

## **Adaptable Design Helps Business Practices Become Automated Practices**

by Wayne Cunneyworth  
SUMO Software Services Inc.

### **Doing Business in a Dynamic World**

In today's competitive business environment, adaptability is a high priority requirement for end-to-end business processes and their automated counterparts. Recent surveys demonstrate that the best insurance carriers have come to realize the need for both corporate efficiency and customer centricity. Business practice improvements are being implemented, and the improvements themselves are being refined over time. Technology promises to help inefficient business practices adapt to recognized best practices and other improvements. In fact, the successful evolution of today's business often depends upon the timely integration of these new and revised practices into existing software infrastructures. The challenge is compounded when each new generation of software products is followed closely by intense pressure for more functional enhancements, greater customer focus, performance improvements and defect corrections.

Today's software users have increasingly high expectations for automated systems. They have become more knowledgeable in their use of technology; they are more critical and less tolerant of defective products delivered by IT professionals. Insurance systems that fail to evolve with dynamic business requirements are considered defective. Businesses will not be competitive and clients will quickly seek a more adaptable solution elsewhere. With a few notable exceptions, however, adaptability is one requirement that software development efforts have most consistently failed to deliver. In the meantime, the transition from one cycle of products to the next is time-consuming, costly and frustrating to all.

Insurance project managers and developers need to understand the primary reasons for such failures. Adaptability is not a natural side-effect of system development efforts; ideally, it should be designed into system architectures from the very beginning. Inflexible software design leads to difficulty in manipulating (adding, deleting, changing or simply locating) business rules and processes. Over time, successive software modifications build up a texture of "patches upon patches" in these systems, leading to degradation in performance and quality. Each new change runs the risk of introducing more defects, more unintended side-effects. Systems tend to be fragile, rather than robust, if they are not well structured, easy to analyze and easy to test.

### **Adaptable Systems Design**

The concept of adaptable systems design is not new. Various related approaches have been described as table-driven, data-driven, rules-based, parameter-driven, dynamic, configurable, component-based, plug-and-play and expert systems. Depending on one's definition of a rule, all of these could be classified generally as rules based systems.

Adaptable software systems are developed today by many insurance technology vendors, proving that even complex applications can be designed to be robust in the face of change.

Bob Becker, from Foremost Insurance, part of the Farmers Insurance Group, describes how table-driven design helps to process 30 million database calls every day for 1.2 million individually tailored customer policies: “You can’t do business with that volume of calls without using a table-based approach”. Foremost uses a product called tableBASE®, from [Data Kinetics Ltd.](#), to design flexible insurance applications.

“When we’re creating applications,” indicated Stephanie Grady, Data Administration Analyst at Foremost, “we find that a table-based approach gives us incredible flexibility. I think this is the most important single benefit. Because it’s easier to change the business rules in the system, our development cycles are faster. Changes are easier to implement, so we save time and money. I’m proud to say that we’re designing new applications as fast as our business needs require! I would say that anyone who has information that changes quickly will benefit from a table-based approach.”

“Quality is another issue that is of great importance to a company like ours, which has a very high reputation in the market,” commented Dave Kempisty, Senior Architect. “The mere fact that we are recompiling the program could change it. We want to ensure that software logic errors are reduced. Because the business rules are stored outside the software program, our technical staff can spend less time fixing small bugs. Also, lengthy software development cycles are avoided, so software update and maintenance costs are reduced. Overall, this means reduced costs because the software development cycle is streamlined. The increase in quality means better customer satisfaction—even for our internal clients—because their needs are met more quickly, with less time and money spent fixing mistakes.”

“We have achieved tremendous advantages over our competitors,” said Becker. “We have been able to cut down on people costs and errors. We are able to create new products as fast as the business needs require. We never have a situation where we have to disrupt customers”. In attributing kudos for the high reliability of their systems, Becker adds, “The excellence of our people is a big part of that reason—and so is our use of tableBASE® software.”

### **Separation of Stable Processes and Dynamic Parameters**

Chaos Theory, a scientific discipline devoted to the study of dynamic systems, provides a simple theoretical metaphor for the development and testing of adaptable systems. Chaos Theory has demonstrated that even well defined systems with complex interactions (such as those in the insurance industry) can produce effects which are not predictable in advance by any straightforward analysis. On the positive side, Chaos Theory also suggests that apparently random aspects of such systems do in fact have high level foundations which remain stable for long periods of time. The trick is to distinguish what parts of an insurance process, for example, are inherently stable (and thus suitable for

implementation in program code) from more dynamic parameters in need of periodic tailoring (more appropriate for an external table of business rules).

The immediate objective is to avoid applying similar changes repetitively to software for changes in the business which are clearly predictable in nature. Consider the benefits, for example, of creating a rate table containing a series of insurance rates associated with various customer profiles. If the rates or customer profile details change, the table might only need to be updated and tested by an authorized agent. Contrast this with a program which contains the same information in a hard coded form. Every time a rate or customer profile changes, the system development group has to be notified, a project initiated, the program must be modified, recompiled, retested and finally re-released into production.

“We originally bought the tableBASE® product to get away from hard-coding data,” recalls Becker. “We wanted flexibility. In the past, when changes were made to hard-coded data, we needed to change the entire application. Now, we can spend our time developing products that provide strategic advantages to the company.”

### **Benefits for Legacy Systems**

Given the historical development of insurance systems, a complete redesign and redevelopment effort to accommodate more modern, adaptable modeling techniques is often not practical, nor is it necessary. Long established legacy systems can be redesigned at any level of detail on a piecemeal basis, component by component, as candidate processes for adaptability are identified and generalized.

In a recent article from [Insurance Networking News](#) titled, “Technology Helps Best Practices Become Business Practices”, Jamie Bisker, research director for the insurance practice at TowerGroup, states: "One of the ways to ensure consistency with technology is through the use of rules-based technology, such as expert systems”.

Bisker describes efforts being made in insurers' IT shops to extract business rules from legacy system code into a middleware layer also accessible to new front-end technologies. "Then, instead of a programmer having to modify the legacy system to respond to a change in business, those business rules and best practices move to a layer that can be easily worked with and manipulated independently."

Some rules based systems compile rules from stored language constructs of the form:

IF [condition set] THEN [perform some set of actions]

New rules can be added in an ad-hoc, step-wise fashion, in any order, with a wide variety of condition sets. At execution time, the rules are scanned serially for appropriate prevailing conditions and all relevant rules are “fired”. This organization can be quite effective for new development of systems with relatively stable business rules. Unless the rules base is well structured, however, manually searching the rules base in order to locate and change specific conditions or actions may be problematic. Ease of integration with legacy applications varies depending on the rules based product. Universal adaptability is not an automatic characteristic, even for rules based systems.

Rules may also be represented in collections of tables relating columns of conditions with associated actions. Automated capabilities are available for table structures to provide customized searching of the rules base under various tailored organizations, along with a host of other features for manipulating rules. A high performance table management product like tableBASE® facilitates the integration of legacy systems with middleware tables of parameterized business rules. Business rules are accessed and executed efficiently and are maintained easily through a front end interface, independent of the legacy system code. In addition, a tabular set of rules sorted by condition sets may be scanned manually by a developer or tester with relative ease.

### **Empowering the User**

Insurers cannot implement flexible business processes within a framework of rigid software systems. They must be able to tailor complete end-to-end processes effectively and efficiently. If insurers step back and look at their end-to-end business processes, they will recognize many places where those processes could be improved. It should be a natural next step to locate and improve the corresponding automated versions of those processes. Technologies should implement and manipulate business rules using an agent that mirrors human managers of the business process - so directly that an authorized business expert can change the implementation for other users as easily and as securely as she herself adapts to the changing business process.

Inefficiency and the chance of error both increase with the number of contact points in any processes that involve agents, insureds, claimants, or other third parties. As one of those "other third parties", insurers' IT shops strive to empower their clients in adapting insurance systems to match changing business rules. Adding new insurance business rules for exceptions or tailoring rules for regulatory or competitive purposes should be a responsibility of the insurer, a business expert, not a programmer. Many enhancements become easier using this approach, especially those where pre-conditions and resulting actions are already included in the inventory of rules recognized by the system; the new or changed rules simply specify actions which are executed under different sets of previously recognized conditions.

The very idea of allowing a non-programmer (business expert) access to system functionality is usually enough to spread panic through system administrators and business managers alike. Allowing business experts direct access to system driving tables, however, is not equivalent to uncontrolled access. First, these are the same people who originate the user requirements for any traditional system modification. Only authorized business administrators would have appropriate permission for these changes. Second, all the usual phases of testing would still be followed in isolated test environments prior to implementation in the field.

New types of conditions, new actions and entirely new classes of business rules - not supported in the original design of the system - can only be implemented by additions to

the generalized system infrastructure. These extensions to functionality may always require new code to be written by a developer.

### **Iterative Development and Rules Based Systems**

In 1937, [Dr. Joseph Juran](#) recognized a universal principle he called the “vital few and trivial many”. Juran’s principle, also known as the “80/20 Rule”, may be paraphrased as: 20% of something is responsible for 80% of the results. One derivative of this rule, for example, might be stated as: 20% of the program code for an insurance system is responsible for handling 80% of the daily user requirements (i.e., the most common use cases). In other words, 80% of the time, clients only exercise 20% of the system functionality. A client may well require a wide range of functionality over the life of the system, but most of it is not used on a day-to-day basis.

Unfortunately, development teams today consistently expend enormous effort up front to program, test and maintain complex volumes of code for exceptions that may rarely (or never) be exercised over the life of the system. Design and coding for every unique exception implies new opportunities to introduce software defects. In addition, over time, new code added to traditional systems by different programmers with different programming styles tends to degrade the overall quality of the system. In the Insurance Networking News article, “Technology Helps Best Practices Become Business Practices”, Alan Pelz-Sharpe, vice president of research firm Ovum, says "Gains can be made in the claims process in particular. People aren't very good at filling in forms and explaining themselves, and it's the exceptions that cost the money. The general rule of thumb is that your exceptions cost you 85 percent of your actual costs for running a process."

Rules based systems are particularly amenable to incremental development approaches. Productivity gains may be delivered quickly through an initial working prototype, followed by stepwise refinements in subsequent versions. The initial implementation of a rules based system may be incomplete but still very useful. For example, a collection of rules intended to address only 20% of all use cases may represent 80% of a user’s daily business requirements.

"Carriers often find that they have an underwriting rule that says one thing, but the end data showed that in a particular instance they did something different," Bisker says. "You then ask underwriting who says, 'That's ok, it's an exception to the rule.' Well, that exception itself is a rule you need to capture." The question now becomes, “When do you capture it?” The most “important” rules include those that the user is most likely to encounter on a daily basis, along with those which have the greatest impact on the business. These are the rules that have to be captured for the initial release. As other exceptional circumstances are encountered during day-to-day operations, the system may be enhanced incrementally (i.e. adapted to the changing business requirements) by adding or changing rules. In many cases, no actual program code needs to be modified and there is no associated need for comprehensive regression testing.

An initial implementation of the system that is acceptable to the client may be delivered very quickly **if**:

- The system includes the “right” 20% of the rules - those which truly address the users’ most important requirements. These are the high risk rules for development and testing.
- The system works satisfactorily most of the time, with a minimum of problems reported. Most clients are more concerned about productivity than absolute perfection.

A failure to recognize these simple points has proven fatal to many rules based development projects which had initially demonstrated impressive potential.

Additional rules may be inserted in the rules base over time as the need arises, to address new combinations of conditions encountered in daily operations, or as part of planned system enhancements. If the new rules simply execute existing functionality under new combinations of recognized conditions, then no system code has to be developed and the enhancement can be accepted with minimal testing. Updates to configuration tables and user profiles, for example, usually require only minimal testing after the initial system development is completed.

It may well be that this iterative development process never ends, but continues throughout the life of the system.

### **Testing Rules Based Systems**

An expert system is a rules based system which uses a generalized inference engine. This is a generic mechanism for navigating through the rules base and executing appropriate actions for a set of task-specific conditions, a mechanism that may be shared by multiple rules based applications. It’s extremely important that this engine be thoroughly tested, since the impact of defects in the system infrastructure will extend to all rules and all associated client applications. Fortunately, the severe impact of a faulty inference engine usually makes testing relatively simple. The generalized process of establishing search criteria based on system inputs, retrieving rules data and executing a rule either works for every rule or it doesn’t work at all. Each new application built using the same inference engine benefits from a stable infrastructure which has been proven in the field by preceding development efforts. An inference engine is usually purchased from an expert system vendor and tested as a Commercial Off-The-Shelf (COTS) product.

Some expert systems may be difficult to reconcile with legacy applications, however; a hybrid architecture may offer a more suitable alternative.

In the case of a high performance table management system such as tableBASE®, the application program, developed in-house, provides the inference logic while tableBASE® manages the organization and search capabilities for the rules base in a hybrid architecture. The application-specific inference logic is subject to normal system testing,

while the table manager itself may be tested as any other COTS product. As a mature table management system installed and in regular use at many Fortune 500 organizations, tableBASE® has been – and continues to be – field proven under the most extreme conditions.

In either case, since the hybrid application program or expert system inference engine is completely data driven and isolated from the rules base, each business rule is structurally independent of all other rules. This is significant for testing because it means that the act of integrating new rules into the system is a low risk endeavor, at least as far as structural impact is concerned. Integration testing can proceed in an iterative manner, closely paralleling the iterative development process. Of course, new rules may have a functional effect on other rules. Potential inconsistencies in the rules base must be addressed at the time requirements and specifications are formulated, early in the development effort. This applies equally to rules based systems and to traditional procedural systems.

There is a direct correspondence between functional specifications and rules in a rules base. The rules base represents an inventory of implemented specifications. Every specification is mapped to one or more rules. Valid test cases are then created, first and foremost, to verify results for those specifications. Conversely, if there is no specification for some behaviour, then there is no associated rule in the rules base to trigger that behaviour.

There is relatively little difference between testing specifications for a flexible, adaptable rules based system and a more rigid, hard-coded procedural system. A significant benefit may be noted, however, when considering the number of test cases in a comprehensive “negative” test effort. The term “negative testing” may be used to describe test cases for behaviour which may or may not be explicitly prohibited in the specifications, but which is nonetheless unacceptable.

In traditional systems, developers write code to handle known error conditions. They attempt to exclude everything that matches expectations for invalid inputs. Each exception is handled uniquely. Each exception must be tested individually. Beyond functional specifications provided for identified exception processing, there are very few guidelines to assist in the selective development of useful negative test cases for traditional systems. Many combinations of inputs are not explicitly addressed by typical specifications, and there are no expected results for these combinations which may be prioritized and tested according to risk. Testers are usually left with ad-hoc approaches to negative test case design. Rules based systems, on the other hand, exclude everything that does not match expectations for valid inputs. Unspecified rules are clearly associated with entire condition sets (search keys) which are simply missing in the rules base. This “incompleteness” of the rules base can be used to the advantage of system testers.

In the case where some valid set of conditions has not yet been addressed in an iterative development process, a missing rule is a normal and expected situation. When it occurs, a default action is invoked. In a hierarchical (or otherwise partitioned) rules base, there

may be various default actions for different “classes” of rules. Each class of rules (each table) has a default action for condition sets which are not found. One exception handling routine applies to many exceptions; exceptions are no longer unique and testing is simplified. The system designers may even take advantage of the default actions to trigger an enhancement request for the rules base. The key point here for testing is that all the conditions for activation of a rule are localized in the rule itself. If any single condition in the set is not satisfied, the rule does not fire. Unlike traditional procedural systems, no partial actions can be executed for a rule in the rules base. If no rule satisfies all conditions in the set, then the search fails and a default non-match action is executed. No other processing takes place. Rule activation is predictably “all or nothing”. Fault (missing rule) and failure (default action) are precisely defined for all negative test cases. This is a significant testing advantage over procedural systems.

Conversely, in a traditional procedural design, partial actions can be executed prior to a failure. Depending on the faulty business rule, the associated failure could occur at an unpredictable location in the code.

In the case of a hybrid procedural / rules based system such as those which use tableBASE®, the middleware layer is used to search for matching rule conditions in the rules base, and the application invokes the appropriate rule actions. Rule actions are themselves procedural components within rules based systems. Testing advantages may be drawn from the separation of business rules and program infrastructure, but the usual cautions apply for all procedural components. Developers should provide the test group with complete functional specifications for all procedural components and a complete list of business rules in the rules base for verification of completeness and consistency.

### **The Rules Based Journey**

Development and testing of adaptable systems may proceed incrementally, from an initial rudimentary system toward a fully specified pot of gold at the end of the development rainbow. Unlike a rainbow, our moving target of complete specifications is no mirage but it often remains just as distant, forever on the horizon. Fortunately for developers and testers of adaptable rules based systems, the journey is often more important than the destination.

### **Bibliography**

1. *Technology Helps Best Practices Become Business Practices*, Insurance Networking News, The Thompson Corporation  
<http://www.insurancenetworking.com/supplements/2003maxit/tech.cfm>
2. Brooks Jr., Frederick P., *Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975, ISBN 0-201-00650-2
3. *Insurance Industry Milestone Marks “Coming of Age” for America’s Love of Mobile Lifestyle*, Data Kinetics Ltd.
4. Data Kinetics Ltd. Product Information

<http://www.dkl.com/>

5. Juran Institute web site

<http://clk.about.com/?zi=1/XJ&sdn=management&zu=http%3A%2F%2Fwww.juran.co>

6. Cunneyworth, Wayne, *Table Driven Design – A Development Strategy for Minimal Maintenance Information Systems*, Data Kinetics Ltd., 1994