

Special Issue

Six Sigma Applied Throughout the Lifecycle of an Automated Decision System

Angie Patterson, Piero Bonissone*[†] and Marc Pavese
General Electric Global Research, Schenectady, NY 12309, U.S.A.

Automated decision-making systems have been deployed in many industrial, commercial, and financial applications. The needs for such systems are usually motivated by requirements for variation reduction, capacity increase, cost and cycle time reduction, and end-to-end traceability of the transaction or product. Before we can use any automated decision-making system in a production environment we must develop a strategy to insure high quality throughout its entire lifecycle. We need to guarantee its performance through a rigorous Design for Six Sigma process (DFSS). This process includes validation, tuning, and production testing of the system. Once the system is in production we must monitor and maintain its performance over its lifecycle. In this paper we will outline the Six Sigma process that led to the deployment of an automated decision-making system in one of the General Electric Financial Assurance businesses. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: Six Sigma; artificial intelligence; soft computing; fuzzy logic; evolutionary algorithms; classification; neural networks; automated decision-making

INTRODUCTION TO AUTOMATED DECISION TECHNOLOGY

The technology used in these decision systems comes from a broad base of work in the fields of computer science, mathematics, finance, management science, and statistics. The information requirements and decision complexity handled by these techniques can vary widely by application. These applications can vary from common, repetitive transactions (such as the approval of credit purchases) to the risk assessment and underwriting of complex insurance products. The selection of the supporting technologies depends on many factors, from information requirements (*can the inputs be described in a metric space?*), to its output characteristics (*is the output a discrete or continuous value?*), to design constraints and trade-offs that might prevent the use of specific technologies. Furthermore, the development of a decision engine is only the first step in a longer lifecycle process that covers the monitoring, updating, and maintenance of such engines.

In this next section we will describe some of the design trade-offs, analyze an illustrative example in using automated decision technology within an insurance underwriting problem, and explore the lifecycle challenges of such a system. In the remaining sections, we will then describe the Six Sigma framework that was actually used for its development. We will use a Design for Six Sigma (DFSS) approach for the *Build* stage of the system,

*Correspondence to: Piero Bonissone, General Electric Global Research, Schenectady, NY 12309, U.S.A.

[†]E-mail: bonissone@crd.ge.com

and a Design–Measure–Analyze–Improve–Control (DMAIC) approach for the *Use & Monitor* and *Maintain & Update* stages that complete the lifecycle of the system.

THE LIFECYCLE OF A DECISION SYSTEM

Design trade-offs

In the development of any type of decision algorithm, we usually face several design trade-offs. Among the most common, we find:

- (1) accuracy versus coverage;
- (2) accuracy versus interpretability;
- (3) run-time efficiency versus configuration-driven architecture.

These trade-offs are always present no matter what the application of decision engine technology. In the define phase of the project, enough insight into the requirements of a specific application must be gathered in order to be able to make the appropriate trade-off for that situation. In the DFSS build process for the insurance underwriting engine described below, each of these trade-offs was determined in this way.

The first trade-off is similar to the *precision versus recall* compromise found in the design of information retrieval systems. We could tune a classifier to maximize its number of correct decisions, declining to make any commitment if we are not confident about the conclusion. This behavior would increase accuracy at the expense of coverage. Alternatively, we could tune the same classifier to always issue a decision for each probe, increasing coverage at the expense of accuracy.

The second trade-off, typically dictated by legal or compliance regulations, constrains the underlying technologies used to implement the classifier^{1,2}. In our approach we have used soft computing (SC) techniques, a collection of computational paradigms (probabilistic, fuzzy, neural, and evolutionary) in which the equation '*model = structure + parameters*' takes a different connotation, as we have a much richer repertoire to represent the structure, to tune the parameters, and to iterate this process³. This repertoire enables us to choose among different trade-offs between the model's interpretability and its accuracy. For instance, one approach aimed at maintaining the model's transparency usually starts with knowledge-derived linguistic models, in which domain knowledge is translated into an initial structure and parameters. The model's accuracy is further improved by using global or local data-driven search methods to tune the structure and/or parameters. An alternative approach, aimed at building more accurate models, might start with data-driven search methods. Then, we could embed domain knowledge into the search operators to control or limit the search space, or to maintain the model's interpretability. Post-processing approaches can also be used to extract explicit structural information from the models.

The third trade-off is related to the use of configuration files to drive the behavior of the classifiers, instead of hard-coding their logical behavior. The essential idea here is that the actual coded software would implement a more *generic* approach to solving the problem, which could then be specialized not within the code itself but by reading parameters from a configuration file. In fact, any external data source, such as a database table, could also be used to supply engine parameters. While slightly less efficient at run-time, the use of a common decision engine driven by configuration files produces a more maintainable classifier than one whose parametric values are intertwined in the engine's code. This additional computational cost is introduced for the purpose of lifecycle benefits. Its software implementation is described in Aggour and Pavese⁴.

Description of the underwriting classifier for an insurance product

To illustrate the key issues in developing and deploying an automated decision engine (ADE), we will use a representative, challenging classification problem: the process of underwriting insurance applications. Insurance underwriting is a complex decision-making task that is traditionally performed by trained individuals. An underwriter must evaluate each insurance application in terms of its potential risk for generating a claim,

such as mortality in the case of term life insurance. An application is compared against standards adopted by the insurance company, which are derived from actuarial principles related to mortality. Based on this comparison, the application is classified into one of the risk categories available for the type of insurance requested by the applicant. The *accept/reject* decision is also part of this risk classification, since risks above a certain tolerance level will typically be rejected. The estimated risk, in conjunction with other factors such as gender, age, and policy face value, will determine the appropriate price (premium) for the insurance policy. When all other factors are the same, to retain the fair value of expected return, higher risk entails higher premium.

We represent an insurance application as an input vector \bar{X} that contains a combination of discrete, continuous, and attribute variables. These variables represent the applicant's medical and demographic information that has been identified by actuarial studies to be pertinent to the estimation of the applicant's claim risk. Similarly, we represent the output space \bar{Y} , e.g. the underwriting decision space, as an ordered list of rate classes. Due to the intrinsic difficulty of representing risk as a real number on a scale, e.g. 97% of nominal mortality, the output space \bar{Y} is subdivided into bins (rate classes) containing similar risks. For example 96–104% nominal mortality could be labeled the *Standard* rate class. Therefore, we consider the underwriting process as a discrete classifier mapping an input vector \bar{X} into a discrete decision space \bar{Y} , where $|\bar{X}| = n$ and $|\bar{Y}| = T$.

This problem is not straightforward due to several requirements.

1. The underwriting mapping is *highly nonlinear*, since small incremental changes in one of the input components can cause large changes in the corresponding rate class.
2. Most inputs require *interpretations*. Underwriting standards cannot explicitly cover all possible variations of an insurance application, causing ambiguity. Thus the underwriter's subjective judgment will almost always play a role in this process. Variations in factors such as underwriter training and experience will likely cause underwriters variability in their decisions.
3. These interpretations require an intrinsic amount of *flexibility* to preserve a balance between *risk-tolerance*, necessary to preserve price competitiveness, and *risk-avoidance*, necessary to prevent overexposure to risk.
4. Legal and compliance regulations require that the models used to make the underwriting decisions be *transparent* and *interpretable*.

To address these requirements we decided to extend traditional artificial intelligence (AI) reasoning methodologies, such as rule-based and case-based reasoning, coupled with SC techniques, such as fuzzy logic and evolutionary algorithms. With this hybrid system, we were able to provide both flexibility and consistency, while maintaining interpretability and accuracy as part of an underwriting and a risk management platform. Our DFSS process for building such a system and our DMAIC process for maintaining it are described below.

Importance of building in quality: the lifecycle perspective

In real-world applications, before we can use a classifier in a production environment we must address the classifier's entire lifecycle, from its design and implementation, to its validation, tuning, production testing, use, monitoring, and maintenance. A lifecycle diagram can be seen in Figure 1. By maintenance we mean all the steps required to keep the classifier vital (e.g. non-obsolete) and able to adapt to changes. Two reasons justify our focus on maintenance. Over the lifecycle of the classifier, maintenance costs are the most expensive component (as software maintenance is the most expensive lifecycle component of software). Secondly, when dealing with mission-critical software we need to guarantee continuous operations or at least fast recovery from system failures to avoid incurring a loss of revenue and other business costs. An obsolete classifier is as useful as one that can no longer run under a new version of the operating system. We will focus on a design methodology that will explicitly address classifiers' obsolescence, with the aim to facilitate their maintenance.

In this paper we will describe the overall classifier's lifecycle, with a specific emphasis on our approach to design for maintainability and on a Six Sigma roadmap that we follow throughout that lifecycle.

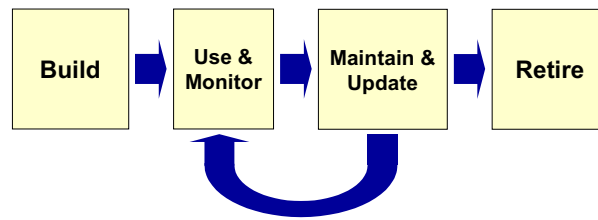


Figure 1. Lifecycle of an ADE

A SIX SIGMA ROADMAP

The DMAIC and DFSS roadmaps

Six Sigma provides a rigorous roadmap for quality improvement. When the objective is to improve an existing product or process, the roadmap consists of five critical steps: Define, Measure, Analyze, Improve, and Control. The primary deliverables of each of these five steps are as follows:

1. *Define*—identify the customers and what is important to them about the product/process.
2. *Measure*—translate customer wants into quantifiable ‘critical-to-quality’ metrics (CTQs) and validate your ability measure them accurately and precisely.
3. *Analyze*—assess how well the current product/process meets the performance desired by the customer; quantify performance improvement goals.
4. *Improve*—identify the key drivers preventing the product/process from meeting performance goals and correct them.
5. *Control*—validate that improvements have resulted in meeting performance and put mechanisms in place to ensure the improvement will be sustained.

When the objective is to develop a new product/process, then following DFSS discipline results in a new design that meets the expectations of its customers at a level of Six Sigma performance. The level of rigor in DFSS is equivalent to that described by the DMAIC roadmap. Although the detailed GE roadmap for DFSS is proprietary, in general the essential components can be described as follows:

1. identify customers and quantify their needs (CTQs);
2. evaluate/select technology to meet CTQs;
3. Build and optimize design prototype;
4. deliver production-ready robust system with Six Sigma quality.

A DFSS + DMAIC roadmap for the ADE lifecycle

As depicted in Figure 1, the full lifecycle of an ADE consists of four basic phases: Build, Use/Monitor, Maintain/Update, and Retirement. During the first phase, the ADE technology must be designed and built with the discipline of DFSS, to ensure that the engine that is delivered to production meets all of the customers’ needs. Once in production, the ADE must be maintained and improved over time to ensure a long and prosperous lifecycle of the business’ technology investment. Since the DMAIC roadmap was developed specifically to improve existing processes, it seems only natural to combine the DFSS and DMAIC roadmaps into an extended roadmap (shown in Figure 2) that can be used to ensure Six Sigma quality throughout the lifecycle of the ADE. The remainder of this paper will focus on the application of this combined roadmap and the quality technology that was employed along the way.

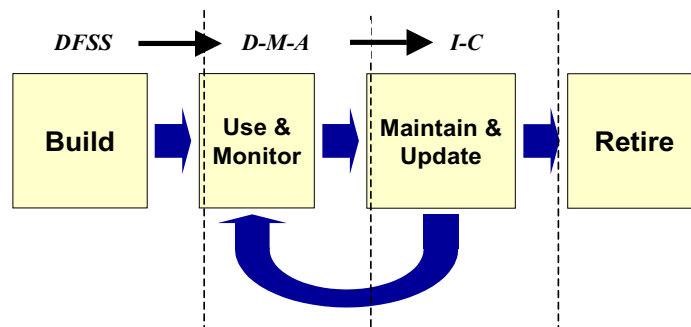


Figure 2. DFSS+DMAIC roadmap for the lifecycle of an ADE

DFSS

Building the ADE

Identification of CTQs

As described above, the goal was to create a decision engine for automated underwriting of insurance. To further specify the problem, a set of measurable requirements (CTQs) was gathered, as is always done in the Define phase of a Six Sigma project. The Define phase is one of the keys to a successful DFSS process. The main questions we are trying to answer here are ‘What do I want to do?’ and ‘How will I know I did it?’. Without being able to articulate the answers to these questions, the decision engine Build will be at much higher risk of failure. For decision engines, these requirements are usually of the following types.

Model CTQs:

- related to the operation of the actual decision making model;
- requirements defining lower bounds on:
 - *scope*: a specified subset of the total population that is eligible for passing to the engine;
 - *coverage*: within the scope, the fraction which the engine must be able to decide on;
 - *accuracy*: percentage of correctness for the cases where the engine makes a decision.

Information technology (IT) CTQs:

- integration with a certain IT or computing environment;
- speed of decision making;
- uptime/reliability metrics.

Maintainability CTQs:

- time required to update the system with anticipated types of changes;
- resources/effort required to do the same.

In the present case, the target scope of the engine was determined to be a significant percentage of the total population. Engine coverage was specified to be all cases that could be placed into rate classes *Standard* or better. Sub-standard cases are ‘punted’ by the engine and handled by decision support. It was expected that this sub-standard group would be no more than 10% of the cases within the engine’s scope, but the exact statistics would not be known until the data collection phase of the project. The accuracy CTQs are given in the following way:

- (i) the engine must have an accuracy rate equal to or better than current manual underwriting;
- (ii) the expected net present value (NPV) of the cases underwritten by the engine must be equal to or better than the expected NPV of cases underwritten by the current manual process as well.

The reason for the two CTQs was that simple accuracy counts were not sufficient for the project champions—it could still have been true that rare mistakes made by the engine could have been so costly to the business that it would not want to deploy the engine. Therefore, accuracy was measured both in terms of percentage and dollar value (as measured by NPV). More will be said about the exact definition of these metrics below.

The IT CTQs were specified by the following. First, the enterprise IT system interacting with the engine would be Java-based system, necessitating the use of a Java framework for the engine. Communication between the system and the engine would be via the Enterprise Java Bean (EJB) communication protocol⁵. An additional challenge in this case was that this IT system was being developed concurrently with the engine. Therefore, fulfillment of the IT CTQs for the engine would be verified by specifying a suite of cases for which the engine's behavior was known. (This suite would be the same set of cases on which the model CTQs—scope, coverage, accuracy—were verified.) These cases would be used in 'on-line' testing of the engine integrated with the surrounding IT system. A successful pass of all these tests would verify that the IT CTQs were met.

In terms of maintainability, the primary CTQ was for business users with functional knowledge (i.e. underwriters and actuaries) to be able to maintain the knowledge base of the engine. To verify this, a checklist of maintenance activities was developed, listing all the generic activities necessary to maintain the engine after deployment. During the final knowledge transfer sessions, this checklist would be used to confirm that the team selected to maintain the engine was familiar enough with each topic to perform the task without help. This would be verified by defining 'homework' problems for each task. The development team would sign off that the homework was done correctly, and the business maintenance team would sign off that they were able to perform the tasks without help and that the process was not overly cumbersome.

We also needed to maintain full accountability, i.e. transparency of the production model decisions. This CTQ was a legal requirement, imposed by most state insurance commissioners, mandating insurance companies to notify their customers and justify the reasons for issuing policies that are not priced at the most competitive rates. The verification that the engine would meet this CTQ consisted of the ability to correctly translate the engine's outputs into the actual language in a letter that would go to the customers.

Data gathering

With those CTQs specified, the next important task was to gather data that would be used to create the decision model, and to verify that the CTQs were being met. Data is critical to the Six Sigma process for decision engines because the data is the measurement system for the model (which can be seen simply as a transfer function from X 's to Y 's—the X 's being the decision engine inputs and the Y 's being the decision engine outputs). There is no other way to test and validate a decisioning model than with data.

It is important to try to capture all the critical X 's for the decision-making. The widest set of X 's to be considered for the model was initially developed in a brainstorming session with the business subject matter experts. Next, one must discover where that information is stored, and how it can be obtained in a digital format.

In the present case, the answers to these questions exposed some gaps in the present day information system. When the project was started, not all of these X 's were digitally available. (As mentioned above, a parallel IT effort was underway to improve the system.) Therefore, we had to design a data gathering process. This involved taking a random sample of historical cases, and having the relevant data from those cases entered into a temporary database to be used for analysis. In addition to the X 's to be used as inputs to the decision engine, the actual historical decisions (Y 's) on these cases were also available, which was critical for development of the engine.

Although resources were available to capture close to 3000 cases, an initial set of 1200 cases was obtained. From past experience, it was clear that we should not spend our entire 'data budget' at once before looking at an initial set. It is never possible to predict, beforehand, what the data needs of the project will be, so a multi-phase approach was taken.

This data was then assessed using data mining techniques such as CART (Classification and Regression Trees)⁶ to determine the relevance of each X gathered, as a predictor of the final decision. This exercise yielded several key findings. First, none of the X 's could be eliminated right away as being irrelevant.

Second, certain types of case were under-represented in the data, particularly those that, although within the engine's defined scope, were placed into worse risk categories (i.e. higher insurance premiums, closer to standard rates). Third, since these data would be fundamental to the measurement system to validate that the CTQs were met, we needed to perform a measurement system analysis (MSA) to confirm its ability to accurately measure the performance of the engine.

The first issue was simple to accommodate: in the initial engine designs, all X 's would be incorporated. The second issue was addressed in the next round of data collection. A stratified random sample was used to include those cases that were under-represented in the first sample. Since the resulting sample would not be representative of the population of incoming applicants, the resulting performance metrics on the automated decision engine would not be representative of this population either. To overcome this problem, the performance metrics would be measured on each stratum separately. The expected performance on the incoming population would be calculated utilizing the population proportion of the strata to reweigh the importance of each stratum in the sample.

The third issue necessitated the creation of a standard reference (SR) procedure by which the data itself could be tested. Briefly, this procedure, which we defined with the subject matter experts, called upon a committee of experienced underwriters to conduct a blind review of a sample of cases from our database and arrive at a 'consensus' decision for each case, referred to as the SR decision. The resulting dataset containing all of the standard reference decisions (SRDs) was used in the MSA of the original data. The SRDs also served as the benchmark decisions against which the decisions of both the engine and the current underwriting process could be independently compared. Without the SRDs, there would have been no way to measure the accuracy of the automated decision engine. The SRDs were used during the building of the engine as well. As will be described below, we used a cross-validation approach⁷ in which the full data set was divided into multiple segments, with each segment being used to validate the accuracy of the decisioning model built based on the other segments.

As a final point about the data collection process, the additional set of X 's, which were not presently digitally captured, were documented and supplied back to the team responsible for expanding the information systems. It was made clear exactly what data the decision engine will need, any range restrictions or cross checks to be applied, etc. These issues are *interface* issues. The point to understand is that the decision engine will always be used as part of a larger *system*. The *system* is the entity that can actually produce business results. The decision engine may be an important, even critical, component in that system, but it does not 'stand alone'. It has no value until it is integrated within that system and the system *as a whole* is performing and producing results. By keeping this 'system view' in mind while developing the decision engine, these interface issues are more likely to get the attention they deserve. This philosophy was used throughout the entire project.

Conceptual design

Once enough data were gathered, the next step was to begin to develop conceptual designs for the decision engine. There are many possible approaches derived from the AI and SC domains, such as rule-based systems, neural networks, case-based reasoning, fuzzy logic, as well as traditional approaches such as regression analysis or scoring models.

Technology selection

The underwriting problem was formulated as a discrete classification problem: creating a transfer function from (mostly) continuous X 's to discrete Y 's. Several modeling approaches were proposed and tested: logistic regression (LR)⁷, neural networks (NNs)^{8,9}, multivariate adaptive regression splines (MARSs)¹⁰, rule-based (RB) systems^{11,12} and fuzzy logic (FL)^{13,14}. Each approach has benefits and drawbacks. A RB system for underwriting already existed and was under patent. An additional drawback to the RB system was that, for a required level of accuracy, the RB could become quite complex and thus become unmanageable. The LR model was unable to show the required level of accuracy, owing to the strong nonlinearities in the classification problem: changes to the level of a single variable can cause huge changes to the final decision in one case but not in others, depending on the values of all other variables. The NN, MARS, and FL approaches were found

to be accurate enough to meet the model CTQs. However, MARS and NN systems have a drawback in that the interpretation of the model results is problematic. In most cases, it is difficult to say, in easy to explain terms, *why* the final decision was made. This model interpretability was a requirement in our case. Underwriting decisions of an insurance company must be explainable to an applicant, as well as defensible in court. This ruled out the NN and MARS approaches to the decision-making. The remaining FL model showed excellent initial results in terms of accuracy, and did not suffer from the interpretability problems of the others. The design and final accuracy results for the FL model will be discussed in the next sections of the paper.

This being said, we did not want to discard the benefits of these alternative models, some of which were capable of extremely high accuracy as well. This application calls for extreme accuracy to avoid underestimating the applicants' risk, which would decrease the company's profitability, or overestimating it, which would reduce the company's competitive position in the market. So, in different parts of this project we have strived to achieve balance between interpretability and accuracy. We selected the transparent hybrid SC models, like the FL model, for on-line production use, and opaque ones, like NNs and MARSs¹⁰, for offline quality assurance. The commonalities among these models are the tight integration of knowledge and data, leveraged in their construction, and the loose integration of their outputs, exploited in their offline use.

A remaining challenge was how to plan for ongoing maintenance. The first step to solving this issue was to design the engine so that additions and changes could be made without altering the actual engine software itself. Therefore, a completely configuration-file-driven design was chosen. No arbitrary limits were placed on the number of FL rules or the type of data they could consider. Therefore, the maintenance of the knowledge base of the model was separated from the maintenance of the software. Business users could be in charge of the former, while the IT staff would be in charge of the latter. The second step in planning for maintenance was to realize that business users would not always be able to select the optimal set of parameters for the FL model. They may be able to provide the structure of the model (e.g. that an applicant's blood pressure should influence their rate class assignment), and an initial estimate of the parameters, but setting the detailed parameters would be a challenge. To address this, a novel approach was taken, to couple the FL model with an evolutionary algorithm (EA) optimization^{15,16}, which could aid the business users in finding the correct set of model parameters to make the right decisions given a data set. Thus, business users would not have to worry about setting exactly precise parameters within the engine, but could simply set up the basic structure and allow the EA to perform the final parameter search and optimization. An additional benefit of this approach was that we could apply it right away to the SRD data set, to ensure that the decision engine was making optimal decisions even prior to its being put into production.

Designing the decision engine

The Fuzzy Logic Engine (FLE) uses rule sets to encode underwriting standards. Each rule set represents a set of fuzzy constraints defining the boundaries between rate classes. These constraints were first determined from the underwriting guidelines. They were then refined using knowledge engineering sessions with expert underwriters to identify factors such as blood pressure levels and cholesterol levels, which are critical in defining the applicant's risk and corresponding premium. The goal of the classifier is to assign an applicant to the most competitive rate class, providing that the applicant's vital data meet all of the constraints of that particular rate class to a minimum degree of satisfaction. The constraints for each rate class r are represented by n fuzzy sets¹³: $A_i^r(x_i)$, $i = 1, \dots, n$. Each constraint $A_i^r(x_i)$ can be interpreted as the *degree of preference* induced by value x_i , for satisfying constraint A_i^r ^{3,14}. After evaluating all constraints, we compute two measures for each rate class r . The first one is the degree of intersection of all the constraints and measures the lowest degree of constraint satisfaction[‡]:

$$I(r) = \bigcap_{i=1}^n A_i^r(x_i) = \text{Min}_{i=1}^n A_i^r(x_i)$$

[‡]This expression implies that each criterion has equal weight. If we want to attach a weight w_i to each criterion A_i we could use the weighted minimum operator $I'(r) = \bigcap_{i=1}^n W_i A_i^r(x_i) = \text{Min}_{i=1}^n (\text{Max}((1 - w_i), A_i^r(x_i)))$, where $w_i \in [0, 1]$.

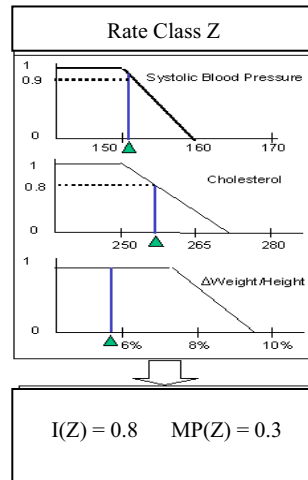


Figure 3. Example of three fuzzy constraints for rate class Z

The second one is a cumulative measure of missing points (the complement of the average satisfaction of all constraints), and measures the *overall tolerance* allowed to each applicant, i.e.

$$MP(r) = \sum_{i=1}^n (1 - A_i^r(x_i)) = n \left(1 - \frac{1}{n} \sum_{i=1}^n A_i^r(x_i) \right) = n(1 - \bar{A}^r)$$

The final classification is obtained by comparing the two measures, $I(r)$ and $MP(r)$, against two lower bounds defined by thresholds τ_1 and τ_2 . The parametric definition of each fuzzy constraint $A_i^r(x_i)$ and the values of τ_1 and τ_2 are design parameters that were initialized with knowledge engineering sessions.

Figure 3 illustrates an example of three constraints (trapezoidal membership functions) associated with rate class Z, the input data corresponding to an application, and the evaluation of the first measure, indicating the weakest degree of satisfaction of all constraints.

Optimizing the decision engine

The FLE design parameters must be tuned, monitored, and maintained to assure the classifier's optimal performance. To this end, we have chosen EAs, an optimization paradigm based on the theory of evolution and natural selection¹⁵. Our EA is composed of a population of individuals ('chromosomes'), each of which contains a vector of elements that represent distinct tunable parameters to configure the FLE classifier, i.e. the parametric definition of the fuzzy constraints $A_i^r(x_i)$ and thresholds τ_1 and τ_2 .

A chromosome, the genotypic representation of an individual, defines a complete parametric configuration of the classifier. Thus, an instance of such a classifier can be initialized for each chromosome, as shown in Figure 4. Each chromosome c_i , of the population $P(t)$ (left-hand side of Figure 4), goes through a decoding process to allow them to initialize the classifier on the right. Each classifier is then tested on all the cases in the test set, assigning a rate class to each case. In our case, the test set was composed of a subset of the SRDs. We can determine the quality of the configuration encoded by the chromosome (the 'fitness' of the chromosome) by analyzing the results of the test. Our EA uses mutation (randomly permuting parameters of a single chromosome) to produce new individuals in the population. The more fit chromosomes in generation t will be more likely to be selected for this and pass their genetic material to the next generation $t + 1$. Similarly, the less fit solutions will be culled from the population. At the conclusion of the EA's execution the *best* chromosome of the *last* generation determines the classifier's configuration^{15,17}.

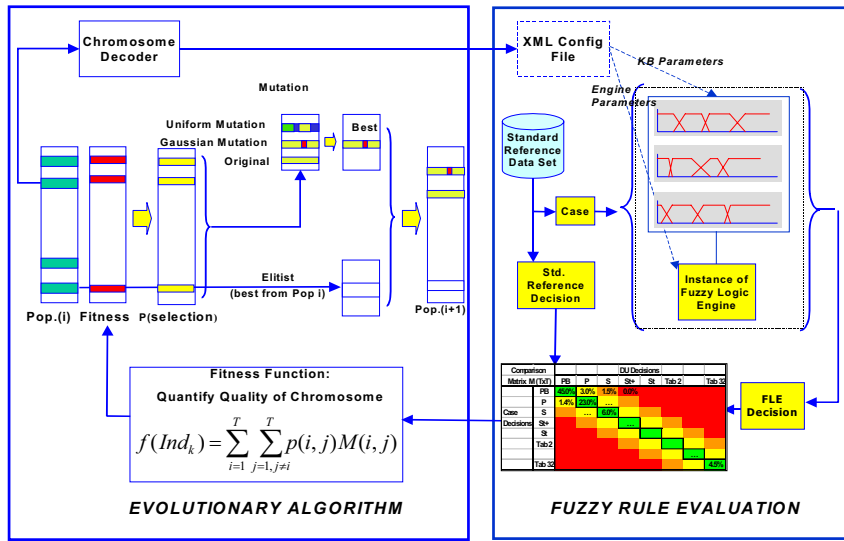


Figure 4. FLE optimization using EA

Fitness function

The fitness function plays a key role in the evolution, since it creates different selection probabilities, reflecting the quality of the solutions. The fitness is the performance function that we want to maximize (in the case of minimization problems, such as costs minimization, we maximize the negation of such functions). Typically, in continuous applications, such as predictions or estimations, we would use a measure of the cumulative prediction/estimation error. In discrete classification problems such as this one, we will use two matrices to construct the fitness function that we want to optimize. The first matrix is a $T \times T$ confusion matrix M that contains frequencies of correct and incorrect classifications for all possible combinations of the SRD and classifier decisions. The first $(T - 1)$ columns represent the rate classes available to the classifier. Column T represents the classifier’s choice of not assigning any rate class, sending the case to a human underwriter. The same ordering is used to sort the rows for the SRDs. The second matrix is a $T \times T$ penalty matrix P that contains the cost of misclassification. The fitness function f combines the values of M , resulting from a test run of the classifier configured with chromosome c_i , with the penalty matrix P to produce a single value:

$$f(c_i) = \sum_{j=1}^T \sum_{k=1}^T M(j, k)P(j, k)$$

Function f represents the overall misclassification cost.

Validating the decision engine performance

After defining measures of coverage and (relative and global) accuracy, we performed a comparison against the SRDs. The results, partially reported by Bonissone *et al.*¹⁸, show a remarkable improvement in all measures. We evaluated the performance of the decision systems based on the following three metrics.

- *Coverage*: percentage of cases as a fraction of the total number of input cases, whose rate class assignments are decided by a decision system. Each decision system has the option to not make a decision on a case, and to refer an undecided case to a human underwriter.
- *Relative accuracy*: percentage of correct decisions on those cases that were not referred to the human underwriter.

Table I. Performance of the un-tuned and tuned rule-based decision system (FLE)

Metric	Initial parameters based on written guidelines (%)	Best knowledge engineered parameters (%)	Optimized parameters (%)
Coverage	87.37	90.38	91.71
Relative accuracy	80.02	92.99	95.52
Global accuracy	75.92	90.07	93.63

Table II. Average FLE performance over five *tuning* case sets compared to five *disjoint test* sets

Metric	Average performance on tuning sets (%)	Average performance on test sets (%)
Coverage	91.81	91.80
Relative accuracy	94.52	93.60
Global accuracy	92.74	91.60

- *Global accuracy*: percentage of correct decisions, including making correct rate class decisions and making a correct decision to refer cases to human underwriters as a fraction of total input cases.

Specifically we obtained the results shown in Table I. Using the initial parameters (first column of Table I) we can observe a moderate Coverage (~87%) associated with a low Relative accuracy (~80%) and a lower Global accuracy (~76%). These performance values are the result of applying a strict interpretation of the underwriting guidelines, without allowing for any tolerance. If we were to implement such crisp rules with a traditional RB system, we would obtain the same evaluations. This strictness would prevent the insurer from being price competitive, and would not represent the typical *modus operandi* of human underwriters. However, by allowing each underwriter to use his/her own interpretation of such guidelines, we could create large underwriters' variability. One of our main CTQs was to provide a *uniform* interpretation, while still *allowing for some tolerance*. This CTQ is addressed in the second column of Table I, which shows the results of performing knowledge engineering and encoding the desired trade-off between risk and price competitiveness as fuzzy constraints with *preference* semantics. This intermediate stage shows that this new point dominates (in the Pareto sense) the first set, since both Coverage and Relative accuracy have improved. Although we obtained this initial parameter set by interviewing the experts, we have no guarantee that such parameters are optimal. Therefore, we used an evolutionary algorithm to tune them. We allowed the parameters to move within a predefine range centered around their initial values and, using the SRDs and the fitness function described above, we obtained an optimized parameter set, whose results are described in the third column of Table I. The results of the optimization stage show that the point corresponding to the final parameter set dominates the second point (in the same Pareto sense). Finally, we can observe that the final metric, Global accuracy (last row in Table I), improves monotonically as we move from using the strict interpretation of the guidelines (~76%), through the knowledge-engineered parameters (90%), to the optimized parameters (~94%). To evaluate the robustness of the optimization, we decided to use a five-fold, cross-validation method. This approach consists of dividing the data into five disjoint sub-samples, each containing 20% of the data. Each sub-sample is the test set for the model constructed from the other four sub-samples. This process is repeated five times. The average error from the five sub-samples represents an efficient and unbiased estimate of the error⁷. Using this five-fold, cross-validation process, we obtained stable parameters in the design space and stable metrics in the performance space. Specifically, we found the results presented in Table II.

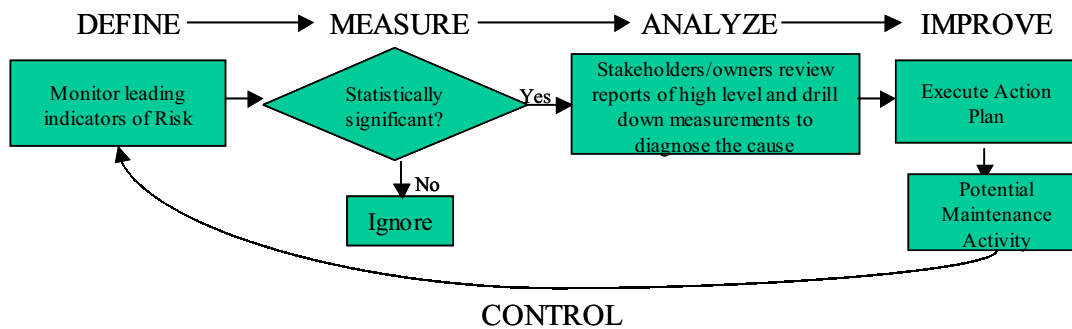


Figure 5. Framework for Six Sigma based monitoring and maintenance

DMAIC

Using and monitoring the FLE (Define–Measure–Analyze)

In the past, this stage has often received less attention than the build stage. It is easy to take the attitude that, once the engine has been built and deployed into the system, the ‘hard’ work is done and one can simply allow the engine to work as designed. The focus can easily transfer to the IT department, which is responsible for keeping the engine running, and doing routine upkeep such as cleaning up log files, etc. However, to leave it at this would ignore a large opportunity.

By applying the rigor of the Six Sigma DMAIC roadmap during the remainder of the engine’s lifecycle, we can minimize business risk and maximize the return on the investment to build the automated decision system. The basic framework is a statistical monitoring and maintenance/control plan, which is depicted in Figure 5.

Define

In defining a monitoring and control plan, one must answer the following questions.

1. What should be monitored?
 - What are the CTQs that must be monitored and what are leading indicators that there is degradation in the quality of the CTQs?
2. How should the metrics be monitored?
 - What are the appropriate statistical process control methods for these metrics?
 - Will the control charts be developed manually or will they be automated?
 - Will signals be obtained from viewing the charts manually, or via automated messages to the process owners?
3. What is the action plan to address signals? Who will own the actions?
4. How will the monitoring system be maintained? Who will maintain it?

A few comments will now be made about ‘what to monitor’. The CTQs that were defined during the Build phase must continue to be met over the lifecycle of the engine. Priorities must be set according to the level of business risk, which may be incurred if there is a decrease in the engine’s ability to meet these CTQs. A quality tool that is extremely useful for prioritizing plans to mitigate risk is the failure mode and effects analysis (FMEA), which the reader can learn more about from Breyfogle¹⁹. Two high-priority areas are the coverage and accuracy of the engine over time. For an insurance product, the engine coverage will vary with the population of applicants. A decrease in coverage results in a reduction in productivity, since decisions that cannot be automated must be made by the more time-consuming (and perhaps less accurate) manual process. Similarly, the accuracy of automated underwriting decisions may degrade if there is a decrease in the accuracy of data input to the engine. Decision accuracy may also be impacted if there is a shift in the applicant population

such that more complex underwriting decisions are required. For these reasons, the monitoring system for the FLE coverage and accuracy includes 'leading indicators' around data entry accuracy, applicant attributes, and the frequency with which each fuzzy rule is exercised in the underwriting decisions (to name a few).

Measure and Analyze

The monitoring system provides a signal when there is statistical evidence that there has been a significant change in one of the leading indicator metrics. It is at this point that further data analysis is required to understand the root cause for the signal, so that an appropriate maintenance action can be recommended (and carried out in the Improve phase). For automated decisioning systems, the root cause analysis will be likely to require a focused audit of the subset of engine decisions that appear suspect. For example, consider the following monitoring scenario and action plan when monitoring the distribution of insurance underwriting classes as determined by the FLE.

Step 1: The monitored distribution of insurance classes has shifted significantly (as indicated by the statistical process control chart).

Step 2: Control charts for the individual classes are examined, to identify specifically which classes have shifted in volume. Other statistical methods are applied to identify which subgroups within each class have shifted (e.g. if specific FLE rules are being exercised more often within the suspect classes).

Step 3: A focused audit is triggered to investigate whether the shift is real (applicant distribution has shifted) or if there is a problem with the FLE rules.

Step 4: If a problem with the FLE rules was detected, then the FLE maintenance owners are notified.

Step 5: All case data and auditors decisions for maintenance of the FLE test set are stored.

Maintaining and updating the FLE (Improve–Control)

Maintenance

A serious challenge to the successful deployment of intelligent systems is their ability to remain valid and accurate over time, while compensating for drifts and accounting for contextual changes that might otherwise render their knowledge base stale or obsolete. This issue has been an ongoing concern in deploying AI expert systems²⁰ and continues to be a critical issue in deploying knowledge-based classifiers. The maintenance of a classifier is essential to its long-term usefulness since, over time, the configuration of a FLE may become sub-optimal. Therefore, it is necessary to modify the SRDs to reflect these contextual changes. We developed specialized editors to achieve this objective. By modifying the SRDs to incorporate the desired changes (by altering some previous standard reference decisions), we created a new *target* for the classifier. Then, we used the same evolutionary optimization tools to find a new configuration (parameter values) for the classifier to *approximate the new target*.

It is equally important to monitor the classifier performance over time to identify those new, highly reliable cases that could be used to update the SRDs. To address this objective, we have implemented an offline quality assurance (QA) process, based on a fusion module, to test and monitor the production FLE that performs online rate classification. At periodic intervals, e.g. every week, the fusion module and its components will review the decisions made by the FLE during the previous week. The purpose of this fusion is to assess the quality of the FLE performance over that week. In addition, this fusion will identify the *best* cases, which could be used to tune the production engine, as well as *controversial* or *unusual* cases that could be audited or reviewed by human underwriters.

In our approach we designed the classifiers around a set of SRDs, which embodies the results of the ideal behavior that we want to *reach* during development and that we want to *track* during production use. The SRDs are the training and validation set for the classifier, and in general is a benchmark set that must be maintained over time. To this end we have proposed the use of a fusion module to update the SRDs with new cases, without resorting to manual screening²¹.

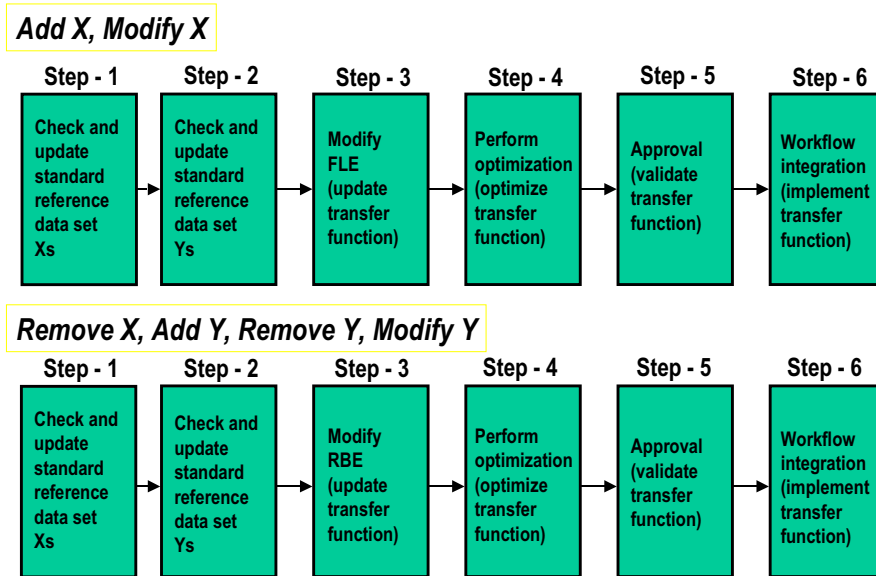


Figure 6. FLE maintenance and update process flow

Furthermore, during the life of the classifier we may need to change the underwriting rules. These modifications may be driven by new government regulations, changes among data suppliers, new medical findings, etc. Market considerations or actuarial studies may also suggest the creation of new rate classes, or the elimination or modification of current ones. These changes need to be incorporated into an updated set of SRDs via maintenance tools that identify the subset of cases whose decisions must be altered. The updated set of SRDs represents the new *target* that we want our classifier to approximate. At this point, we can use the same EA-based optimization tools, employed during the initial tuning, to find a parametric configuration that structures a classifier whose behavior better approximates the new SRDs. This lifecycle, and in particular the maintenance of the knowledge base, is critical to the long-term applicability of a classifier.

Updating

We have suggested many possible reasons for modifying the inputs, outputs, or classifier's transfer function. Now, we will illustrate the maintenance/updating process and its supporting tool. Any change to the model, inputs, or outputs will fall within one of the following six cases: *add* or *remove* an *input* (X), *add* or *remove* an *output* (Y), or *modify* an *existing input or output*. For any of these cases the underlying model (transfer function) will also need to be modified appropriately.

Figure 6 illustrates the process for each case. For instance, let us assume that we want to *modify an input*, such as increasing the influence that a given input variable (e.g. cholesterol ratio) plays in the risk assessment for the underwriting of an insurance application. The first step in the process, *Check and update standard reference data set X 's*, does not require any action, since the original cholesterol ratio readings are still valid inputs. The second step in the process, *Check and update standard reference data set Y 's*, requires two actions: (a) verify if the original SRDs still apply; (b) obtain new SRDs when needed.

Since the rule thresholds will be modified in this example, we need to perform a new underwriting assessment of the affected cases in the SRDs set. The best way to accomplish this reassessment is to compare the classifier's decisions (based on the old rule thresholds) with the new decisions (based on the new rules thresholds). All cases with a mismatch between the two decisions need to be isolated and re-evaluated by a new expert underwriting assessment. The third step in the process, *Modify FLE (update transfer function)*, requires three actions: (a) modify rule and thresholds corresponding to the new input; (b) define type of specialized processing (pre/post); (c) evaluate base-line performance of engine.

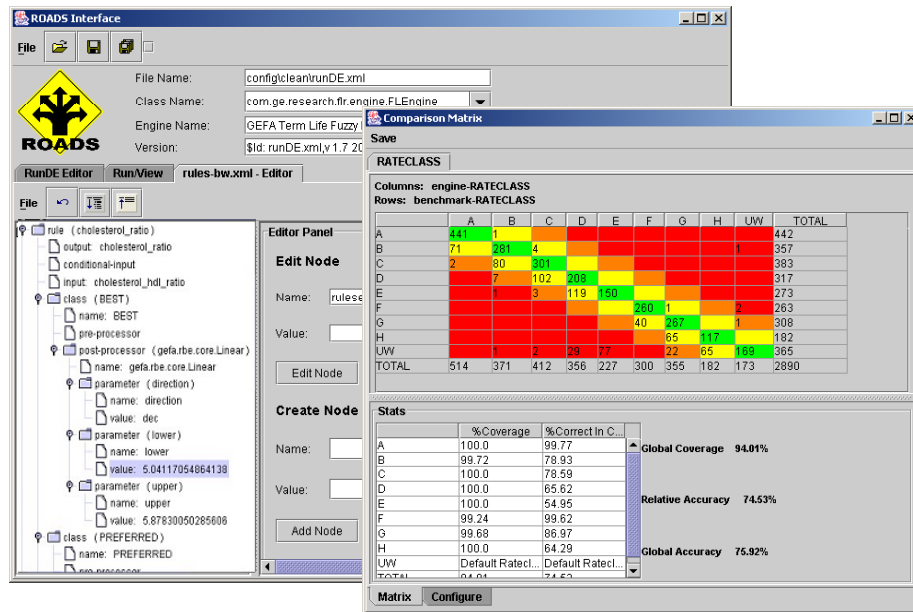


Figure 7. FLE maintenance and update tools

Using the maintenance tools illustrated in Figure 7, we can modify the parameters of a particular X , e.g. the cholesterol ratio rule, changing the thresholds of its lower and upper fuzzy parameters for each rate class. The type of pre/post processing is unchanged in this example. Once the new parameters have been saved, the maintenance tool computes a new confusion matrix M reflecting the effects of the new thresholds, as illustrated in the right portion of Figure 7. Having encoded our best estimates of cholesterol ratio thresholds obtained from the new guidelines, we need to fine tune them. This is addressed by the fourth step, *Perform optimization*. The modified SRDs, which reflects the results of applying the new guidelines to the original SRDs, is used by the EAs to fine tune the classifier's parameters, including the newly updated ones. The resulting confusion matrix M should show an improvement over the one obtained before the optimization. The last two steps in the process, *Approval* and *Workflow integration*, are related to the certification and integration of the new decision engine within its supporting workflow.

CONCLUSIONS

In this paper, we have emphasized the lifecycle perspective for developing automated decision technology, and the importance of leveraging structures and approaches such as Six Sigma in design as well as implementation. We have demonstrated how this can be achieved by describing the development of an automated decision system for insurance underwriting. In conclusion, we would like to highlight the key elements of the Six Sigma roadmap as they apply to the development and implementation of any automated decision system.

The lifecycle of an automated decision system has been defined in four phases: Build, Use and Monitor, Maintain and Update, and Retire. The rigor of DFSS applied during the Build phase will insure that the engine meets its customers' requirements at the time it is ready for use. During the time the engine is in use by the business, it must be monitored, maintained, and updated in order to maximize the value that the engine delivers between the Build and Retire phases, which in turn maximizes the return on investment for the business owners of the system.

During the Build phase, the decision system must be designed and constructed to meet the CTQs for the decision model, IT integration, and maintenance. Design trade-offs are always present no matter what the application of decision engine technology. In the Define phase of the project, enough insight into the requirements of a specific application must be gathered in order to be able to make the appropriate trade-off for that situation. It is critical to gather data to be used to create the decision model, and to verify that the CTQs are being met. Data are critical to the DFSS process for decision engines because the data are the measurement system for the model (which can be seen simply as a transfer function from X 's to Y 's—the X 's being the decision engine inputs and the Y 's being the decision engine outputs). There is no other way to test and validate a decision model than with data.

After the Build phase, the engine is in use by the business owners until it is retired. Since it is the 'Post-build' portion of the lifecycle that delivers value to the business owners, the engine deserves as much attention from the developers and business owners as it received during the Build phase. A serious challenge to the successful deployment of intelligent systems is their ability to remain valid and accurate over time, while compensating for drifts and accounting for contextual changes that might otherwise render their knowledge base stale or obsolete. By monitoring, maintaining, and updating the engine and the knowledge base, the post-build portion of its lifecycle can be maximized, along with the value it delivers to the business while in use. This requires a monitoring system that alerts process owners when maintenance actions are needed, along with a maintenance plan and tools to react quickly.

Future work

There are many potential extensions to the work presented in this paper. Some extensions could be achieved by applying this methodology to other application areas. We have demonstrated a Six Sigma process used throughout the lifecycle to date of a classifier for underwriting insurance applications. However, both classifiers (fuzzy-rule-based engines and case-based engines) and their common underlying process have a much broader applicability. They could be used in other financial applications, such as credit risk assessment, and in engineering applications, such as equipment diagnostics²².

Other potential extensions might involve changing the nature of the decision engine. Should the output variables require continuous rather than discrete values, we would require an underlying model to produce value estimations or predictions, rather than discrete classifications. However, the supporting Six Sigma process would still be applicable with appropriate modifications. In such a case, we would use a different model structure, the parameter encoding for the evolutionary algorithm would be dictated by the new model structure, the fitness function would represent some measure of global error rather than the cost of misclassification, etc., but the overall Six Sigma process with all its steps (DFSS + DMAIC) would still be valid.

Some of these steps could be extended. In particular, we could extend the evolutionary algorithms to generate *both model structure and parameters*. Model structure means determining the type of input variables required to make the decision and their relationship to intermediate and output variables. For example, in an artificial NN, the structure determines the network topology, which in turn defines the connectivity (functional dependency) among the input, intermediate, and output nodes. The parameters for such NN model would be the weights attached to each link²³. The purpose of this extension would be to simplify model maintenance. However, there might be competing CTQs (from legal or compliance regulations) that might prevent the use of such a variable structure. For instance, in the insurance application, the state insurance commissioner determines the type of inputs that the model can use. Hence the structure of the classifier is basically predefined. For other applications this legal constraint might not be an issue.

Finally, we could extend the optimization process to consider multiple objectives, e.g. multiple criteria that must be satisfied and optimized simultaneously, a requirement common to many design problems. In the case of the insurance application described in this paper, we used an evolutionary optimization based on a single-valued fitness function f . Such a function represented the overall misclassification cost for a specific trade-off between Coverage and Relative accuracy. Let us recall that function f was obtained by the element-wise multiplication of the confusion matrix M with the penalty matrix P , both $T \times T$ matrices. To enforce *coverage in conjunction with accuracy*, we needed to dissuade the classifier from not issuing decisions when it was appropriate to issue one. This was achieved by creating a non-zero penalty value for the cells $P(i, T)$, $i = 1, \dots, T - 1$.

These cells described all cases in which the engine would refuse to provide a rate class, while the SRDs would support a specific rate class for the case. By imposing these high penalty values, *we selected a specific trade-off between the coverage and relative accuracy*. In a separate experiment, we used multi-objective evolutionary algorithms (MOEA) to define a Pareto frontier in the space (Coverage, Relative accuracy). This frontier is composed of non-dominated points, defined by distinct vectors of model parameters. Each point corresponds to a different trade-off between the two criteria. Decision makers with different CTQs could use their preferences to select their final point from this Pareto frontier. Then, using the parameters associated to the selected point, they can instantiate the desired classifier²⁴.

REFERENCES

1. Casillas J, Cordón O, Herrera F, Magdalena L (eds.). *Interpretability Issues in Fuzzy Modeling, and Accuracy Improvements in Linguistic Fuzzy Modeling (Studies in Fuzziness and Soft Computing, vol. 128–129)*. Springer: Berlin, 2003.
2. Guillaume S. Designing fuzzy inference systems from data: An interpretability-oriented review. *IEEE Transactions on Fuzzy Systems* 2001; **9**(3):426–443.
3. Bonissone P, Chen Y-T, Goebel K, Khedkar P. Hybrid soft computing systems: Industrial and commercial applications. *Proceedings of the IEEE* 1999; **87**(9):1641–166.
4. Aggour K, Pavese M. ROADS: A reusable, optimizable architecture for decision systems. *Proceedings of the Software Engineering and Knowledge Engineering Conference (SEKE'03)*. Knowledge Systems Institute: Skokie, IL, 2003; 297–305.
5. Farley J, Crawford W, Flanagan D. *Enterprise JavaBeans*. O'Reilly: Sebastopol, CA, 2002.
6. Breiman L, Friedman JH, Olshen RA, Stone CJ. *Classification and Regression Trees*. Wadsworth and Brooks: Monterey, CA, 1985.
7. Michie D, Spiegelhater DJ, Taylor CC. *Machine Learning, Neural, and Statistical Classification*. Ellis Horwood: Englewood Cliffs, NJ, 1994.
8. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 1943; **5**:115–133.
9. Fiesler E, Beale R. *Handbook of Neural Computation*. Institute of Physics: Bristol, U.K. and Oxford University Press: New York, 1997.
10. Friedman JH. Multivariate adaptive regression splines. *Annals of Statistics* 1991; **19**:1–141.
11. Buchanan B, Shortliffe E. *Rule-based Expert Systems*. Addison-Wesley: Reading, MA, 1984.
12. Quinlan J. *C4.5: Programs for Machine Learning*. Morgan Kaufmann: San Francisco, CA, 1993.
13. Zadeh LA. Fuzzy sets, *Information and Control* 1965; **8**:338–353.
14. Ruspini EH, Bonissone PP, Pedycz W. *Handbook of Fuzzy Computation*. Institute of Physics: Bristol, U.K., 1998.
15. Holland JH. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence* (3rd edn). MIT Press: Cambridge, MA, 1994.
16. Back T, Fogel DB, Michalewicz Z. *Handbook of Evolutionary Computation*. Institute of Physics: Bristol, U.K. and Oxford University Press: New York, 1997.
17. Michalewicz Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer: New York, 1994.
18. Bonissone P, Subbu R, Aggour K. Evolutionary optimization of fuzzy decision systems for automated insurance underwriting. *Proceedings of the IEEE International Conference on Fuzzy Systems*, Honolulu, HI, 2002. IEEE Press: Piscataway, NJ, 2002.
19. Breyfogle F. *Implementing Six Sigma: Smarter Solutions Using Statistical Methods*. Wiley: New York, 2002.
20. Wu C-H, Lee S-J, Chou H-S. Dependency analysis for knowledge validation in rule-based expert systems. *Proceedings of the 10th Conference on Artificial Intelligence for Applications*. IEEE Press: Piscataway, NJ, 1994; 327–333.
21. Bonissone P. The life cycle of a fuzzy knowledge-based classifier. *Proceedings of the North American Fuzzy Information Processing Society (NAFIPS 2003)*, Chicago, IL, August 2003. IEEE Press: Piscataway, NJ, 2003.
22. Aggour K, Pavese M, Bonissone P, Cheetham W. SOFT-CBR: A self-optimizing fuzzy tool for case-based reasoning. *Proceedings of ICCBR 2003 (Lecture Notes in Artificial Intelligence, vol. 2689)*. Springer: Berlin, 2003; 5–19.
23. Vonk E, Jain LC, Johnson RP. *Automatic Generation of Neural Network Architecture Using Evolutionary Computation*. World Scientific: Singapore, 1997.
24. Guhde R. Multiobjective evolutionary algorithms: Targeting migrant election and selection schemes on an island model PMOEA. *MS Thesis*, Computer Science Department, RPI, Troy, NY, 2003.

Authors' biographies

Angie Patterson received her PhD in Statistics from Virginia Tech in 1997, after which she joined General Electric Global Research with five years prior experience as an Applied Statistician. At General Electric, she received certification as a Six Sigma Master Black Belt and has mentored the application of statistics and quality methodology to a wide range of Research and Development programs. More recently, she has led the development of risk management tools for life insurance products and her current research interest is probabilistic design for complex systems. She was the 2004 chair of the Quality and Productivity section of the American Statistical Association and has also been an Adjunct Professor in the Virginia Tech Department of Statistics.

Piero Bonissone has been a computer scientist at General Electric Global Research since 1979. He has carried out research in AI, expert systems, fuzzy sets, and Soft Computing, ranging from the control of turbo-shaft engines to the use of fuzzy logic in dishwashers, locomotives, and power supplies. He has developed case-based and fuzzy-neural systems to accurately estimate the value of residential properties when used as mortgage collaterals and to predict paper-web breakages in paper mills. He has led a large internal project that uses fuzzy rules to partially automate the underwriting process of life insurance applications. Recently he led a soft computing group in the development of prognostics of products' remaining life. He is an Adjunct Professor in the DSES and ECSE Departments at the RPI, Troy, NY. Since 1993 he has been the Editor-in-Chief of the *International Journal of Approximate Reasoning*. In 1993 he received the Coolidge Fellowship Award from GE CRD for overall technical accomplishments. He is also a Fellow of the American Association for Artificial Intelligence (AAAI) and of the Institute of Electrical and Electronics Engineers (IEEE). In 2002, he was the President of the IEEE Neural Networks Society. He has co-edited four books and published over 100 articles. He has received 33 patents (and has 25+ pending) from the U.S. Patent Office.

Marc Pavese received his PhD in Theoretical and Computational Physical Chemistry from the University of Pennsylvania in 1999. He then worked at GE's Global Research Center until 2004, focusing on computation and algorithms for financial services businesses. He currently works as a risk manager for Genworth Financial, a leading insurance holding company serving the lifestyle protection, retirement income, investment and mortgage needs of more than 15 million customers globally.