A FRAMEWORK FOR EVALUATING INDUCTIVE MODELS OF SOFTWARE DEVELOPMENT

Scott Serich
Assistant Professor
Management Science Department
George Washington University
Washington, DC, USA
(703) 288-3068
Fax: (703) 288-3078
E-Mail: serich@acm.org
http://gwis2.circ.gwu.edu/~serich

ABSTRACT

This study introduces a framework for evaluating decision models in organizations that conduct custom software development. The framework takes the form of a meta-model into which decision models can be embedded and assessed. In response to the turbulent, heterogeneous task environments facing software firms, the framework targets each model's self-adaptive or *inductive* features for analysis. The evaluation mechanism is comprised of homomorphisms from abstract algebra and the transition function, observability and controllability features of control systems theory. The meta-model is tested on three candidates, two static models and a dynamic model based on Simon's behavioral model of rational choice. It correctly distinguishes the former models as having weak induction features and the latter as being strong on this aspect.

INTRODUCTION

Consider the following scenario. An executive from a software development firm approaches the management information systems community in search of a formal decision-making model to help the firm identify and generate optimal performance across its portfolio of projects. She is seeking a single, firm-wide model that the entire management team can adopt and utilize consistently in developing its strategies. Her company differentiates itself by employing highly skilled software engineers capable of delivering tailored, state-of-the-art products to meet the specific needs of customers.

In response, we suggest that she start with the static optimization approach from her MBA microeconomics course. Her firm should first identify its production function, to characterize costs, and its target market niche. It should then generate forecasts for the production factors and for the market, and choose an output level that maximizes the revenue-cost differential. Cyert and March [3, pp. 22-25] summarize this approach for the case of multi-product firms. If the market has a dominant player, the firm should also consider a game-theoretic strategy to maximize minimum profit level across various competitor actions.

The executive responds by confirming that we have properly understood the firm's goal, but she insists that it differentiates itself on an ability to create *custom* software for whatever needs might arise. The firm cannot identify which particular market niche it will occupy, what the corresponding demand will be or who the competitors are. In fact, many of the competitors are likely to be in-house information systems departments within customer companies. She further states that continuing technology improvements are exacerbating the challenge. Increased capacity in the hardware and networks on which the software executes has enabled a concomitant growth in the number and complexity of possible system features. A static model will be of little use. What recommendation should we offer now?

Because the firm performs custom software development, its *buyers* will reserve ultimate authority over product content. But buyer desires may change over time, and perhaps even several times over the life of a single development project. The firm is exposed to the risk that the final desired function and performance might not be delivered. Thus the executive's claim that her firm cannot determine its market niche *ex ante* appears reasonable.

In fact, the dilemma of not being able to accurately pre-specify desired software features may be a pervasive one. As much as 80% of software projects are judged to be at risk because of "creeping user requirements" [9]. As a side effect, a vendor's ability to formulate and implement a profit-maximization strategy is threatened because it is based on a moving target.

Our response to the executive, then, might be to enhance the static optimization models with features enabling them to be modified by the firm as it pursues a moving target. For example, an approach grounded in dynamic programming might be more suitable. An even more comprehensive response would be to develop an *inductive* approach whereby the model itself can adapt to pursue a moving target.

This study lays a foundation for the latter response by introducing a framework for emulating and formally comparing inductive decision models of custom software development. Its inspiration was taken from Holland's work in artificial intelligence to test the *genetic algorithm* and *learning classifier system* as models of adaptation [5] [6]. This research used an approach called a *quasi-homomorphism*, introduced by Holland [8], which allowed the co-existence of multiple control policies during stochastic transitions of inductive, rule-based agents. The *meta-model* framework in the current study is a deterministic variant emphasizing a feature allowing the decision model's output signals to influence both the environment and its own subsequent performance. Otherwise the two frameworks are very similar.

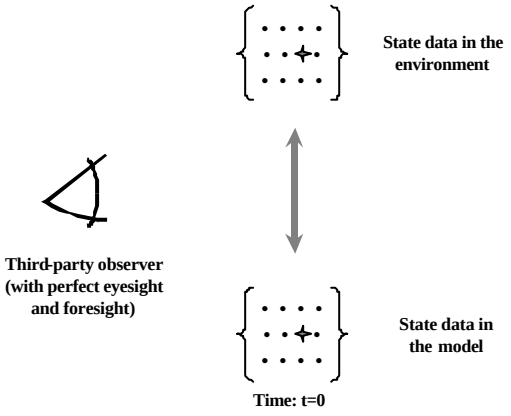
In developing this meta-model, the firm's presumed objective could have been to maximize *profitability*, accumulated time-discounted profit over an extended time horizon. Recognizing this as a potentially intractable goal for custom software firms led to a less-ambitious objective: to mitigate the risk introduced by changes in desired function and performance after commencement of software development projects. This will be referred to as the risk of *late specification changes*. The term *risk* in this case refers to any event that might threaten the firm's profitability, even though a particular profit level is not being assumed.

META-MODEL FRAMEWORK

The first requirement to be addressed will be to establish the decision-making boundary by separating model and environment and locate both internal and external firm information in the latter. Figure 1 provides a graphical depiction of this separation, showing the *state data* for both roles. The term "data" is used in the broad sense, referring to both the model's store of information at a particular time as well as any procedures for manipulating that store. The arrow between the two roles reflects the fact that signals can flow across the boundary from the model to the environment and vice-versa. The model performance data or *payoff* can be included in the signals being sent from the environment. Ideally this data would include indicators of future profitability. In the case of custom software development, however, the cost of obtaining such information was considered prohibitively high.

FIGURE 1

Separation Of Model From Environment



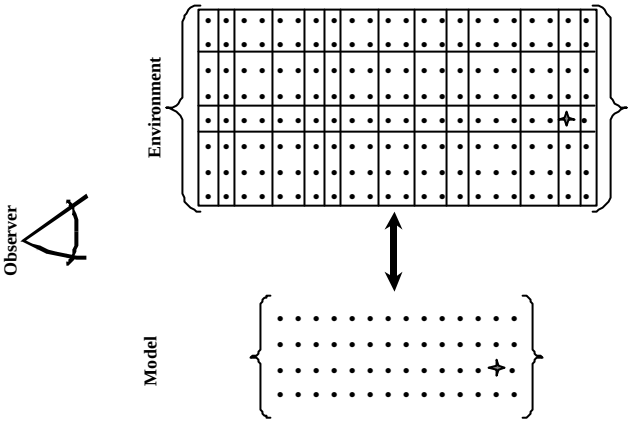
A third role of outside observer has been added. In practice, an observer such a consultant may not have any better information than the firm itself. For purposes of the study, however, the observer was assumed to have full knowledge at all times, and represents the portal through which the operation of decision models can be observed. In models that utilize *supervised learning* rather than inductive learning, this role would be capable of intervening on behalf of the firm to assist with decision-making.

The example in Figure 1 shows an environment consisting of twelve possible states, represented within brackets, with stars identifying current states. The model in this case enjoys the luxury of having estimated its environment with 100% accuracy, though it may not necessarily be aware of this match. A single time period is depicted, as might be appropriate for a decision model whose sole purpose was simply to estimate the current environment.

Note the subtle difference between the number of distinct states, in this case twelve, and the number of different variables comprising each state. The latter count is the dimension of the vector of component items, which could include past performance data, the status of ongoing projects, latest specifications, *et cetera*. Heterogeneous task environments are accommodated by not placing any bounds on number or form of these state representations, and by allowing the finite state machine information processing activities described earlier. Other descriptions for such environments are as having high *complexity* and large *algorithmic information content* [2], the smallest amount of data required to simulate them on computers. Figure 2 shows a model that attempts to estimate only a subset of information from a heterogeneous environment. This could be accomplished either by filtering out selected variables or by forming aggregates from combinations of multiple environmental readings (as shown in Figure 2).

FIGURE 2

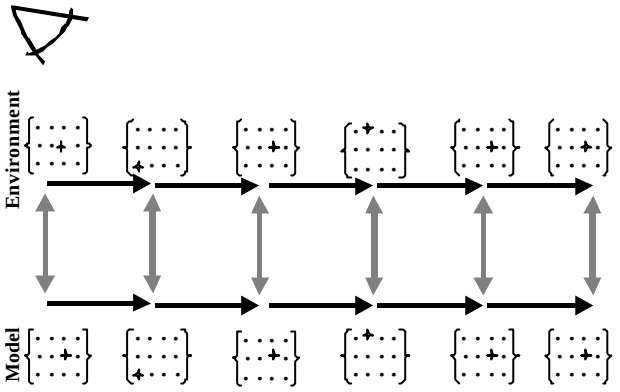
Modeling a Heterogeneous Environment



To enable induction in turbulent environments, the meta-model would have to allow execution over multiple time periods, with inspection of intermediate results. Thus intermediate representations of both model and environment would be required. A tool that meets this requirement is the *transition function* from control systems theory. Holland [6] provides examples applying the genetic algorithm and learning classifier system models to the control domain. It is distinguished by its ability to model dynamic phenomena and was thus deemed a promising choice as a referent for modeling turbulent, heterogeneous task environments. The transition function carries state variables from period to period in discrete, equally spaced intervals. As the meta-model executes, the environment and decision model each trace out a *trajectory* of states, in full view of the observer, as shown in Figure 3. In referring to the value of any particular variable from this point forward, a time index will be required.

FIGURE 3

Transition Function



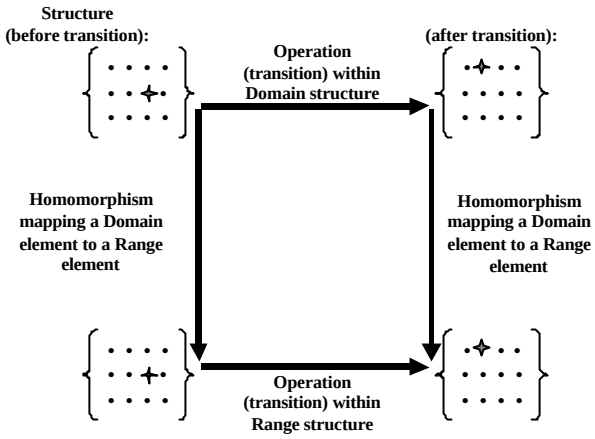
Unlike the quasi-homomorphisms used by Holland [8], the meta-model uses a deterministic transition function. Whether a decision model can discover this function and use it to help forecast future states represents a different challenge. Determining the sequence of decisions required to induce a particular transition function poses another challenge. In trying to overcome these challenges, the better a model can mimic its environment, *ceteris paribus*, the better its overall performance should be. The tool that formalizes the faithfulness of this mimicry is the *homomorphism*.

Homomorphisms are functions that arise in *group theory* in the study of *structures*. Certain transition functions are closed when applied to these structures, offering stability in repeated transitions within dynamic models. An even more appealing feature is that we may be able to define a function from a *domain* structure to a *range* structure so as to preserve the results of all operations within the first structure. So when the transition function is applied to any element of the domain structure, followed by application of this second function, the result would be same as applying the second function first, and then invoking the transition function. When such a second function is discovered, it is called a homomorphism.

Figure 4 illustrates the defining features of homomorphisms. The horizontal arrows correspond to the transition function from one structure element to another. The vertical arrows represent the potential homomorphism from each element of the top structure to its corresponding element in the bottom. The homomorphism obtains if the diagram *commutes*; that is, if the element picked out by performing an operation within the top structure followed by an application of the homomorphism is identical to the element picked out by applying the homomorphism first and then the operation in the bottom structure. Figure 4 depicts most extreme case where the model has sufficient resources to fully estimate all twelve states in its environment at both points in time. A typical decision environment is likely to be vastly more complex than can be described in only twelve states; and many models may choose to represent a proper subset of that environment. The homomorphism, in this case an *isomorphism*, obtains because the model perfectly mimics the twelve-state environment in its state data at each step.

FIGURE 4

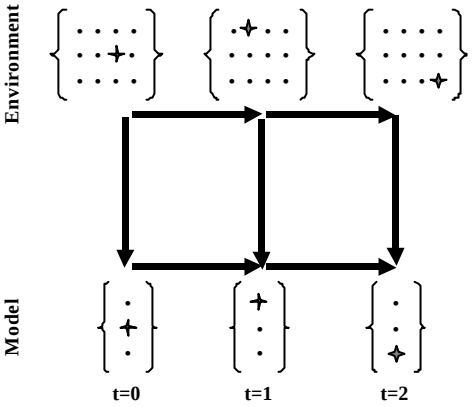
Illustration of a Homomorphism



A more representative example is depicted in Figure 5, where the model's state space is smaller than the environment's, but where faithful transitions are still maintained throughout the trajectory. In this case, the homomorphism is a surjection that maps each environmental state to its corresponding row in the model state space for three time periods. The homo morphism obtains because no matter which path is used to get from the environment at time t=0 to the model at a subsequent time, the same state is always chosen.

FIGURE 5

Homomorphism with Smaller Range than Domain



Homomorphisms are useful in the current context because in spite of the separation of model and environment, discovery of such a function would indicate a faithful correspondence between their trajectories. This would mitigate late specification changes by helping the model better understand the true states of the environment along its trajectory. A poorly performing model might not engender a homomorphism, due either to poor estimates of its environment or failure to accurately predict the impact of its own decisions on that environment. Models enact decisions at each step by sending a multivariate signal to the environment and simultaneously receiving a

multivariate signal in return. The closer the firm is to inducing a homomorphism, the closer these return signals should be to those anticipated by the firm.

Measuring Model Performance

The indicators used to evaluate each model's relative performance were also derived from control theory. Control systems in engineering contexts tend to focus on stability as the primary performance criterion [1]. The focus for the meta-model, however, was on the features contributing to induction toward homomorphisms.

Observability: The first control system feature, *observability* [16], describes whether the model gains sufficient information to estimate and forecast the environment's state. For purposes of the current study, this criterion was not operationalized into a real-valued measure. Instead, it was sufficient to create a weaker binary indicator of whether a decision model identifies future states that could help mitigate the risk of late specification changes. In more mundane terms, this indicator reflects whether the model is attempting to forecast a target at which the firm should aim. Referring to Figure 5, the decision model would be trying to project which state or states it expected the environment to attain at time t=2.

A decision model will have little hope of inducing a homomorphism without some indication of whether its control policies are driving the system toward the desired target. When no target is identified up front, the model's performance cannot be shown to be better than a randomly chosen strategy. This doesn't mean that the target must be specified to the level of detail of identifying an individual state. In fact, it may be too costly for the model to create a target of such high specificity. It does mean, however, that a target beyond just any randomly chosen future state must be called out. The optimal future state for a learning classifier system [5] would provide the agent with the largest possible payoff. In the current context, this would be the state that results in minimum specification risk.

Controllability: The second control system performance indicator is *controllability* [16], which indicates whether the model can drive the environment into a desired state in a finite time. This measure was also operationalized as a binary variable, this one indicating whether the model identified an initial control policy. It reflects whether the model starts with a strategy to guide the firm toward a particular target, ideally the one identified under the observability criterion. In Figure 5, this would appear as a pre-defined decision table that attempts to induce a desired environmental states when time t=2 is reached. The homomorphism would obtain if it were, in fact, to induce the desired state and recognize it as such.

The term "induce" is used here rather than "cause" since the model can exert only partial control over its environment. This reveals the following duality: as the decision model is trying to inductively learn how to better mimic its environment, it is also trying to induce that environment to move closer to a desired future state. In other words, a desirable outcome of the operation of an inductive decision model is for the environment to also undergo "induction" toward the future state most advantageous to the firm. Holland [7] identifies this moving-target phenomenon as *perpetual novelty*, but focuses on the genetic algorithm to support agent adaptation rather than on finding ways to influence the environment. Control system engineering has tended in the past to place more focus on modifying the environment, but with intervention by the outside observer rather than via model induction.

Model Induction: The final indicator directly measures the sought-after self-modification characteristics. When present, this feature is often referred to as *adaptation* [6] or *intelligent control* [4]. For purposes of the current study, the measure is a binary indication of whether a model modifies its control policy based on how well it performs on the first two criteria. This would be manifested in Figure 5 if the model were to detect that the firm was off course at time t=1 and made a correction to its original strategy. Real-time adjustment based on feedback from the environment can give a decision model "second chances" in attempting to attain the desired homomorphism.

For the coarse-grained analysis in the current study, an affirmative response on all three indicators would represent a sufficient condition for optimal model performance. The more negative responses, the worse the model's performance.

The meta-model will now undergo a test to determine whether it can identify the weaknesses in the static models identified in the scenario. Failure to detect the known shortcomings of these non-inductive models would cast doubt on its utility in assessing more sophisticated approaches.

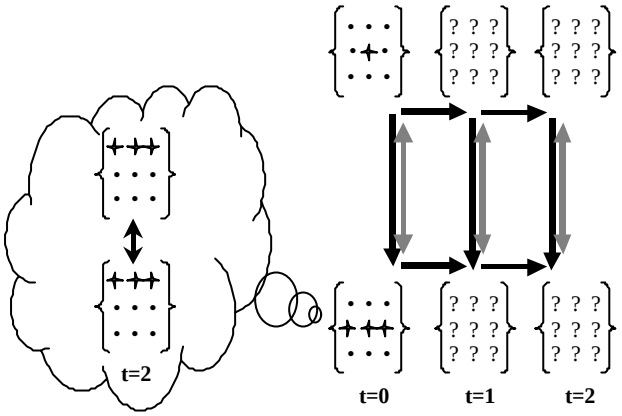
TESTING THE META-MODEL

Rejection of the Static Model

Observability: The static optimization model calls for forecasts of both supply and demand and a production decision that maximizes the difference. This translates into multiple steps for the custom software firm. First, the supply side will be considered sufficiently stable so as to not provide any differentiation among the models. It will be ignored from this point forward. On the demand side, the model would choose a time horizon and forecast the future state of the environmental variable identifying the software firm's customer set. This selects a subset of the future state space, but not necessarily a particular point since the values of many other variables have yet to be fixed. Figure 6 depicts this by displaying multiple anticipated states (stars in the first row inside the cloud) at the projected terminal time t=2. Thus the estimate is expected to be only partially correct in identifying the correct row, but would become fully correct if and when the proper column had been estimated. The static model would, of course, have neither an identified forecast nor a response at the intermediate time t=1. The double-ended arrows represent the flow of signals across the boundary separating the environment and the model. The other arrows represent the desirable possibility that a homomorphism has been induced between the model and the environment.

FIGURE 6

A Moderately Specific Forecast in the Static Models



The model would then forecast the function and performance for the final product these customers are expected to purchase. While the actual mechanism for constructing such a forecast was not identified, it will be assumed that the static model possesses a valid forecasting technique. Affording it a higher score than has explicitly been earned is permissible since the goal in testing the meta-model here is to show that the static model performs poorly on the three criteria *in toto*. This move simply shifts the burden of showing the model's weakness to the other two criteria. The claim that it can accurately identify correct future states to help mitigate the risk of late specification changes is conceded, and it is granted a positive score on the first criterion.

Controllability: Though it may perform sufficiently well in identifying the target, the static model does little to help custom software firms establish the correct control policy to drive toward this target. The prescription advises the firm at the outset of the production cycle to select a profit-maximizing output level. The high degree of asset specificity for custom software renders the notion of a production level meaningless. The firm will produce one system per contract, or perhaps several, but not nearly enough to make the notion of a profit-maximizing output decision useful. The model scores negatively on this criterion.

Model Induction: A static model utilizes no dynamic features, and thus no induction, causing it to score poorly on the third criterion.

The overall meta-model assessment of the static model is that it performs poorly on two out of the three criteria, placing it in an exposed position vis-à-vis competing models. The assessment successfully replicates the negative response from the scenario.

Rejection of the Game Theoretic Variation

Consider the game theoretic variation of the static model, also rejected in the scenario. In addition to its own demand forecast, the game theoretic model requires forecasts of what future states competitor firms will reach. Thus the specificity of the overall forecast of the environment must be even higher for this model. Assuming that such forecasting mechanism is available, this model would score higher on the first criterion. It provides no improvement on the remaining two criteria, on which the static microeconomic model had already scored poorly. A maximin strategy is a choice of whatever production level maximizes the firm's profit in light of knowledge that a competitor can prevent it from reaching the global optimum. Inductive adaptation and control are not available to players in static games. The overall assessment of the static game theoretic variation is that it performs poorly on two out of the three criteria, and is thus also exposed to competition in these areas.

The meta-model successfully replicates the negative reactions from the scenario.

THE BEHAVIORAL MODEL

The more accurately a decision model can describe a future target state, *ceteris paribus*, the more likely it will be to lead the firm to the optimal outcome. Similarly, the higher the specificity of this forecast, the more likely an optimal outcome will obtain. Unfortunately, precise, accurate forecasts come only at a cost. Otherwise, the executive from the scenario would not have been seeking help in the first place.

Simon's Behavioral Model of Rational Choice [12] formalizes this phenomenon. The resources that most often tend to constrain this forecasting endeavor are those of management *attention* [13] and *time* [15, p.22]. In other words, the more time and attention an organization spends searching for an optimal future state, the less profitable this state will be due to the dissipation of scarce management resources. This lack of sufficient resource to discover a specific, optimal target is typically labeled *bounded rationality*.

Additional resource dissipation can occur as the organization's production resources stand idle, waiting for a decision to be made. Ultimately, the organization may engage in *satisficing*, reducing aspirations and settling for a sub-optimal decision in order to prevent further dissipation. Any *ex post* rationality that might be extracted from the decision process and captured as a decision rule would be called a *heuristic*.

*Observability***:** The behavioral model's first primitive [12] is a set **S** of possible future states of affairs, including the payoff associated with each state. In order to generate this payoff function, the model must have access to the same forecast information as was required in the static model. Thus the behavioral model will score identically on the meta-model's first evaluation criterion. In fact, Simon used variants of the static models presented earlier as reference points for launching the behavioral model.

Controllability: The second primitive is a set of alternatives, **A**, enumerating all possible decisions that could be made. In contrast to static models, the behavioral model divides **A** into alternatives the firm perceives, which Simon labeled **Å**, and those that remain unknown. So while **A** could include many decision variables beyond just production level, the model does not assume full knowledge of those variables. Thus it scores better than the static models by permitting access to a larger set of decision variables to help aim custom software firms toward correct targets. This feature's value is tempered, however, by the fact that the decision-maker may not be aware of these variables and is

provided with no tools to help utilize them. Thus the behavioral model earns a reserved positive score on the second criterion.

Model Induction: Because the behavioral model presumes less up-front knowledge, it must and does provide the decision maker with the opportunity to adapt to data gained later. Acknowledging that adaptation entails added cost, the behavioral model allows for *satisficing* to a sub-optimal decision. These features are well suited for mitigating the risk of late specification changes. A software firm that is confident in its initial forecasts of product specifications and costs can obviate the need for induction altogether. If it lacks such confidence, however, it can execute the behavioral model until either the desired confidence level is attained or the firm grows fatigued from resource dissipation and settles for a sub-optimal outcome.

During model operation, the firm may wish to keep a repository of lessons learned to reduce the cost of induction in the future. Unfortunately, the behavioral model itself provided little guidance for cataloguing and re-utilizing these heuristics in real time. Simon's later work in artificial intelligence [10] indicates that a *production system* might be the appropriate approach to accumulating decision rules over time, though [5] raises *brittleness* as a potential stumbling block for such approaches. Another hurdle is that production systems may not scale gracefully, resulting in an overload of distracting information [14, pp.143-144] or increased energy dissipation [11]. These questions are left for future research.

The net result is that the behavioral model earns a reserved positive score on the third criterion. Overall, based on its equivalent performance on the first criterion and stronger performance on the second and third criteria, it scores higher than both static models.

Note the following additional challenge that was not brought out in the original scenario. The claim that accurate forecasts are more likely to lead organizations to better risk mitigation, and better performance in general, was made earlier. A similar case was made for the value of more precise specifications. The more precisely a model's target state has been defined, however, the more precise must be its aim in reaching that state. Otherwise, the risk of late specification changes increases: software engineering artifacts may have been developed under the assumption of precise knowledge of an incorrect target state. If development under such circumstances were to reach the target, it would be guaranteed to be striking an erroneous target.

DISCUSSION AND FUTURE RESEARCH

What recommendation should be offered to the custom software firm from the original scenario? Based on the results of applying the meta-model framework, Simon's model is better suited to adapt to the firm's turbulent, heterogeneous task environment. The single-step approaches of the static models run higher risks of either forecasting an incorrect target or failing to properly aim for this target. An inductive approach dilutes the risk of a single, monolithic decision across multiple risks of multiple decisions, making risk diversification easier.

A vast amount of work remains to characterize the full costs of utilizing inductive modeling approaches. Perhaps the rework costs associated with initial missteps would be too high to justify an inductive approach, leading a firm to fall back on one of the static models. A software firm might be better off settling for a final product that doesn't fully satisfy desired function and performance if the associated pursuit cost is judged to be excessive.

A promising next step in pursuing such questions would be to replace the three binary performance indicators with commensurate, real-valued variables that include model implementation and rework costs. These could ultimately lead to an absolute performance measure and a much more reliable approach to ranking decision models.

Much work also remains to flesh out the details of operating an inductive model in practice. Based solely on the potential risks uncovered in the current study, it might be prudent for a software firm to first inventory and track all specifications across its entire portfolio of development projects. This data would serve as the initial raw materials for a centralized decision-making authority serving as the model's intelligence, charting initial courses of action, monitoring feedback, and revising course as needed.

Another important function for this authority would be to help project teams resist the temptation to prematurely commit to high-specificity product specifications. Such commitments increase the risk of rework costs as the firm gains more accurate knowledge about correct future target states. This does not imply that the team, including the customer, could not offer speculations about what it estimates the correct final state to be. It does imply that such speculations must be recognized for their potential inaccuracy and managed accordingly, including measures to charge customers who force the commencement of projects before the risk of late specification changes has been mitigated. Similarly, a central authority could monitor and perhaps even enforce other risk mitigation policies such as preventing the use of labor overtime early in project life cycles. This would help preserve the energy level of human engineering resources until later in the cycle, after more accurate specifications had been discovered.

REFERENCES

[1] Bishop, Robert H. and Dorf, Richard C. The Routh Hurwitz Stability Criterion. *The Control Handbook*, ed. William S. Levine, 131-135, CRC Press, 1996.

[2] Chaitin, Gregory. Algorithmic Information Theory. *IBM Journal of Research and Development* 1977, 21 (4) 350-359. IBM Watson Research Center. Yorktown Heights, NY.

[3] Cyert, Richard M. and March, James G. *A Behavioral Theory of the Firm*. Prentice Hall, 1963.

[4] Gupta, Madan M. and Sinha, Naresh K. *Intelligent Control Systems: Theory and Applications*. IEEE Press, 1996.

[5] Holland, John H. Escaping Brittleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. *Machine Learning: An Artificial Intelligence Approach* Vol. 2, ed. Michalski, R. S. Morgan Kaufmann, 1986.

[6] Holland, J. H. *Adaptation in Natural and Artificial Systems*, MIT Press edition. MIT Press, 1992.

[7] Holland, J. H. *Emergence: From Chaos to Order*. Addison Wesley, 1998.

[8] Holland, J. H., Holyoak, K. J., Nisbett, R. E., and Thagard P. R. *Induction: Processes of Inference, Learning and Discovery*, Paperback Edition, MIT Press, 1989.
[9] Jones, Capers. *Assessment and Control of Software Risks.* Prentice-Hall, 1994.
[10] Newell, Allen and Simon, Herbert A. *Human Problem Solving*. Prentice-Hall, 1972.
[11] Serich, S. T. Zipf's Law as a Necessary Condition for Mitigating the Scaling Problem in Rule-Based Agents. PhD. Dissertation, University of Michigan, Ann Arbor, Michigan, 1999.
[12] Simon, Herbert A. A Behavioral Model of Rational Choice. *Quarterly Journal of Economics* 1955, 69: 99-118.
[13] Simon, H. A. Designing Organizations for an Information-Rich World. *Computers, Communications, and the Public Interest*, ed. Greenburger, M. 38-52. The Johns Hopkins Press, 1971.
[14] Simon, H. A. *The Sciences of the Artificial*. The MIT Press, 1996.
[15] Simon, H. A. *Administrative Behavior*, 4th ed. The Free Press, 1997.
[16] Wolovich, W. A. Controllability and Observability. *The Control Handbook*, ed. William S. Levine, 121, CRC Press, 1996.