimately trying

to accomplish' must also be clear and common among the team-members.

Additionally, few essential teamwork factors are division of work, communication and - especially in IT projects - integration.

Division of work

There must be good division of work between the team-members. Everything needs to get done and people must be able to get their own field of work they are motivated to work in and which does not conflict with responsibilities of other members. The division must be clear, well defined and understandable.

There must be a clear understanding of what are the responsibilities of each team-member and what are the dependencies between them. When planning the division of work, the inter-person dependencies should be minimized and the project management and team-members should be aware of the remaining ones. A question naturally rising of the inter-dependency issues is how well can the project be divided into largely independent tasks.

Additionally, of course, a good division of work must utilize the skills and expertise of the team-members.

A good division of work is one that everybody can stick to! People must take the responsibility of performing their tasks. On the other hand, the team-members must be very careful when getting involved to responsibilities of other team-members. It is often quite tempting to do some fixes to a colleague's design to make it better 'fit' to one's design and idea of the system. Of course, feedback should be given to make things work, but basically team-members should

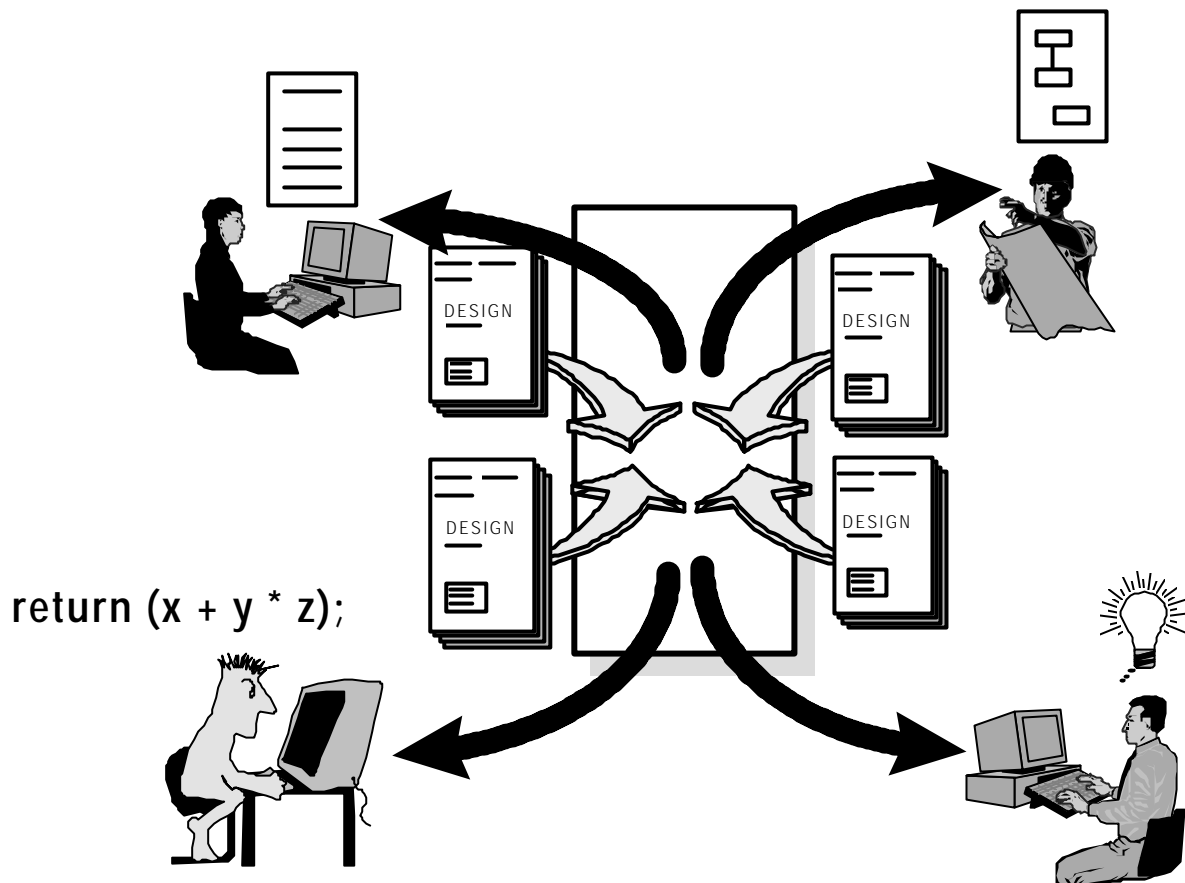


Figure 1. Communication in IT group-work through common datastore.

stick to their responsibilities, and trust other people are doing their jobs.

Communication

Communication between the team-members must work. Of course, communication between human beings can not work without expressing skills and ability and will to listen and receive information. Additionally, for a successful communication, all team-members must be able to talk the same language.

They must use the same terms and a same frame of reference that gives the terms unambiguous meanings.

At certain point of technicality and complexity, when things are hard to express verbally, various more or less formal modeling and

diagramming techniques can help. However, in order to a technical notation to be useful and cost-effective, it must be commonly understandable among the team-members and other affected parties. It must also be easy to use. Of course, the notation must have enough expressive power to cover the problem domain.

Typical communication methods in any teamwork include direct communication by talking and sending e-mail messages and different kinds of meetings. This direct communication is necessary in all cooperation, but when dealing with complex things, indirect communication using some kind of common datastore - or just a set of written specification documents - becomes essential. Trying to communicate complex highly technical structures and designs in project meetings can sometimes be a total waste of time for everybody. People

need to write it all down and get others to read it!

The idea represented in figure 1 is that depending on their background and exact profession, people have tendency to think about and express similar things in very different ways and in different abstraction levels. In datastore-centric communication, there is a standardized way in which the specifications are presented. In the simplest form, the datastore consists of documents all written using common templates specifying and instructing people on both contents and format of specifications.

Datastore-centric communication means that information can be produced and consumed asynchronously, at any time. A prerequisite to make datastore-centric communication work is that people produce standardized and complete information, keeping it up-to-date.

Integration

To get a working information system, the results of individual team-members somehow need to be brought together. In IT terms this is called integration and, as many of us know, often means that modules people have written are linked together and the system is then expected to start working as whole.

Sometimes the integration can be carried through with minor modifications to the system, sometimes all the misunderstanding and incompatible interfaces pop up at once and the integration effort appears to greatly exceed the effort of writing the individual modules. In the latter case, term *effort of integration* is actually wrong. We should talk about the effort of understanding and defining the dependencies and interfaces, all too late.

It is clear, that so-called *big-bang integration* introduces great risks in large projects. Integration needs to be started at very early phases of projects. Ways to validate compatibility between early software life-cycle

products such as analysis and design specifications should be considered.

So, does the object orientation help?

Could the object oriented paradigm be used to improve the cooperation of team-members in software engineering projects? Do we get better divisions of work, easier communication and free integration? Sure, OO tool vendors have promised us largely independent units, called classes, with data and functionality combined and encapsulated. There is the division of work: each designer just picks the classes he/she will design and implement. These units only interact through well-defined interfaces, which must lead to easy integration. We also have commonly understood notations that make people talk the 'same language'. Communication must now be easier.

But what have we got? The OO paradigm, notations, methods and tools do not really automatically bring minimized dependencies and well-defined interfaces, better modularity in more traditional words. Sometimes they deteriorate the modularity of your design by suggesting you to follow design principles of a popular application framework targeted to different kinds of applications than yours.

The variety of different OO methods and notations bring IT specialists difficulties in learning and understanding object orientation. Understanding new notations has proven to be especially hard to customers not specialized in information technology.

Does object orientation then have anything to do with improving the information system teamwork? In my opinion, yes. How should we use object orientation to get the teamwork better organized? I believe that the answer is in using OO as a vehicle in designing good, modular software architectures, writing good, understandable object oriented specifications and systematically applying object oriented methods from the very beginning of the software life-cycle.

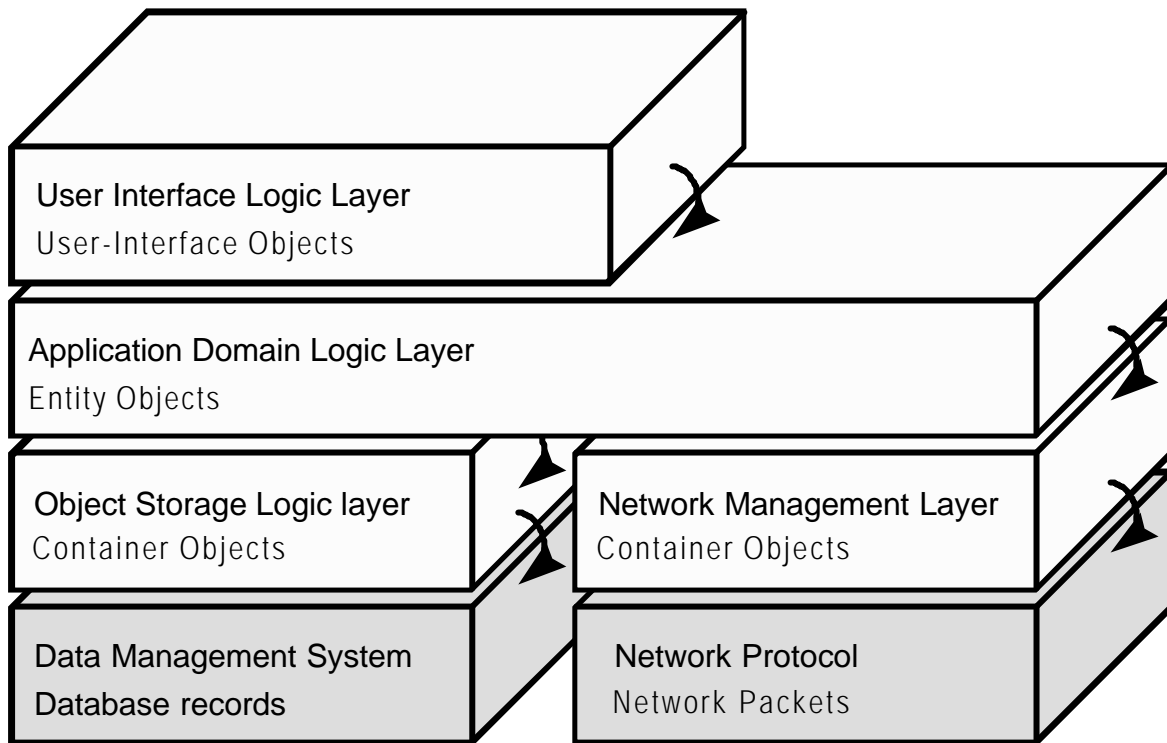


Figure 2. An example of a layered application architecture

Next, we will look at some constructs that lead towards more modular software architectures. Then we will show how to organize our common datastore containing both object oriented and more traditional information. Finally, we will examine how to consider the integration issues from the beginning of the project.

What about layered architecture?

Let us start with the modularity issues. Since object orientation itself does not lead to better modularity, we should explicitly try to design modular software architectures. The first step in using object orientation to design modular software architectures can often be taken simply by aiming at a layered architecture.

In layered architecture, the system is divided into parts that do not have any cyclic dependencies between them. The upper layer knows about the layer below and can use

services offered by the lower layer. The lower layer in turn does not depend from the upper layer in any way. In fact, it does not even need to know what layers exist above it.

In object orientation, using services often happens through one-way-navigable associations between objects belonging to classes at different layers.

It's characteristic to each layer that the technologies or fundamental concepts used are somehow different. Layered architecture can thus be called vertical or technical partitioning of the information system.

Using a layered architecture such as one illustrated in figure 2, is a way to natural, technology-based division of work. Persons specializing in database or network technologies are responsible of design and implementation of the lower layers. The application domain specialists can concentrate to their expertise areas without having to worry about any

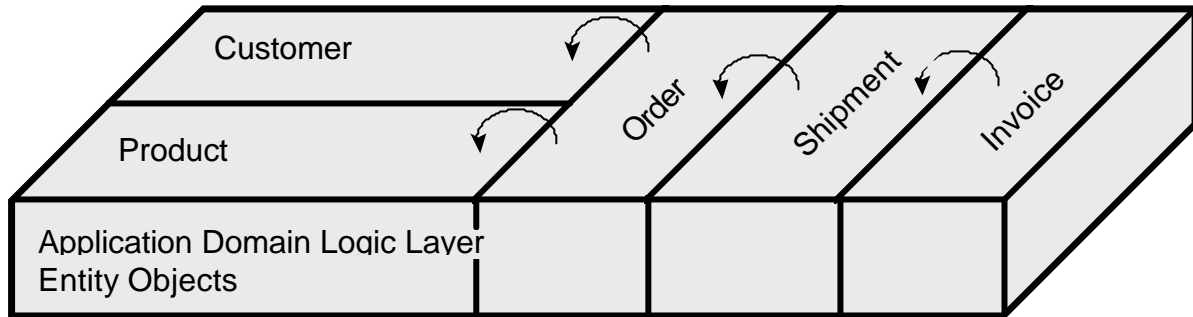


Figure 3. A layer partitioned into domains

database technologies or network protocols. Neither do they need to know about how the information is presented to the user with graphical user-interface. Responsibilities left to the UI designer are the user-interface logic and the graphical layout.

Layered architecture has not been invented with the object orientation. Examples of layered architectures existed long before the object oriented systems were popular. Any application interfacing with the application programming interface (API) of an operating system is an example of layered architecture. Operating systems themselves often tend to be very layered. However, object orientation gives us perfect tools to model and implement layered architectures.

Slicing the layers

If each layer in a layered architecture is responsible for one technical concept, dividing the application according to functionality it is responsible for leads to horizontal or functional partitioning. Many layers can - and they often should - be divided into parts, called domains for example. Each of the domains represents certain well-defined, limited functionality. Clear interfaces and well-understood non-cyclic dependencies should exist between the domains.

In a familiar sales and invoicing system example, the application domain logic layer could be divided into domains presented in figure 3.

Each domain in turn consists of number of relatively closely related classes, which are the smallest units of modularity in object orientation. The *Order* domain could have classes such as *Order* and *OrderLine*. Domains in different layers typically correspond to each other. The *Order* domain in the application domain layer is associated to *Order user-interface* domain in the user-interface layer.

Horizontal partitioning can be used to target a clear application domain knowledge-based division of work. Horizontal - like vertical - partitioning forces you to think about dependencies and interfaces when designing the application architecture.

When should we go to object oriented application frameworks?

When several applications with similar kind of application logic, functionality and technical solutions - possibly in the same application domain - are produced, the need to reuse parts of applications rises. Separating common functionality, logic and technical solutions into a common part shareable among several applications is called constructing an application framework.

The difference between application frameworks and other reusable constructs, such as class libraries, is that frameworks also contain common logic of functionality. Typically,

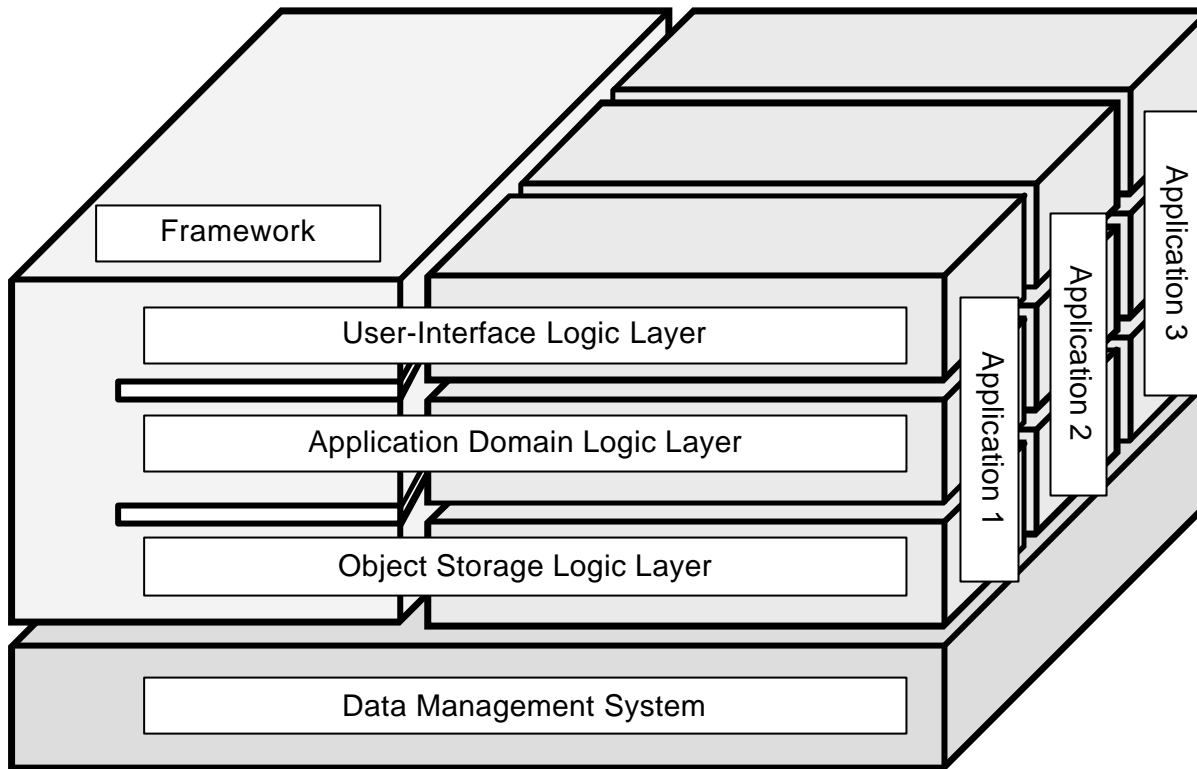


Figure 4. An example of object oriented application framework and three applications using it. Network management layers and inter-layer dependencies shown in previous figures have been left out to simplify the illustration.

frameworks are responsible of the main logic of the application. In graphical user-interface applications, this means in practice that the framework handles the main event loop passing the appropriate messages to the application as callbacks.

The object oriented paradigm can very well be applied into application framework architectures. Handling the application logic in the framework can be done by defining certain base classes in the framework. These classes interact with other framework classes. The application-specific functionality is implemented in classes inherited from the framework base classes. Calling overridden methods in the application classes is used as a callback-mechanism. Base classes that define the application logic are often called protocol classes. Framework classes should of course also implement common basic operations and

solve demanding technical problems, thus making application development easier.

The problem with the application frameworks is that they are very complex to design, implement and maintain. The complexity naturally rises from the fact that a framework must act in a common way like several real-world applications expect it to do. Constructing frameworks hardly ever succeeds at the first shot. This iterative process has to be performed in parallel with some pilot applications using the framework. The construction of framework requires to use the most talented and experienced resources.

In object orientation, using services often happens through one-way-navigable associations between objects belonging to classes at different layers.

WebBoard 4.0

Quick & Easy Community Building From O'Reilly!

Online forums and discussions are a must in today's Web-based environment. Whether you're building virtual offices of classrooms, corporate intranets, commerce, or online communities, **WebBoard 4.0** can organize, enhance, and improve communication.

- Create a central location for online communications, discussions, and email
- Customize "boards" for your own look and feel
- Run up to 255 boards with unlimited users and 1,000-user simultaneous chat.

It's fast, powerful, and scalable. It's the original WebBoard from O'Reilly. Visit us at www.webboard.com/nl2 and find out why WebBoard is the world's leading discussion and chat software.

<http://www.webboard.com/nl2>



Putting it all into a repository

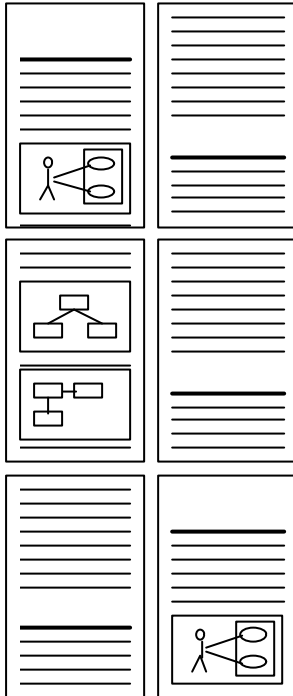
Using object orientation as a tool to achieve highly modularized software architectures with well-known dependencies and clear interfaces helps making a good division of work in software projects. What about communication and integration? Object orientation can offer some help in these issues too.

Using a common datastore as a communication method for complex structures requires a comprehensive, widely understood notation to express things. Traditional textual specifications often make it hard for a reader to get a good overall view of the system, or even a part of it. For an author, it might be hard to get convinced that everything essential is included into the document.

More formal specifications such as object oriented notations, if used systematically, guide the author to a more complete and consistent representation. However, object oriented methods and notations seldom are themselves comprehensive enough.

In most cases, you need both structured and textual presentations consistently completing each other. A good object oriented design must be able to be presented as a readable document. A typical way to compile a specification from textual descriptions and more or less object oriented graphical notations is to embed the diagrams created by a graphical software or a CASE tool into a text document. This is a very natural approach, since pictures and drawings are very commonly used in all kinds of technical documents to illustrate the textual presentation.

A document as the
master specification



The repository as the
master specification

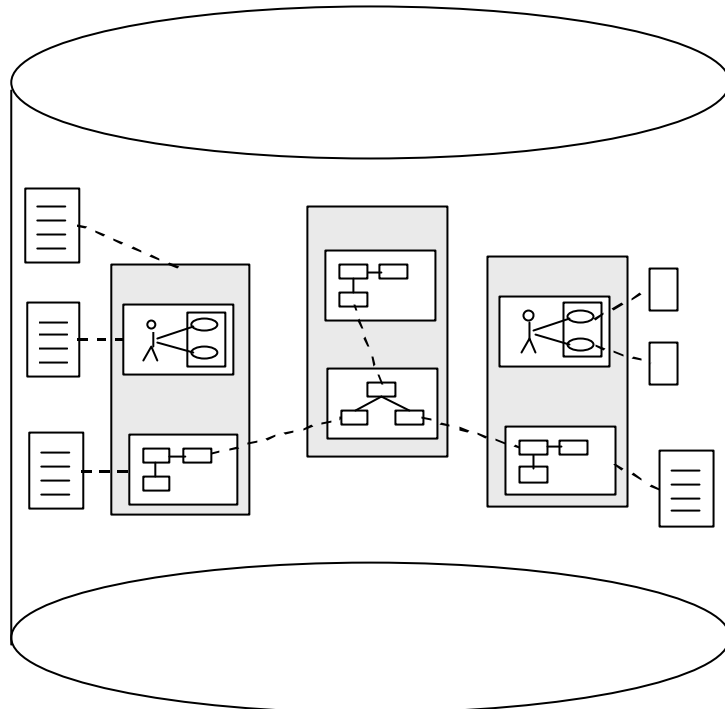


Figure 5. Document versus repository-driven organisation of specifications.

However, when the text document acts as the master-specification, there is a danger that the so-called object oriented design becomes a set of separate, inconsistent pictures. Maintaining the object model and its consistency to the textual documentation becomes hard and error-prone.

A repository is a systematically organized database of design information. An object oriented repository is a database for object models containing logical, graphical and textual information. Advanced repositories are designed for multiple users having authentication, access-control and data-locking properties. They also support versioning and references between different models to allow the models to be partitioned according to the partitioning of the information system.

The idea of using a repository-based solution as a common datastore in information system design is that the repository can act as the master specification for the system. Everything is organized around consistent object models. Textual parts of the specification are written as 'mini-documents' that are stored into the repository and linked to the corresponding elements of the object models.

Using an object oriented repository as the master specification for an information system compared to a more traditional approach is represented in figure 5. Of course, paperless offices do not exist and at some point, the specification needs to be printed. The content of the repository thus needs to be able to be represented in a sequential form as a single document. However, this can be considered as highly mechanical task and can be automated.

Additionally, other representations, such as hypertext documentation, can be derived automatically from the repository.

What about parts of the specifications that do not have anything to do with objects? Examples of this kind of parts include performance characteristics and other general properties of the system.

Yes, that kind of specifications exists and they are very vital to an information system. A specification should consist of mainly two parts: textual general part and object oriented 'reference' part. In a repository-driven approach, the general part is stored into the repository as a document.

Object oriented integration

As mentioned earlier, leaving the integration of a software system too late, can cause unpredictable amounts of extra work, in the phase of the project when it is the least wanted.

A common way to avoid integration-risks is to 'grow' the software by integrating the new parts of the system to the previously implemented parts as soon as they have been implemented. In fact, many popular object oriented languages with strict type-checking strongly encourage programmers to integrate the software all the time.

This way, the integration is done incrementally and the implemented parts are kept consistent with each other all the time. Still, the integration is left to the implementation phase of the software life-cycle. Major interface-design flaws in the analysis and design phases may lead to a system that is impossible to integrate.

Continuous software integration - from analysis to design - means understanding and managing the interfaces and keeping the parts consistent with each other. Consistency of different parts of the system with each other should be validated as early as possible. It is generally easier to validate the consistency of the system if

it is modeled formally and exactly. The model of the system tends to be more formal and exact at the later phases of the software life-cycle. At the point when a system is specified as program source code, the model is very formal and exact. The compiler and linker will pretty much tell you if it is consistent or not. Of course, you need the integration testing to validate the semantic consistency too.

In other words, what you need to do, is to aim to more formal and exact specifications as early as possible. Consistency of object models can be largely validated mechanically. Systematic use of object oriented specifications and maintaining a consistent repository as the major datastore from the very beginning thus reduces the integration risks

Get organized

The object oriented paradigm itself does not lead to better group-work. However, object orientation, when carefully applied to good software architecture design principles and used with organized repositories, can help to achieve a better division of work, a better communication and it can reduce the integration risks.

Object orientation gives good tools to design modular software architectures and understand and manage the inter-module dependencies. You can start by aiming to layered architectures and further divide the layers into logical parts naturally derived from the application domain. Application frameworks are advanced software architectures that can be used to divide the development of a family of applications among several work-groups and still benefit from reuse. Improving the group-work is one more reason to design good modular software architectures in large systems.

Organization of design information based on object oriented repository directs the designers towards more consistent object models. Incorporating the textual specifications into the

repository as well makes the information more manageable and maintainable.

Systematically applying object oriented techniques and keeping the models as consistent as possible from the very beginning of the project can really pay itself back when the system is being integrated. In other words, we really need to think of minimized dependencies and good interfaces already in the early phases of the project. We really need to know how to write good specifications and maintain them. We really need to get organized in things that cannot work without a systematic approach. There is no substitute for good project management, talented system architects and good software designers, with or without object orientation.

*Originally published in the April 1998 issue
of the paper-based Methods & Tools*

The WebSite Quality Challenge

Edward Miller, miller@soft.com, Software Research, Inc., www.soft.com
901 Minnesota Street, San Francisco, CA 94107 USA

Because of its possible instant worldwide audience a WebSite's quality and reliability are crucial. The very special nature of the WWW and WebSites pose unique software testing challenges. Webmasters, WWW applications developers, and WebSite quality assurance managers need tools and methods that can match up to the new needs. Mechanized testing via special purpose WWW testing software offers the potential to meet these challenges.

Introduction

WebSites are something entirely new in the world of software quality!

Within minutes of going live, a WWW application can have many thousands more users than a conventional, non-WWW application. The immediacy of the WWW creates an immediate expectation of quality and rapid application delivery, but the technical complexities of a WebSite and variances in the browser make testing and quality control more difficult, and in some ways, more subtle. Automated testing of WebSites is both an opportunity and a challenge.

Defining WebSite Quality & Reliability

A WebSite is like any piece of software: no single, all-inclusive quality measure applies, and even multiple quality metrics may not apply. Yet, verifying user-critical impressions of "quality" and "reliability" take on new importance.

Dimensions of Quality. There are many dimensions of quality, and each measure will pertain to a particular WebSite in varying degrees. Here are some of them:

- **Time:** WebSites change often and rapidly? How much has a WebSite changed since the last upgrade? How do you highlight the parts that have changed?
- **Structural:** How well do all of the parts of the WebSite hold together. Are all links inside and outside the WebSite working? Do all of the images work? Are there parts of the WebSite that are not connected?
- **Content:** Does the content of critical pages match what is supposed to be there? Do key phrases exist continually in highly-changeable pages? Do critical pages maintain quality content from version to version? What about dynamically generated HTML pages?
- **Accuracy and Consistency:** Are today's copies of the pages downloaded the same as yesterday's? Close enough? Is the data presented accurate enough? How do you know?
- **Response Time and Latency:** Does the WebSite server respond to a browser request within certain parameters? In an E-commerce context, how is the end to end response time after a SUBMIT? Are there parts of a site that are so slow the user declines to continue working on it?
- **Performance:** Is the Browser-Web-WebSite-Web-Browser connection quick enough? How does the performance vary by time of day, by load and usage? Is performance adequate for E-commerce applications? Taking 10 minutes to

respond to an E-commerce purchase is clearly not acceptable!

Impact of Quality. Quality is in the mind of the user. A poor-quality WebSite, one with many broken pages and faulty images, with Cgi-Bin error messages, etc. may cost in poor customer relations, lost corporate image, and even in lost revenue. Very complex WebSites can sometimes overload the user.

The combination of WebSite complexity and low quality is potentially lethal to an E-commerce operation. Unhappy users will quickly depart for a different site! And they won't leave with any good impressions.

WebSite Architecture

A WebSite can be complex, and that complexity -- which is what provides the power, of course -- can be an impediment in assuring WebSite Quality. Add in the possibilities of multiple authors, very-rapid updates and changes, and the problem compounds.

Here are the major parts of WebSites as seen from a Quality perspective.

Browser. The browser is the viewer of a WebSite and there are so many different browsers and browser options that a well-done WebSite is probably designed to look good on as many browsers as possible. This imposes a kind of de facto standard: the WebSite must use only those constructs that work with the majority of browsers. But this still leaves room for a lot of creativity, and a range of technical difficulties.

Display Technologies. What you see in your browser is actually composed from many sources:

- **HTML.** There are various versions of HTML supported, and the WebSite ought to be built in a version of HTML that is compatible. And this should be checkable.

- **Java, JavaScript, ActiveX.** Obviously JavaScript and Java applets will be part of any serious WebSite, so the quality process must be able to support these. On the Windows side, ActiveX controls have to be handled as well.
- **Cgi-Bin Scripts.** This is link from a user action of some kind (typically, from a FORM passage or otherwise directly from the HTML, and possibly also from within a Java applet). All of the different types of Cgi-Bin Scripts (perl, awk, shell-scripts, etc.) need to be handled, and tests need to check "end to end" operation. This kind of a "loop" check is crucial for E-commerce situations.

Database Access. In E-commerce applications either you are building data up or retrieving data from a database. How does that interaction perform in real world use? If you give in "correct" or "specified" input does the result produce what you expect? Some access to information from the database may be appropriate, depending on the application, but this is typically found by other means.

Navigation. Users move to and from pages, click on links, click on images (thumbnails), etc. Navigation in a WebSite often is complex and has to be quick and error free.

Object Mode. The display you see changes dynamically; the only constants are the "objects" that make up the display. These aren't real objects in the OO sense; but they have to be treated that way. So, the quality test tools have to be able to handle URL links, forms, tables, anchors, buttons of all types in an "object like" manner so that validations are independent of representation.

Server Response. How fast the WebSite host responds influences whether a user (i.e. someone on the browser) moves on or continues. Obviously, Internet loading affects this too, but this factor is often outside the Webmaster's control at least in terms of how the

WebSite is written. Instead, it seems to be more an issue of server hardware capacity and throughput. Yet, if a WebSite becomes very popular -- this can happen overnight! -- loading and tuning are real issues that often are imposed -- perhaps not fairly -- on the webmaster.

Interaction & Feedback. For passive, content-only sites the only issue is availability, but for a WebSite that interacts with the user, how fast and how reliable that interaction is can be a big factor.

Concurrent Users. Do multiple users interact on a WebSite? Can they get in each other's way? While WebSites often resemble conventional client/server software structures, with multiple users at multiple locations a WebSite can be much different, and much more complex, than complex applications.

Assuring WebSite Quality Automatically

Assuring WebSite quality requires conducting sets of tests, automatically and repeatably, that demonstrate required properties and behaviors. Here are some required elements of tools that aim to do this.

Test Sessions. Typical elements of tests involve these characteristics:

- **Browser Independent.** Tests should be realistic, but not be dependent on a particular browser, whose biases and characteristics might mask a WebSite's problems.
- **No Buffering, Caching.** Local caching and buffering -- often a way to improve apparent performance -- should be disabled so that timed experiments are a true measure of the Browser-Web-WebSite-Web-Browser response time.
- **Fonts and Preferences.** Most browsers support a wide range of fonts and presentation preferences, and these should

not affect how quality on a WebSite is assessed or assured.

- **Object Mode.** Edit fields, push buttons, radio buttons, check boxes, etc. All should be treatable in object mode, i.e. independent of the fonts and preferences. Object mode operation is essential to protect an investment in tests and to assure tests' continued operation when WebSite pages change. When buttons and form entries change location -- as they often do -- the tests should still work. When a button or other object is deleted, that error should be sensed! Adding objects to a page clearly implies re-making the test.
- **Tables and Forms.** Even when the layout of a table or form varies in the browser's view, tests of it should continue independent of these factors.
- **Frames.** Windows with multiple frames ought to be processed simply, i.e. as if they were multiple single-page frames.
- **Test Context.** Tests need to operate from the browser level for two reasons: (1) this is where users see a WebSite, so tests based in browser operation are the most realistic; and (2) tests based in browsers can be run locally or across the Web equally well. Local execution is fine for quality control, but not for performance measurement work, where response time including Web-variable delays reflective of real-world usage is essential.

WebSite Validation Processes

Confirming validity of what is tested is the key to assuring WebSite quality -- and is the most difficult challenge of all. Here are four key areas where test automation will have a significant impact.

Operational Testing. Individual test steps may involve a variety of checks on individual pages in the WebSite:

- Page Quality. Is the entire page identical with a prior version? Are key parts of the text the same or different?
- Table, Form Quality. Are all of the parts of a table or form present? Correctly laid out? Can you confirm that selected texts are in the "right place"?
- Page Relationships. Are all of the links a page mentions the same as before? Are there new or missing links?
- Performance, Response Times. Is the response time for a user action the same as it was (within a range)?

Test Suites. Typically you may have dozens or hundreds (or thousands?) of tests, and you may wish to run tests in a variety of modes:

- Unattended Testing. Individual and/or groups of tests should be executable singly or in parallel from one or many workstations.
- Background Testing. Tests should be executable from multiple browsers running "in the background" [on an appropriately equipped workstation].
- Distributed Testing. Independent parts of a test suite should be executable from separate workstations without conflict.
- Performance Testing. Timing in performance tests should be resolved to 1 millisecond levels; this gives a strong basis for averaging data.
- Random Testing. There should be a capability for randomizing certain parts of tests.

- Error Recovery. While browser failure due to user inputs is rare, test suites should have the capability of resynchronizing after an error.

- Content Validation. Apart from how a WebSite responds dynamically, the content should be checkable either exactly or approximately. Here are some ways that should be possible:

- Structural. All of the links and anchors match with prior "baseline" data. Images should be characterizable by byte-count and/or file type or other file properties.

- Checkpoints, Exact Reproduction. One or more text elements -- or even all text elements -- in a page should be markable as "required to match".

- Gross Statistics. Page statistics (e.g. line, word, byte-count, checksum, etc.).

- Selected Images/Fragments. The tester should have the option to rubber band sections of an image and require that the selection image match later during a subsequent rendition of it. This ought to be possible for several images or image fragments.

- Load Simulation. Load analysis needs to proceed by having a special purpose browser act like a human user. This assures that the performance checking experiment indicates true performance -- not performance on simulated but unrealistic conditions.

- Sessions should be recorded live or edited from live recordings to assure faithful timing. There should be adjustable speed up and slow down ratios and intervals.

Load generation should proceed from:

- Single Browser. One session played on a browser with one or multiple responses.

Timing data should be put in a file for separate analysis.

- **Multiple Independent Browsers.** Multiple sessions played on multiple browsers with one or multiple responses. Timing data should be put in a file for separate analysis. Multivariate statistical methods may be needed for a complex but general performance model.
- **Multiple Coordinated Browsers.** This is the most-complex form -- two or more browsers behaving in a coordinated fashion. Special synchronization and control capabilities have to be available to support this.

Situation Summary

All of these needs and requirements impose constraints on the test automation tools used to confirm the quality and reliability of a WebSite. At the same time they present a real opportunity to amplify human tester/analyst capabilities. Better, more reliable WebSites should be the result.



Companies

No Viasoft for Compuware

The U.S. Justice Department said last October it planned to block the proposed merger between Compuware and Viasoft in court because the combination could result in higher software fees. Justice Department officials said they were concerned the merger would give Compuware a dominant hold over certain types of critical/mainframe software, testing/debugging software used to check for errors as program code is written.

Consolidation is the name of the game in the software tools industry, but things will not run as smoothly as liked by some big companies' executives. This decision could be linked to the current anti-trust Microsoft trial. Governments are now considering software like another industry and they are acting against dominant positions. This move could be labeled as limiting a free market economy, but it is also true that IT managers are worried when mergers and acquisitions gradually limit their options for specific products. Did I have to mention the recent acquisition of Platinum Technology by Computer Associates...

Sun sets in the East

After the recent acquisition of Forte, Sun announced in October the purchase of Prague Czech Republic-based Java tools vendor NetBeans. This 40-person company will be integrated in the Forte division and its products will be renamed Forte for Java, Community and Internet Edition.

This move allows Sun to offer a complete line of products to develop Java-based applications. It leaves however open the point of being at the same time the unique provider of Java specifications to software tools editors like IBM, Oracle or Inprise and a rival vendor of tools.

\$9%£ Numbers

Databases Ups and ERP Downs

The recent period of quarterly results announcement allows us to feel the current trends on the world software market.

In the database sector, both Informix and Sybase announced better than expected results with licensing revenues growing around 10%. After some terrible years, these two companies are getting better, fueled by data warehousing and Internet projects.

On the ERP Side, SAP said its net income fell 64% in its third quarter where revenues for this quarter rose only 7% from a year earlier. Baan suffered a \$25 million loss for revenues of \$143 million, minus 27% from the previous year. In this area, the Y2K effect and the transition to Internet are mentioned as causes for the problems. The fact is that many initiatives are changing the distribution and pricing models of ERP companies. External application hosting, leasing, user-based fees are "new" concepts that make the industry feel like 20 years younger...



In Others' Words

How to Manage Geeks...

"According to the traditional stereotype, geeks are people who are primarily fascinated by technology and its uses. The negative part of that stereotype is the assumption that they have poor social skills. Like most stereotypes, it's true in general – but false at the level of specifics. By society's definition, they are antisocial. But within their own community, they are actually quite social. You'll find that they break themselves into tribes: [...]. They're tribal in the way that they subdivide their own community, but the tribes don't fight each other. In fact, those tribes get along very well – because all of them fight management."

"Perhaps the least-becoming aspect of geek community is its institutional arrogance. Remember, just because geeks have under-developed social skills doesn't mean that they don't have egos. Tech people are uppity by definition: a lot of them would like to have been astronauts. They enjoy the limelight. In a power relationship with management, they have more in common with pro basketball players than they do with average workers. Think of your techies as free agents in a highly specialized sports draft. And the more specialized they are, the more you need to be concerned about what each of them needs as an individual."

"Today technology salaries are at least twice the national average. In fact, tech salaries are going through the roof, and non-tech salaries are not – which presents a serious problem for many companies. But, as important money is to tech people, it's not the most important thing. Fundamentally, geeks are interested in having an impact. They believe in their ideas, and they like to win. They care about getting credit for their accomplishments. In that sense, they're no different from a scientist who wants credit for work that leads to a Nobel Prize. They may not be operating at that exalted level, but the same principle applies."

"If you don't want to lose your geeks, you have to find a way to give them promotions without turning them into managers. Most of them are not going to make very good executives – and, in fact, most of them would probably turn out to be terrible managers. But you need to give them a forward career path, you need to them recognition, and you need to give them more money."

"Either geeks are part of the solution – or they're the problem. Here is another thing you need to know about the geek mind-set: because tech people are scientists or engineers by training, they love to solve really hard problems. They love to tackle a challenge. The more you can get them to feel that they're helping to come up with a solution to a tough problem, the more likely

they are to perform in a way that works for you. When you talk with them, your real goal should be to engage them in a dialogue about what you and they are trying to do. If you can get your engineering team to agree with what you're trying to accomplish, then you'll see them self-organize to achieve that outcome. You'll also need to figure out what they're trying to accomplish – because no matter what you want, that's probably what they're going to do. The next thing you need to remember is that you can tell them what to do, but you can't tell them how to do it."

"Make sure that there is always peer-group pressure within your project teams. For example, if you want to motivate your project leaders, just require them to make presentation to each other. They care a great deal about how they are perceived within their own web of friends and by the professional community that they belong to. [...] It sounds like I'm touting tech people as gods, but there are plenty of bad projects, and there is plenty of bad engineering and bad technology. You're always going to encounter techies who are arrogant and who aren't as good as they think they are. A team approach is the best way to deal with that problem. Tech people know how to deal with the wild ducks in their group – on their own and with the right king of peer pressure."

"You can listen to lots of exceptionally bright people talk about their brilliant vision. But what matters is, which ones deliver on their vision? When a project is on the line, who actually gets the job done?"

"In a high-tech company that is run by engineers, what matters most is being right. And what's 'right' is determined by outcome. You can listen to lots of exceptionally bright people talk about their brilliant vision. I've done it for the past 25 years. But what matters is, which ones deliver

on their vision? When a project is on the line, who actually gets the job done? Every team has a natural leader – and often that leader is not a team's official manager. Your job is to get the team motivated. Once you do that, the natural leaders will emerge very quickly. If you keep an eye on the team, you can figure out who those natural leaders are – and then make sure that they're happy and that they have everything they need to do their job. [...] There are easy ways that you can help them to bypass layers of middle management and to send you email directly. Sure, that will piss off the people in the middle management, but it's better to piss off those people than to piss off your key project leaders."

"You can divide project teams into two categories. First there is the preferred variety: you get an engineering team that's absolutely brilliant, that executes well, and that's exactly right in its assumptions. Second, there is the more usual variety: you get an engineering team that has very strong opinion about what it's trying to do – but that's on the wrong track, because some of its assumptions are wrong. That second kind of team is what you have to focus your attention on. But often you can't intervene from the top down. You have to find a way to come at the problem from the side. [...] In general, as long as you consider everyone's ideas, most teams react well to management decisions. If you have to make a business decision that conflicts with what your engineers want to do, they'll accept it – as long as it is truly a business decision. On the other hand, if the decision is based on a technology analysis by someone whom the engineers do not respect professionally, then they'll never agree to it. So, if you're facing a decision that you know will affect a team negatively, you must vet that decision through a technologist who has that team's respect."

"Too many geeks spoil the soup. If you want your geeks to be productive, keep your team small. The productivity of any project is inversely proportional to the size of the project team. In the software business, most problems

draw on the efforts of large number of people. Typically, companies deal with a problem by putting together a large team and then giving that team a mission. But in this industry, that approach almost never works. The results are almost invariably disappointing. Still, people keep doing it that way – presumably because that's the way they did it last year. The question is, how do you break out of that mode? It seems to be a cancer on the industry. On a large team, the contributions of the best people are always smaller, and overall productivity is always lower. As a general rule, you can count on each new software project doubling in team size and in the amount of code involved – and taking twice as long – as the preceding project. [...] Two or three people invent a brilliant piece of software, and then five years later, 1000 people do a bad job of following up on their idea. History is littered with projects that follow this pattern: Windows, Unix, Java, Netscape Navigator. The smaller the team, the faster the team members work. When you make the team smaller, you make the schedule shorter. That may sound counterintuitive, but it's been true for the past 20 years in this industry, and it will be true for another 20 years. The only method that I've found that works is to restrict the size of teams arbitrarily and painfully. Here's a simple rule of thumb for techie teams: no team should be larger than the largest conference room that's available for them to meet in."

Source: "How to Manage Geeks", Russ Mitchell, Fast Company, June 1999.

This interview of Novell's CEO Eric Schmidt provides a insider's vision of what our work's environment often looks like. Schmidt is an ex-Sun chief technology officer, so his point of view comes from practical experience on both sides of the management barrier. It corresponds with Methods & Tools philosophy to bring to you practical and open visions, just to make you think, even with a different point of view...

No Silver Bullet...

"InfoWorld: How will the adoption of Internet technologies such as HTTP, HTML, Java, and XML ease the maintenance of applications and the cost of maintaining systems?"

Phipps: Well, the most important thing to say there is there are no silver bullets. None of these technologies is a silver bullet. The observation is that the way that we've done software development over the past two decades is to assume the most important question is, "When can I have it?" And what we're discovering in the era of e-business [is that] probably the most important question will be, "What will it cost?" And if that starts to be the most important question, it seems you need ask yourself, "What's it going to cost me to use this proprietary technology in a month with all the other things I'm doing on my Web site?" What it's going to cost me is that proprietary technology is anchored to the operating system. And every time there's an upgrade to the operating system, every time I install other software that works with the operating system, I have to regression test my server to find out if anything is broken. But if I'm using Java or my data is expressed in XML, I can pretty much guarantee that any change in my server won't affect those programs, that there won't be any

codedependencies of those technologies and the platform. The result of that is that you'll see less frequently [that] a change [will] impact the rest of your systems, and that's how it reduces your cost. ... You can't say it's going to save you loads of money to use open technologies, but you can say it will cost you loads of money to use proprietary technologies."

Source: www.infoworld.com, InfoWorld Electric, November 22, 1999, "IBM's chief XML evangelist examines how the language will change e-commerce", interview with Simon Phipps.

... even on the Web! But independence is worthwhile in the software development area!



Coming next in Methods & Tools

- Using Use Cases
- Data Warehouse Design
- How to Sponsor a Successful Project
- GUI Testing Checklist
- 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 4300 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

METHODS & TOOLS is published by **Martinig & Associates**, Rue des Marronniers 25, CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch

Editor: Franco Martinig; e-mail franco@martinig.ch

Free subscription: www.martinig.ch

ISSN 1023-4918.

The content of this publication cannot be reproduced without prior written consent of the publisher

Copyright © 1999, Martinig & Associates