
METHODS & TOOLS

Global knowledge source for software development professionals

ISSN 1023-4918

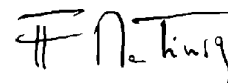
Winter 2001 (Volume 9 - number 4)

Choosing the Risk to Keep the Choice

Not so long ago, it was usual to say in the IT world that you could not be blamed to buy from IBM. Today the presence of Big Blue is perhaps less widespread, but the situation is not so different. Even if the time of the domination of a unique supplier has ended, there are many segments of the IT industry where the situation is quite similar, the only difference being that different suppliers are in the dominant position.

In the personal computer segment, Microsoft is the more flagrant case with dominant market shares in operating systems, development tools and Internet browsers. We can also name Oracle in the database market or Computer Associates in the mainframe system tools area. A large share of the software infrastructure of the IT world is composed of products from these dominant suppliers. In many cases also, the opinion is widely shared that dominant companies have achieved this position more because of their political and financial skills than for the quality of their products. Sadly, product quality is not always a key success factor to survive the regular financial crises that hurt the software industry and the price wars that explode in shrinking markets. It is perhaps not surprising that the most recent successful products' "launches" in Microsoft dominated markets have been with open sources software like Linux or Apache. One of their strength is that they are not produced by a company against which Microsoft can play its financial power, even if it also helps that Microsoft's enemies are contributing to their diffusion.

It is in the interest of the software consumers to keep many sources of supply and to let the competition open, not only as far as prices are concerned, but also in the product improvement and the customer service areas. Organisations that have exclusive agreements with dominant suppliers are acting against their long-term interests. They should also make "riskier" choices with smaller suppliers to help them demonstrate their knowledge. Therefore they will preserve their choices open.



Inside

UML: Understanding Structural Modeling	page 2
Process: Software Process Improvement.....	page 14
Testing: Test Tool Evaluations.....	page 19

Understanding Structural Modeling

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

Introduction

Following the “method wars” of the 1970s and 1980s, the Unified Modeling Language (UML) emerged from the unification that occurred in the 1990s within the information systems and technology industry. Unification was lead by Rational Software Corporation and the Three Amigos, Grady Booch, James Rumbaugh, and Ivar Jacobson. The UML gained significant industry support from various organizations via the UML Partners Consortium and was submitted to and adopted by the Object Management Group (OMG) as a standard (November 17, 1997).

The UML is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process. The language is broadly applicable to different types of systems (software and non-software), domains (business versus software), and methods and processes. The UML enables and promotes (but does not require nor mandate) a use-case-driven, architecture-centric, iterative, and incremental process that is object oriented and component based.

System development may be characterized as problem solving, including understanding or conceptualize and representing a problem, solving the problem by manipulating the representation of the problem to derive a representation of the desired solution, and implementing or realizing and constructing the solution. Within problem-solving, the UML enables capturing, communicating, and leveraging knowledge concerning systems: models capture knowledge (semantics), architectural views organize knowledge in accordance with guidelines expressing idioms of usage, and diagrams depict knowledge (syntax) for communication. This process is very natural and often occurs subtly and sometimes unconsciously in problem solving.

A system is a purposefully organized collection of elements or units. The architecture of a system entails two dimensions, the structural dimension and behavioral dimension. The structural dimension (also known as the static dimension) involves what elements constitute the system and their relationships. The behavioral dimension (also known as the dynamic dimension) involves how these elements interact to satisfy the purpose of the system and provide its functionality or behavior.

A model is a complete abstraction of a system that captures knowledge (semantics) about a problem and solution. An architectural view is an abstraction of a model that organizes knowledge in accordance with guidelines expressing idioms of usage. A diagram is a graphical projection of sets of model elements that depicts knowledge (syntax) about problems and solutions. Within the fundamental UML notation, concepts are depicted as symbols and relationships among concepts are depicted as paths (lines) connecting symbols.

Structural modeling from the structural model view (also known as the static view), which encompasses the structural dimension of a problem and solution, involves class diagrams to depict the static structure of a system in general and object diagrams to depict the static structure

of a system at a particular time. Class diagrams contain classes, which are general concept, and association, which are general relationships between classes. Object diagrams contain objects, which are specific concepts that are instances of classes, and links, which are specific relationships between objects that are instances of associations.

Figure 1 through 5 show various class diagrams related to books, catalogs, customers, authors, and publishers.

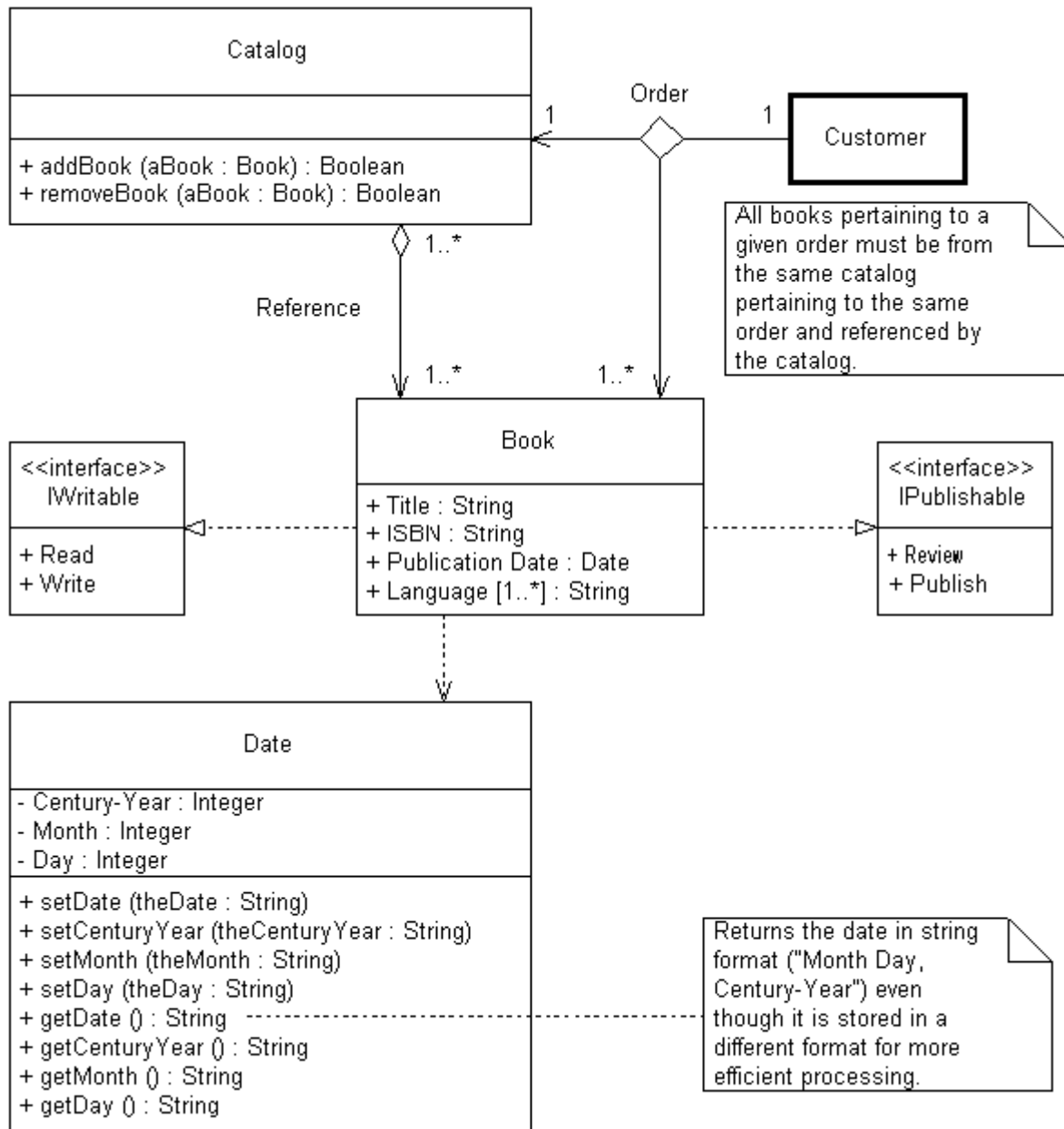


Figure 1: Books, catalogs, and customers.

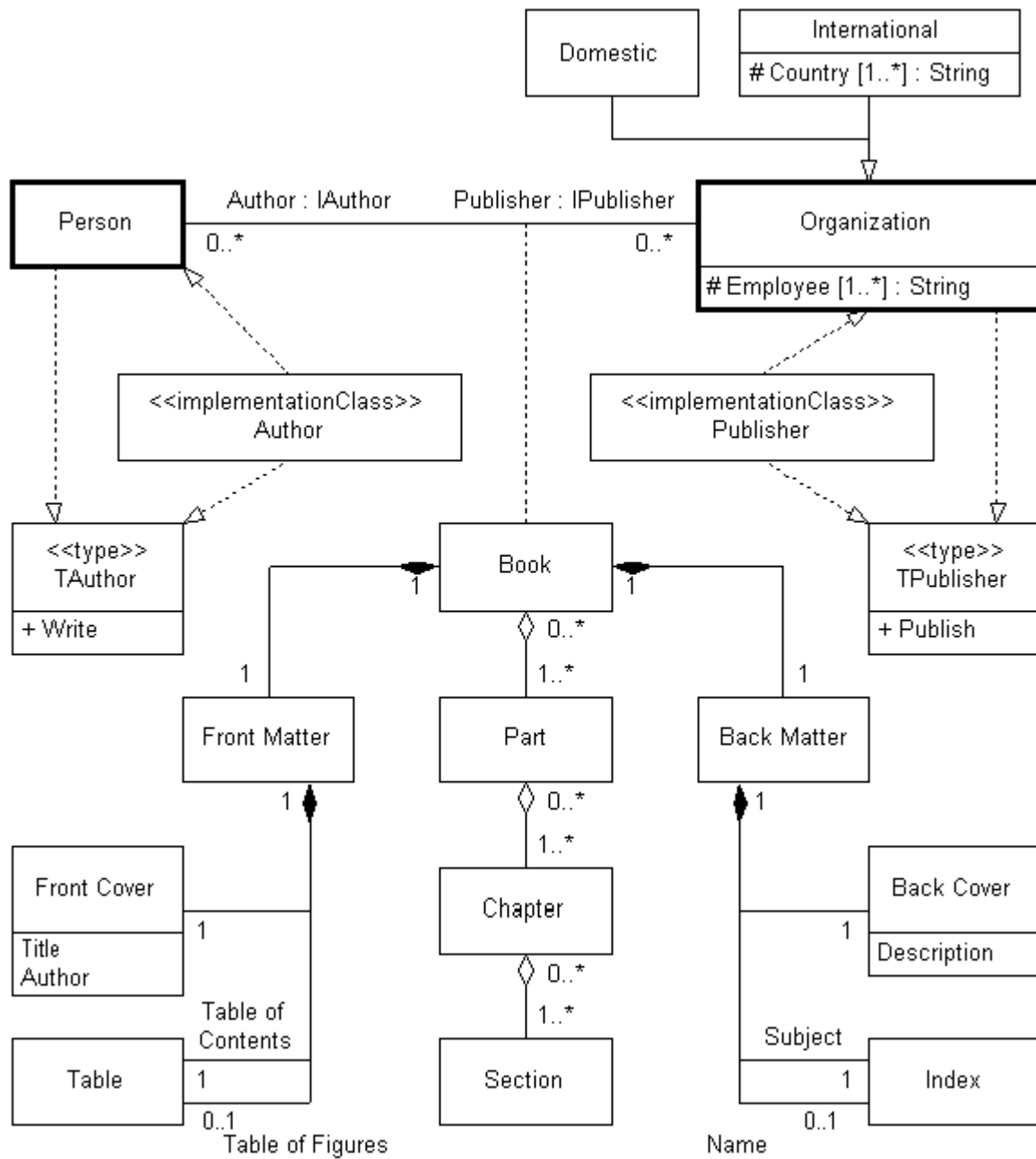


Figure 2: Books.

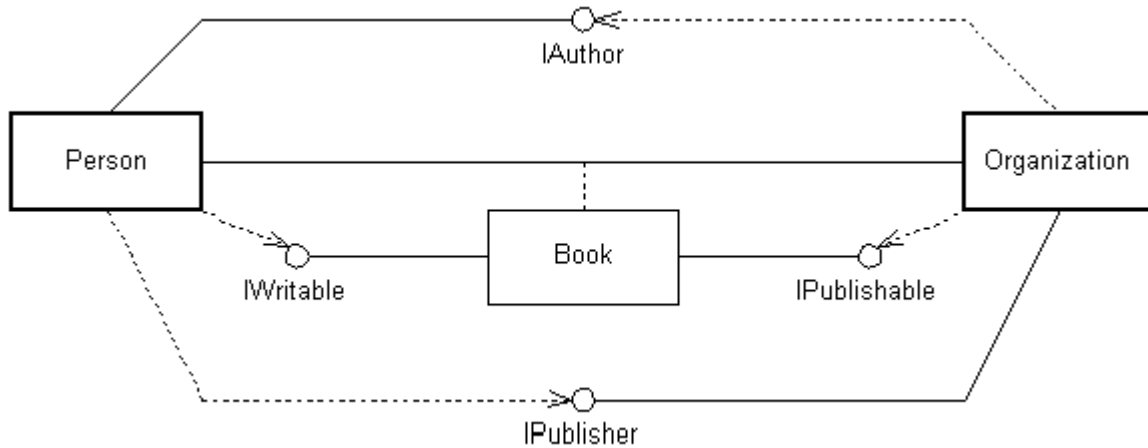


Figure 3: Books, authors, and publishers.

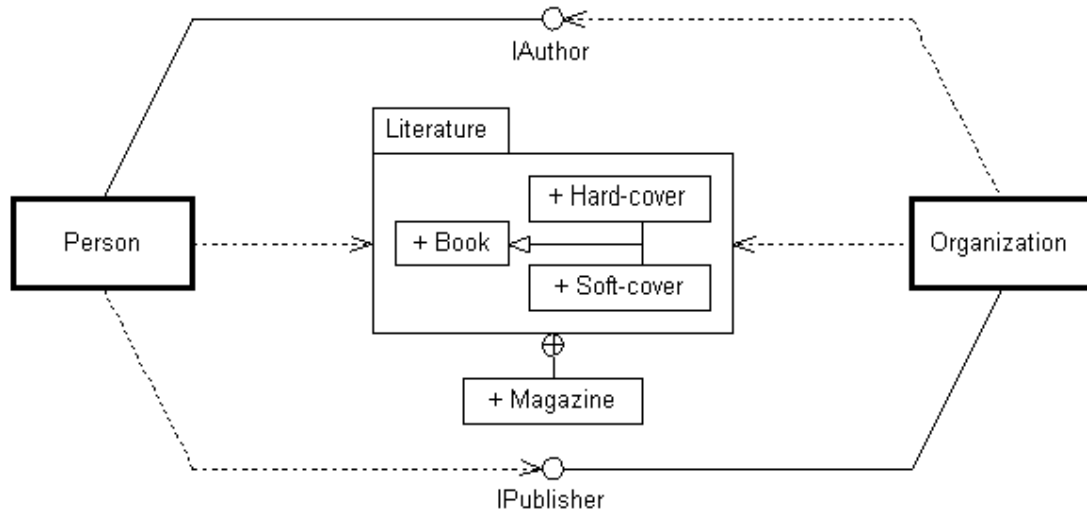


Figure 4: Literature, authors, and publishers.

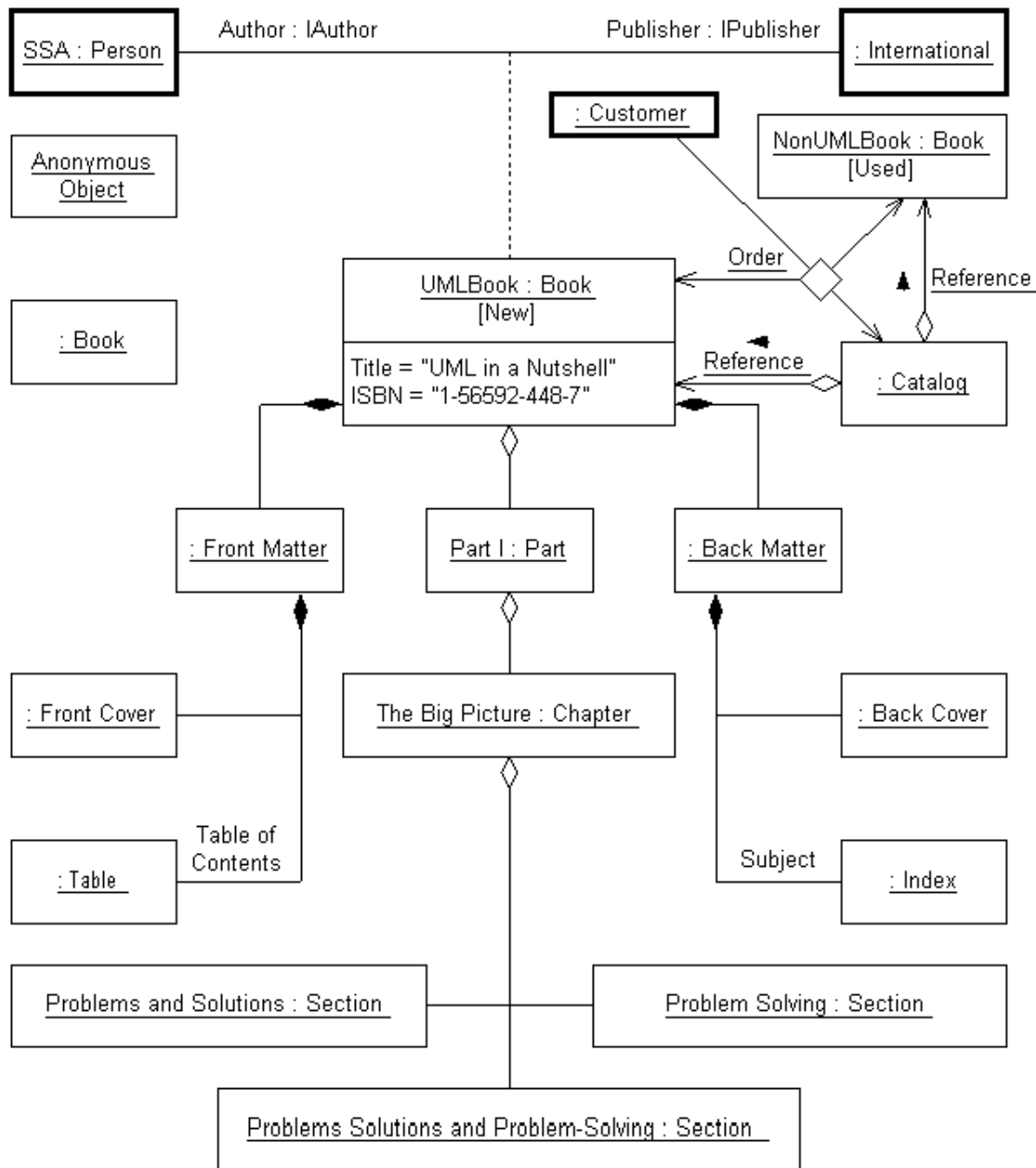


Figure 5: Specific books, a catalog, and a customer.

Class Diagrams

To effectively and successfully apply class diagrams, we must first understand the types of elements used in class diagrams.

Classes

A class is a general concept having structural features and behavioral features. Structural features, including attributes and relationships called associations, define what elements constitute the class. An attribute is an element of information or data. A relationship is a connection between classes. Behavioral features, including operations and methods, define how the constituents of the class interact to provide the behavior of the class. An operation is a specification of a service or processing. A method is an implementation of a service or processing. The most crucial aspect of a class is that it has semantics, some meaning.

A class is denoted as a solid-outline rectangle with three standard compartments separated by horizontal lines. The required top compartment contains the class name; the optional middle compartment contains a list of attributes, each denoted as a text string; and the bottom optional compartment contains a list of operations, each denoted as a text string. Each compartment needs only to show what is relevant to the diagram.

Figure 1 shows various classes, including Catalog, Book, Customer, and Date. Figure 2 shows various classes related to the Book class, including Person, Organization, Front Matter and its related classes, Back Matter and its related classes, Part, Chapter, and Section. Figure 3 and 4 focus on the Book, Person, and Organization classes. Figure 1 shows that a book may be added to or removed from a catalog, and each book has a title, International Standard Book Number (ISBN), publication date, and various languages in which it is available.

The visibility symbol “+” means an element is public and accessible from outside of its container, “-” means an element is private and inaccessible from outside of its container, and “#” means an element is protected and is inaccessible from outside of its container but is accessible by those containers that specialize or have a generalization to its container. There is no default visibility. Figure 1 shows that all of the attributes of the Book class and operations of the Catalog class are public while the attributes of the Date class are private and the operations of the Date class are public. Figure 2 shows that the Employee attribute of the Organization class and the Country attribute of the International class are protected.

A class or other model element may be marked using stereotypes to communicate some specific meaning concerning the model element. A stereotype is denoted as a text string keyword enclosed in guillemets («») or double-angle brackets preceding or above the model element name. Alternatively, a user-defined icon may also be used to signify a stereotype. When one or more stereotypes are applied to a model element, the model element is said to be stereotyped.

An active class is a class whose objects own a thread of control and may initiate control activity on their own behalf. A passive class is a class whose objects don't own a thread of control and may only initiate control activity in the course of processing a request wherein a thread of control is received. An active class or object is shown as a rectangle with a heavy border. Figure 1 shows that the Customer class is active while other classes are passive. Figures 2, 3, and 4 show that the Person and Organization classes are active while other classes are passive. Figure 5 shows active objects corresponding to these active classes.

A type is a stereotyped class having attributes, relationships, and operations but no methods. It is used to model a role or a part objects of a class may play in a particular situation without defining the physical implementation of the objects. A type is denoted as a class stereotyped with the “type” keyword.

Figure 2 shows that a person may play the role of an author via the TAuthor type and an organization may play the role of a publisher via the TPublisher type. An implementation class is a stereotyped class having attributes, relationships, operations, and methods. It is used to model a physical implementation of objects of a class. An implementation class is denoted as a class stereotyped with the “implementationClass” keyword. Figure 2 shows that the Author implementation class is related to the Person class and TAuthor type and that the Publisher implementation class is related to the Organization class and TPublisher type.

An interface is a stereotyped class having operations, but no attributes, relationships, nor methods. It is used to model a service or contract as a collection of externally-visible (public visibility) operations. An interface is denoted as a class stereotyped with the “interface” keyword. It may also be denoted as a small circle with the interface name placed near the symbol. The operations specified by the interface are not shown using the circle notation. The circle may be attached by a solid line to an element that supports the interface or provides an implementation for the operations specified by the interface. Figure 1 shows that a book is writable via the IWritable interface and publishable via the IPublishable interface. Figures 3 and 4 show two additional interfaces, IAuthor and IPublisher.

Associations

An association is a general relationship between classes. A binary association involves two classes and is denoted as a path connecting the two participating classes. An n-ary association involves three or more classes and is denoted as a large diamond with paths from the diamond to each participating class. An association may be labeled with a name. The name is read from left to right and top to bottom or may have a small black solid triangle next to it where the point of the triangle indicates the direction in which to read the name. An association end is an endpoint of an association, which connects the association to a class.

Figure 1 shows a Reference binary association between the Catalog and Book classes, and an Order n-ary association between the Customer, Catalog, and Book classes. This communicates that catalogs reference books and that orders involve customers, catalogs, and books. An association may have a related class called an association class defining structural features and behavioral features of the association itself. An association class is denoted as a class rectangle attached by a dashed line to its association path in a binary association, or attached by a dashed line to its association diamond in an n-ary association. The name in the class symbol and the name attached to the association should be the same. Figures 2 and 3 show that the Book association class is between the Person and Organization classes.

A rolename may be used to model how a class participates in an association, the “face” it projects in the relationship. A rolename is placed near the end of the association attached to the class. Figure 2 shows that the association between the Person and Organization classes involves the Person class having the role of an Author and the Organization class having the role of a Publisher.

An interface specifier may be used to model the behavior expected of a class by other classes that participate in an association; that is, a class's responsibilities. An interface specifier may be placed near the end of the association following a rolename (if one is used) and a colon.

Figure 2 shows that the association between the Person and Organization classes involves the Person class being responsible for offering the IAuthor interface and the Organization class being responsible for offering the IPublisher interface.

Navigation may be used to model that a class is identifiable and reference-able from associated classes. Navigation is denoted as an arrow attached to an association end to indicate that navigation is supported toward the class attached to the arrow, otherwise the association is assumed to be bi-directional or navigable in all directions.

Figure 1 shows that the Reference association between the Catalog and Book classes may be navigated from the Catalog class but not the Book class; that is, given a catalog, we can determine if a book is in the catalog, but given a book, we cannot determine if it is in any catalog. Figure 1 shows that the Order n-ary association between the Customer, Catalog, and Book classes may be navigated only from a customer to the catalog from which the customer ordered their books and to the books they ordered. Figures 2 and 3 show that all associations are navigable in all directions.

Multiplicity may be used to model how many instances of a class may participate in an association when the associated classes are fixed. Multiplicity is denoted as a comma-separated sequence of integer intervals, including literal integer values, closed ranges denoted as “lower-bound .. upper-bound,” and a single asterisk to denote an unlimited range. There is no default multiplicity for association ends, it is simply undefined.

Figure 1 shows that a catalog must have one or more books, a book must be in one or more catalogs, and an order involves one customer, one catalog, and one or more books. Figure 2 shows that a person who plays the role of an author may be associated with zero or more organizations playing the role of a publisher, an organization who plays the role of a publisher may be associated with zero or more people playing the role of authors. Figure 2 shows that a book has front matter, back matter, parts, chapters, and sections; front matter includes a front cover, table of contents, and optionally a table of figures all associated with a single book; back matter includes a back cover, a subject index, and optionally a name index all associated with a single book; and a book is associated with one or more parts, each part is associated with one or more chapters, and each chapter is associated with one or more sections where each section, chapter, and part may be associated with zero or more chapters, parts, and books respectively.

An aggregation association models a whole-part relationship between an aggregate, the whole, and its parts. An aggregation association is often called a “has-a” relationship because the whole has its parts. An aggregation association is denoted using a hollow diamond attached to the end of the path connected to the aggregate class. Aggregation is used with binary associations, but is not used with n-ary associations. Figure 1 shows an aggregation association between catalogs and the books they reference. Figure 2 shows various aggregation associations between books and their parts, chapters, and sections.

A composition association, also called composite aggregation, models a whole-part relationships between a composite, the whole, and its parts where the parts must belong only to one whole and the whole is responsible for destroying its parts when it is destroyed. A composition association is often called a “contains-a” relationship because the whole contains its parts. A composition association is denoted using a filled diamond attached to the end of the path connected to the class that is the composite. Composition is used with binary associations, but is not used with n-ary associations. Alternately, composition may be denoted by graphically nesting elements. A nested element's multiplicity may be denoted in its upper right corner and its rolename may be indicated in front of its class name followed by a colon. Attributes are, in

effect, composition associations between a class and the classes of its attributes.

Figure 2 shows various composition associations between books and their front matter, back matter, and their associated material.

Object Diagrams

To effectively and successfully apply object diagrams, we must first understand the types of elements used in object diagrams.

Objects

An object is a specific concept, an instance of a class. A class defines the structural and behavioral characteristics of its objects, and the stereotypes of a class apply to the objects of the class.

An object concretely defines values for its structural features, attribute values and specific relationships called links, and the behavioral features defined by the object's class apply to the object. The state of an object is the situation throughout the object's life during which it satisfies some condition. The state of an object is defined by the values of the object's structural features, what an object knows. The behavior of an object is defined by the behavior of its class, what an object can do. The most crucial aspect of an object is that it has its own identity. No two objects are the same, even if they have the same values for their structural features.

An object is denoted as a solid-outline rectangle with two standard compartments separated by horizontal lines. The required top compartment contains the object name followed by a colon followed by the object's class name (or a comma-separated list of class names) fully underlined, both names are optional and the colon is only present if the class name is specified; the optional second compartment contains a list of attributes, each followed by an equal symbol and its value. Each compartment need only show what is relevant to the diagram. To show the presence of an object in a particular state, follow the object name with the state name (or a comma-separated list of state names) enclosed within square brackets.

Figure 5 shows various objects, including a new UMLBook object, a used NonUMLBook object, an anonymous object named "Anonymous Object" without any identified class, and an anonymous book.

Links

A link is a specific relationship between objects, an instance of an association. If its association has a related association class, the link has a corresponding link object, which defines values for its structural features and the behavioral features defined by the link's association class apply to the link. A link end, similar to an association end, is an endpoint of a link, which connects the link to an object. Rolenames, interface specifiers, and navigation may be shown for links.

A binary link, a specific relationship between two objects, is denoted as a path connecting the two participating objects. A link may have its association name denoted near the path fully underlined, but links do not have instance names as they take their identity from the instances that they relate.

An n-ary link, a relationship between three or more objects, is denoted as a large diamond with paths from the diamond to each participating object. A link may have its association name

denoted near the path fully underlined, but links do not have instance names as they take their identity from the instances that they relate. Aggregation and composition may not be used with n-ary links. A link object, an object representing a link, is denoted as an object rectangle attached by a dashed line to its link path in a binary link, or attached to its link diamond in an n-ary link. Figure 5 shows various links.

Other Types of Elements

Other types of elements are commonly used in class and object diagrams, including notes, generalizations, dependencies, realizations, and packages.

Notes

A note is a notational item containing textual information, similar to a comment in a program. A note is denoted as a rectangle with a bent upper right corner. A note may be attached to zero or more elements by dashed lines. Figure 1 shows a note attached to the getDate operation and a note attached to the Reference and Order associations.

Generalizations

A generalization relationship between elements indicates that one element, the target, is a more general element than another more specific element, the source, which receives the attributes, relationships, operations, and methods from the more general element and whose instances may be substituted in the place of instances of the more general element. The more specific element may add its own more specialized attributes, relationships, operations, and methods, and may override the methods it inherits or receives. Both the more general element and the more specific element must be of the same type of modeling element (types, interfaces, implementation classes). The more general element is called a supertype, superclass, or generalization and the more specific element is called a subtype, subclass, or specialization. Inheritance is the sharing mechanism through which more specific elements receive the characteristics of more general elements. Single inheritance involves only a single more general element, and multiple inheritance involves at least two or more general elements. A generalization relationship is often called an “is-a-kind-of” relationship because the more specific element is a specialized type of the more general element. A generalization relationship on a class diagram is denoted as a solid-line path from the more specific element to the more general element, with a large hollow triangle at the end of the path connected to the more general element.

Figure 2 shows that there may be domestic as well as international organizations that play the role of publishers. Both domestic and international organizations have an Employee attribute identifying the organization’s employees which is inherited from the Organization class, but only international organizations have a Country attribute identifying the countries in which the organization operates.

Dependencies

A dependency relationship between elements indicates that one element, the source or client, uses or depends on another element, the target or supplier, where a change to the target element may require a change to the source element. A dependency on a class or object diagram is denoted as a dashed arrow from the dependent client element to the independent supplier element, and it may also be named.

Figure 1 shows that the Book class depends on or uses the Date class to maintain its Publication Date attribute. Figures 3 and 4 show how the various elements depend on interfaces.

Realizations

A realization dependency between elements indicates that one element, the target, provides a specification and another element, the source, provides an implementation for the specification. The implementing source element is said to realize or implement the specifying target element. Any element may be used as a specification or realization element because a realization relationship indicates conformance of behavior, where the source element supports at least all of the operations defined in the target element without necessarily having to support any structural features of the target element. A realization relationship is denoted on a class or object diagram using a dependency stereotyped with the “realize” keyword drawn from the source element to the target element or a dashed line with a solid triangle arrowhead attached to the target element. When the specifying element is an interface that is denoted as a small circle with the interface name placed near the symbol, the realization relationship is denoted as a solid line from the implementing element to the interface.

Figure 1 shows that the Book class realizes or implements the IWritable and IPublishable interfaces. Figure 2 shows that the Person class realizes or implements the TAuthor type, the Author implementation class realizes or implements the TAuthor type and Person class, the Organization class realizes or implements the TPublisher type, and the Publisher implementation class realizes or implements the TPublisher type and Organization class. Figures 3 and 4 show how the various elements realize interfaces.

Packages

A package is a general means for grouping and organizing elements. A package owns its contents wherein each element must be uniquely named. A package is denoted as a large rectangle with a small rectangle or “tab” attached to the left side of the top of the large rectangle. Packages may be nested and elements within a package see or have visibility all the way out through nested packages to the outermost package. A package may have a dependency on other packages to indicate that its contents use the contents of those packages. A package may have branching lines to contained elements that are denoted outside of the package with a plus sign (+) within a circle drawn at the end attached to the container, called tree notation.

Figure 4 shows how the Person and Organization classes depend on the Literature package, which houses the Magazine, Book, and other classes.

Structural Modeling

To effectively and successfully apply class and object diagrams in structural modeling, we ought to be aware of various guidelines (lessons learned) for applying these techniques.

Classes

When modeling classes, we ought to be aware of the following guidelines:

- A class should be named using a noun phrase.
- A class should be sufficiently described in terms of its attributes, operations, and associations.

- A class should be a well-defined abstraction. A class's attributes, operations, and associations do not sufficiently define the class, but only describe it. To sufficiently define a class, there ought to be sufficiently understandable semantics -- that is, the meaning of the class and its contextual business semantics, rules, and so forth. For example, what does it mean to be a book or catalog? Such semantics are very context dependent, which makes the semantics even more important in defining and effectively using a class.
- An attribute should be named using a noun phrase that is descriptive, understandable, and represent a single well-defined data element maintained by an object of a class.
- An operation should be named using a verb or verb-noun phrase that is descriptive, understandable, and represent a single well-defined service provided by an object of a class.
- An interface should be named using a noun phrase that is descriptive of the service or contract that the interface represents. It is common to name interfaces as “IX” where X is the service name for elements that are active or may initiate control on their own behalf (they have a thread of control); in which case, the interface is named from the perspective of those elements that realize the interface. For example, the IAuthor and IPublisher interfaces are realized by the Person and Organization classes, whose objects are active. It is also common to name interfaces as “Xable” where X is the service name for elements that are passive or may not initiate control on their own behalf (they don't have a thread of control); in which case, the interface is named from the perspective of those elements that use the interface. For example, the IWritable and IPublishable interfaces are realized by the Book class, whose objects are passive.
- Types, interfaces, non-implementation classes, and implementation classes should be used to better communicate about classes in a specific context. Use types to model a role or a part objects of a class may play in a particular situation without defining the physical implementation of objects, implementation classes to model a physical implementation of objects of a class, and interfaces to model a service or contract as a collection of externally-visible (public visibility) operations.
- Use generalization relationships to model the relationship between more general elements and more specific elements. Consider if the phrase “is-a-kind-of” describes the relationship.
- Use dependencies to model the relationship between one element that uses or depends on another element. Consider if the phrase “uses” or “depends on” describes the relationship.
- Use realization dependencies to model the relationship between specification and implementation elements. Consider if the phrase “implements” describes the relationship.
- Use packages to organize elements. However, always consider the criteria for grouping elements into packages.

Furthermore, other guidelines may be applied in addition to those above.

Associations

When modeling associations, we ought to be aware of the following guidelines:

- An association class should be named using a verb phrase.
- An association should be sufficiently described in terms of the classes it relates. An association class should be sufficiently described in terms of its attributes, operations, and associations.
- An association should be a well-defined abstraction. The classes an association relates and an association class's attributes, operations, and associations do not sufficiently define the

association, but only describe it. To sufficiently define an association, there ought to be sufficiently understandable semantics -- that is, the meaning of the association and its contextual business semantics, rules, and so forth. For example, what does it mean to have a relationship between a person and an organization involving a book? Such semantics are very context dependent, which makes the semantics even more important in defining and effectively using an association.

- Rolenames, interface specifiers, navigation, and multiplicity should be used to better communicate about associations in a specific context. Use rolenames to model how a class participates in an association, interface specifiers to model the behavior expected of a class by other classes that participate in an association, navigation to model that a class is identifiable and reference-able from associated classes, and multiplicity to model how many instances of a class may participate in a relationship when the associated classes are fixed.
- Associations, aggregation associations, and composition associations should be used to better communicate about associations in a specific context. Use associations to model a general relationship between classes, aggregation associations to model whole-part relationships, and composition associations or composite aggregation to model whole-part relationships where the parts must belong only to one whole and the whole is responsible for destroying its parts when it is destroyed. Consider if the phrase “has-a” describes aggregation associations and if the phrase “contains-a” describes composition associations.

Furthermore, other guidelines may be applied in addition to those above.

Objects and Links

Use object diagrams to explore and refine class diagrams in a specific context. For example, figure 1 through 4 are used to refine the relationship between books, catalogs, customers, authors, and publishers based upon various object diagrams similar to figure 5.

Conclusion

As the Unified Modeling Language (UML) is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process, by understanding the types of elements used in class and object diagrams and being aware of various guidelines (lessons learned) for applying these techniques in structural modeling, we have a sound foundation for effectively and successfully applying the techniques. Furthermore, it is experience, experimentation, and application of the standard and its various techniques that will enable us to realize its benefits.

Software Process Improvement: the New 'Silver Bullet'?

Hans Sassenburg , hsassenburg@se-cure.ch,
SE-CURE AG, www.se-cure.ch.

Introduction

From the end of the eighties onwards it became clear that the use of information technology tools in our society would expand enormously. This trend is characterised, among other things, by revolutionary developments in the software industry. But the problems experienced in the management of projects are becoming progressively greater. Budgets are substantially exceeded, delivery times are not met, the ultimate product does not satisfy expectations and, in addition, it contains many defects. In the past, numerous 'silver bullet' solutions have been put forward to cope with the problems outlined. The use of advanced methods/techniques and the acquisition of certificates have often cost a great deal of time and money, but has seldom led to the desired result. Managers are often completely at a loss and ask themselves how these problems can be solved. In recent years, an increasing interest in improving the software development process has been observable: Software Process Improvement. This article deals with the question of the extent to which this attention to the process alone is sufficient, together with an explanation based on a practical example.

The software crisis

Tumultuous developments are taking place in the software industry. Turnover in 'embedded software' - software incorporated into products such as television, audio, telephone, medical and telecommunications equipment - will increase sharply in the coming years. A growth from 5 to 20 billion dollars in Europe, say the forecasts. Observable trends:

- A substantial growth in the use and complexity of software in products and as stand-alone products.
- More demand for open systems with standardised interfaces, so that link-ups can be made to other (standard) products.
- Greater importance of hierarchical and robust architectures, aimed at future adaptations and expansions.
- Ever-larger and more highly qualified development teams, working on a geographically scattered basis.
- The availability of progressively newer techniques.
- An increasingly higher investment level for the development of new products.
- A strong growth in subcontracting projects to specialised companies (both within the Western Europe and also to low wage countries).

Running counter to these trends, within both companies and government organisations there is unrelenting pressure aimed at:

- Shortening project lead times, so that products can be introduced to the market faster.
- Increasing productivity by operating more efficiently.
- Improving customer-satisfaction by fulfilling prior agreements in which functionality, quality,

costs and delivery times are laid down as precisely as possible.

- Anchoring activities in the organisation in order to search continually for and implement improvements: the self-teaching organisation (TQM).

The areas of conflicting interests, which have thus arisen, are not new. Numerous attempts have already been made in the past to find an appropriate answer. Among other things, this was sought in the use of advanced methods and techniques. The results were disappointing, however. At the end of the eighties excessive attention suddenly emerged for the ISO-9001 standard, together with the appertaining ISO/9000-3 directive. Many organisations have attempted to obtain certification as quickly as possible. This has often become an end in itself and that certificate is mainly used as a commercial visiting card. As such, these certificates have made scarcely any contribution to achieving a structural improvement in the management of projects. So what is the answer?

Process approach

Even in undisciplined organisations, one occasionally comes across projects which have worked extremely successfully. The success of such projects, however, is generally attributable to the heroic efforts made by a project team instead of following a disciplined procedure laid down for the organisation, and therefore applying to every project. Because of the absence of a clearly established process, future results depend entirely on the availability of these same people for a subsequent project. If success depends on the availability of people, this cannot constitute the basis for long-term continuity and improvement. That can only be achieved by establishing and continually improving those processes, which play a part in software development for the whole organisation. This idea is beginning to be more and more widely accepted and is known as: Software Process Improvement.

In 1986 the Software Engineering Institute (Pittsburgh/USA) - at that time headed by Watts S. Humphrey - started developing a 'process maturity framework' to help organisations improve their software development processes. This was prompted by a request from the American government (Ministry of Defense) to supply a method for assessing suppliers with regard to their ability to implement software projects effectively. After years of experience with this framework, based on investigations carried out in many companies, this led in 1991 to the release of the Capability Maturity Model. The CMM distinguishes between five levels of maturity which an organisation may have reached. This stratified structure is based on ideas of such quality gurus as Deming, Juran and, particularly, Crosby ('Maturity Grid').

SPI in Europe

In our consulting work, we increasingly come across companies which embrace the SPI concept and use the CMM as a reference framework for improving their software capability. At the start of the nineties, this was still confined to larger, internationally operating companies which were active in extensive embedded software projects. Many millions of guilders were invested in extensive improvement programmes. Slowly but surely attractive results are now beginning to become visible. Do not forget: here it is a question of invest first, earn later. Following the example of these organisations, the SPI concept gradually also found application in smaller organisations, but still in the area of 'embedded software'. Since 1995, however, we have seen a change taking place. The entire banking and insurance business is paying great attention to SPI and major improvement programmes will start up have started up since. It is, of course, interesting to see that the problems in this business are identical to those in the technical field. Our experience in numerous organisations shows that the software industry in Europe is in no way inferior to that in other parts of the world. It turns out that over 90% of the organisations known to us are in the

bottom regions of the CMM. We only come across more mature organisations occasionally, and these are generally small development departments in small organisations.

Other possible solutions

The Software Process Improvement concept and the Capability Maturity Model as a reference framework are rapidly gaining in popularity. Numerous organisations are taking this process approach on board and regard it as the latest 'silver bullet' solution. Is this euphoria justified? A comparison with a completely different discipline may possibly prove instructive: the construction world.

Control parameters

A number of control parameters can be distinguished in the construction world which may be regarded as important for the successful completion of a building project: the quality of the construction plan, including working agreements, the availability of the necessary tradesmen and the availability of the right tools. A comparison with the software industry is given below.

Construction world	Software industry
Construction plan, incl. Working agreements	Development processes
Tradesmen	Training, experience, skills
Tools	Technology

All the control parameters are important, both in the construction world and in the software industry. Placing too much emphasis on only one of the parameters is too one-sided and will result in sub-optimisation. After all, what good are excellent tradesmen if the construction plan displays too many defects? What is the point of the most modern tools if there are no well-trained specialists available to use them?

Proposition 1: *“All the control parameters must be in balance with each other.”*

Requirements

Secondly, one may ask oneself which requirements must be met by the various control parameters. Does it make sense to impose the same requirements on the construction plan, the tradesmen and the tools for building a garage as for building a new tunnel? Of course not.

Proposition 2: *“The ultimate requirements at the level of the control parameters are determined by the characteristics of the product to be realised.”*

By analogy, in the software industry it may be said that aiming uncritically to achieve the highest CMM level is not useful for every organisation.

Priorities

Now let us assume that the level of all the various control parameters is not in accordance with the characteristics of the product to be realised. Which control parameter should then take priority? In the software industry at present it is said that attention to the process should take priority. But is this really true? Would you give preference to a good carpenter with a bad hammer or a poor carpenter with a good hammer?

Proposition 3: “*People constitute the most important control parameter in every discipline*”.

Opting for good tradesmen will prove to be the most justified choice in every case. When it comes to making a choice in the software industry, it is therefore probably advisable to invest in recruiting and maintaining qualified staff in addition to improving the development process or buying the latest technology.

Assessments

Over the last ten years, we have increasingly come into contact with organisations which want to use SPI and CMM. The questions most frequently put to our consultants are:

- How much does a CMM assessment cost?
- How do we reach the next CMM level as quickly as possible?

Based on the propositions above, caution is advisable. In every organisation that asks us these questions we try to show that:

- In addition to attention for the dimension 'Process', attention to other control parameters is desirable, namely 'People' and 'Technology'
- It will be necessary to identify the characteristics of the present and future products ('Product') in order to be able to decide what requirements must be met by the various control parameters.

Only then can a sensible strength/weakness analysis of an organisation be made. In every case, we have found that organisations appreciate this expansion of scope and that, in particular, the management is delighted with this. That is particularly because an attempt is also made to find out what over-arching business objectives are eventually aimed at. Recommendations resulting from an assessment should be brought into line with each other as far as possible here.

An assessment generally proceeds as follows:

- Preparation (lead time 2-4 weeks)

In this phase, the organisation to be investigated is studied on the basis of an organisation description, the available manuals and some intake interviews. Next, the scope of the research is determined in consultation with the client by selecting a number of representative projects. The persons to be interviewed during the implementation phase are then selected.

- Implementation (lead time 2-4 days)

The interviews are held during this phase. Before they start, a kick-off presentation is given to all those directly involved. A crucial role is reserved for the client in convincing all those concerned of the need for the investigation by pointing out the main business objectives. Next, the interviews are held. These are strictly confidential and open in nature. For this reason, it is preferable not to allow the client to be present at the interviews unless otherwise explicitly agreed. On average an interview lasts for 1.5 hours.

- Analysis (lead time 2-4 days)

The information collected during the implementation phase is analysed in this phase. Firstly, based on the characteristics of the product it is decided what requirements would have to be satisfied by the control parameters 'People', 'Technology' and 'Process' both in the present situation and within three to five years. After that, the maturity level of each control parameter is determined on a scale of 1 to 5. The CMM is used as a reference here for 'Process'. The risks, conclusions and recommendations are then drawn up. All this information is recorded in a report. The recommendations are generally long-term in nature, varying from one to two

years.

- Reporting (lead time 1 day)

In this phase, the report is discussed with the client. First, there is a discussion with the client himself and later a presentation is given to all the persons interviewed, as well as others concerned and interested parties. Once again, a crucial role is reserved here for the client in promising everyone involved that the recommendations made will actually be followed up in practice.

- Follow-up (lead time 1 year)

The last phase of an assessment is also the most important phase. On the basis of the recommendations made, the client will have to decide how an improvement process can be initiated and what resources will have to be made available for this purpose.

Practical example

In 1999 we carried out an assessment in a company which specialises in automation applications in the petrochemical processing industry: depot monitoring, data acquisition, process control and trend analyses. First, we were asked to analyse a project which had gone off the rails and to help it to get back on course as quickly as possible. By mutual agreement, however, it was decided to make an assessment covering several projects in order to try to analyse the 'capability' of the entire software development department. The characteristics of the type of product mean that the control parameters must at least be at level 3. It was found, however, that the 'People' dimension and the 'Process' dimension (CMM level 1) fell short in this respect. It proved that the so-called software developers had little or no training in information technology. Their backgrounds varied from biology to electrical engineering. The development process we encountered had not been laid down, which resulted in unrealistic planning, unclear working agreements and an ad-hoc approach to work. On the basis of the recommendations made it was then decided to start three improvement processes:

- Recruiting some experienced information technologists ;
- Setting up a training program for the present development engineers;
- Improving the software development process.

Despite the high investment level required for this the company is now in a much better position to plan new projects in terms of budgeting and lead times and the quality of the end products is increasing. The number of complaints reported by customers has decreased by a factor of 4, despite a doubling of the installed base.

Conclusion

This article has dealt with the increasing attention being paid to improving the software development process. In Europe, we see an explosive growth in this interest, particularly in the banking and insurance business. By making an extrapolation to the construction world, however, an attempt has been made to show that attention to other control parameters is also important. Recruiting qualified and experienced staff and keeping them motivated probably deserves even higher priority and is at present an extremely difficult issue, given the tightness of the present labour market. In addition, intelligent use should be made of the available technology as a support in the development process. The challenge will lie in managing to find the correct balance between these different control parameters in relation to the type of product being developed. In that sense, SPI therefore cannot and must not be described as the 'silver bullet' solution. That does not exist.

Test Tool Evaluations

Elfriede Dustin, Jeff Rashka and Douglas McDiarmid

This article is an extract from the book, *QUALITY WEB SYSTEMS*, (appendix B.5 and B.6). © 2002 Pearson Education, Inc, written by E. Dustin, D. McDiarmid and J. Rashka. Reproduced by permission of Pearson Education, Inc. More information on this book can be found at the following website <http://cseng.aw.com/book/0,3828,0201719363,00.html>

Anyone who has contemplated the implementation of an automated test tool has quickly realized the wide variety of options on the market in terms of both the kinds of test tools being offered and the number of vendors. The best tool for any particular situation depends on the system engineering environment that applies and the testing methodology that will be used, which in turn will dictate how automation will be invoked to support the process.

This partial appendix (B.5) evaluates performance testing and analysis, and vendor qualification, while (B.1 – B.4) evaluates major tool vendors on their test tool characteristics, test execution capability, tool integration capability, test reporting capability. The tool vendors evaluated are Compuware, Empirix/RSW, Mercury, Rational, and Segue.

Load and Stress Test Features

Criterion/Feature	Compuware	Empirix/SW	Mercury	Rational	Segue
All users can be queued to execute a specified action at the same time	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes Via sync points	SilkPerformer—Yes
Automatic generation of summary load testing analysis reports	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Ability to change recording of different protocols in the middle of load-recording session	Performance Edition—Yes Can record multiple middleware and protocols during same recording session.	eTest Suite—Yes Can change protocols as long as within the web transaction and can be recorded.	LoadRunner—Yes for some protocols.	Rational Suite TestStudio, Rational Robot—Yes All protocols can be captured during a single recording session, so “changing” protocols is not necessary. After recording, the multiple protocols can be put into a single script, or, filtering could be used to put a single, or a combination of multiple protocols into a single script.	SilkPerformer—Yes

Testing

Criterion/Feature	Compuware	Empirix/SW	Mercury	Rational	Segue
Actions in a script can be iterated any specified number of times without programming or rerecording of the script	Performance Edition—Yes	eTest Suite—Yes With or without variable data.	LoadRunner—Yes Astra LoadTest—Yes A simple runtime setting.	Rational Suite TestStudio, Rational Robot, Rational TestManager—Yes Most other tools require that all user actions be put in a single script Rational Robot allows a user session to be split into multiple scripts, each of which can be iterated any number of times via the graphical scheduling mechanism in TestManager.	SilkPerformer—Yes
Different modem connection speeds and browser types can be applied to a script without any rerecording	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes A simple runtime setting.	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Load runs and groups of users within load runs can be scheduled to execute at different times	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Automatic load scenario generation based on load testing goals: hits/second, number of concurrent users before specified performance degradation, and so on	Performance Edition—Yes	eTest Suite—No	LoadRunner—Yes	SiteLoad, Rational Suite TestStudio—Yes (in TestManager via Transactors)	SilkPerformer—Yes
Cookies and sessions IDs automatically correlated during recording and playback for dynamically changing Web environments	Performance Edition—Yes	eTest Suite—Yes Never requires programming, entirely automatic	LoadRunner—Yes Astra LoadTest—Yes Does not require programming, entirely automatic.	Rational Suite TestStudio, Rational TestManager, Rational Robot—Yes	SilkPerformer—Yes
Allows for variable access methods and ability to mix access methods in a single scenario: modem simulation or various line speed simulation	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational TestManager Suite—Yes Any combination of tests is possible.	SilkPerformer—Yes
Ability to have data-driven scripts that can use a stored pool of data	Performance Edition—Yes	eTest Suite—Yes with DataBank Wizard	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager, Rational Robot—Yes Via Datapools.	SilkPerformer—Yes
Allows for throttle control for dynamic load generation	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes	Rational Suite TestStudio, Rational TestManager—Yes With Transactors (graphical point and click creation) or Shared Variables (hand coded)	SilkPerformer—Yes

E Dustin/D McDiarmid/J Rashka, QUALITY WEB SYSTEMS, (appendix B.5 and B.6).
© 2002 Pearson Education, Inc. Reproduced by permission of Pearson Education, Inc.

Testing

Criterion/Feature	Compuware	Empirix/SW	Mercury	Rational	Segue
Allows for automatic service-level violation (boundary value) checks	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Scripts can be deployed to support these checks.
Allows for variable recording levels(network, Web, API, and so on)	Performance Edition—Yes	eTest Suite—No	LoadRunner—Yes	Rational Suite TestStudio, Rational Robot—Yes	SilkPerformer—Yes Allows variable levels and mixing of protocols for a script, user or transaction.
Allows for transaction breakdown/drill-down capabilities for integrity verification at the per client, per session, and per instance level for virtual users	Performance Edition—Yes Offers integrated load testing and server monitoring.	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Allows for Web application server integration: ColdFusion, NetDynamics, Dynamo, and so on	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes Including ability to record and replay against the applications, and monitor the app server performance during load test.	Rational Suite TestStudio, Rational Robot, Rational TestManager—Yes	SilkPerformer—Yes
Supports workload, resource, and/or performance modeling	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Can run tests on various hardware and software configurations	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Support headless virtual user testing feature	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Requires low overhead for virtual user feature (Web, database, other?)	Performance Edition—Yes Exceptionally low overhead	eTest Suite—Yes Small footprint per Virtual User	LoadRunner—Yes Very high scalability ratings using TurboLoad technology.	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Scales to 500–1,000 virtual users	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Largest load recorded to date generated more than 1.35M concurrent connections. Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Simulated IP addresses for virtual users	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes Both use IPSpoof feature.	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes

E Dustin/D McDiarmid/J Rashka, QUALITY WEB SYSTEMS, (appendix B.5 and B.6).
© 2002 Pearson Education, Inc. Reproduced by permission of Pearson Education, Inc.

Testing

Criterion/Feature	Compuware	Empirix/SW	Mercury	Rational	Segue
Thread-based virtual user simulation	Performance Edition—Yes	eTest Suite—Yes Default and most efficient mode of testing	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Process-based virtual user simulation	Performance Edition—Yes	eTest Suite—Yes Optional	LoadRunner—Yes Included since some apps require it for accuracy.	Rational Suite TestStudio, Rational TestManager—Yes Each virtual user has its own process space and unimpeded access to all resources within a process. Multi-threaded model supported when needed.	SilkPerformer—Yes Can set any number of virtual users as a universal default parameter or set number within a script.
Centralized load test controller	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra LoadTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Allows for reusing scripts from functional test suite	Performance Edition—Yes Functional/load test scripts can be created at same time. Two use different languages.	eTest Suite—Yes Visual Scripts created during functional testing reused for load testing and monitoring	Astra QuickTest—Yes WinRunner scripts used as GUI Vusers in LoadRunner.	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Support for WAP protocol testing against WAP Gateway or Web server	Performance Edition—Yes	eTest Suite Yes for the web server	LoadRunner—Yes	Rational Suite TestStudio, Rational Robot—Yes	SilkPerformer—Yes
Compatible with SSL recording	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational Robot—Yes	SilkPerformer—Yes
Compatible with one or more of the relevant technologies: streaming media, COM, EJB, RMI, CORBA, Siebel, Oracle, SAP	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes	Rational Suite TestStudio, Rational Robot—Yes	SilkPerformer—Yes
Compatible with one or more of the relevant technologies: Linux, UNIX, NT, XWindows, Windows CE, Win3.1, Win95, Win98, Win2000, WinME	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational Robot, Rational TestManager—Yes	SilkPerformer—Yes

E Dustin/D McDiarmid/J Rashka, QUALITY WEB SYSTEMS, (appendix B.5 and B.6).
© 2002 Pearson Education, Inc. Reproduced by permission of Pearson Education, Inc.

Performance Monitor Test Features

Criterion/Feature	Compuware	Empirix/SW	Mercury	Rational	Segue
Monitors various tiers: Web server, database server, and app server separately	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational Robot, Rational TestManager—Yes	SilkPerformer—Yes
Supports monitoring for one or more of ColdFusion, Broadvision, BEA WebLogic, Silverstream, ATG Dynamo, Apache, IBM Websphere, Oracle RDBMS, MS SQL Server, Real Media Server, IIS, Netscape Web Server	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkVision—Yes SilkVision component provides monitoring capability.
Supports monitoring for one or more of the relevant technologies: Linux, NT, UNIX, XWindows, Windows CE, Win3.1, Win95/98, Win2000	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkVision—Yes SilkVision component provides monitoring capability.
Monitors network segments	Performance Edition—Yes With EcoSystems and Application Expert	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkVision—Yes SilkVision component provides monitoring capability.
Supports resource monitoring	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Synchronization ability in order to determine locking, deadlock conditions, and concurrency control problems	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Ability to correlate any metrics from all monitors to identify performance bottlenecks	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Ability to detect when events have completed in a reliable fashion	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Ability to provide client-to-server response times	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes
Ability to provide graphical results and export them to common formats	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes Astra QuickTest—Yes	Rational Suite TestStudio, Rational TestManager—Yes Results can be exported to .csv.	SilkPerformer—Yes
Ability to provide performance measurements of data loading	Performance Edition—Yes	eTest Suite—Yes	LoadRunner—Yes	Rational Suite TestStudio, Rational TestManager—Yes	SilkPerformer—Yes

Classified Advertisement

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organising software development related training? This space is waiting for you at the price of US \$ 20 each line. Reach more than 19'000 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 this section or to place a page ad simply send an e-mail to 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 ISSN 1023-4918
Free subscription: www.martinig.ch/mt/index.html
The content of this publication cannot be reproduced without prior written consent of the publisher
Copyright © 2001, Martinig & Associates