HICSS (Hawaiian International Conference on Systems Science, Jan, 2006, Hawaii
http://menzies.us/pdf/06hicss.pdf

Learning IV&V Strategies

Marcus S. Fisher
NASA / GSFC / IV&V Facility
Marcus.S.Fisher@nasa.gov

Tim Menzies
Computer Science / Portland State University
tim@timmenzies.net

Abstract

Modern business practices are complex. Consider, for example, NASA's software IV&V (independent verification and validation) team that monitors a diverse range of complex software written by a wide range of contractors from around the world. In an effort to better understand the core business of IV&V, the authors recently conducted DELPHI sessions with experienced IV&V analysts to build a model reflecting their understanding of what level of IV&V is appropriate for different projects. The resulting model, while short, contains subtle interactions that are not immediately apparent.

To understand those interactions, we conducted Monte Carlo studies to grow data sets from the model. These data sets where summarized using TAR3 (a minimal contrast set learner) to discover (a) the core business decisions that decide what level of IV&V is appropriate; and (b) whether or not specializations of the problem domain can lead to more simple and robust models.

1. Introduction

Here's how to start a debate: in some public forum claim that software process X or software tool Y increases or decreases overall system quality. If you want the debate to last a long time, you might also care to claim that some combination of processes and tools are better/worse for software quality than some current practice in a current project.

We are engaged in such a lengthy debate and want to optimize that dialogue. By "optimize" we mean that the debate terminates earlier and terminates on better decisions. Our domain is software IV&V policy. NASA's Independent Verification and Validation (IV&V) Facility in Fairmont, West Virginia is responsible for verifying that software developed or acquired to support NASA missions complies with the stated requirements. Additionally, the Facility validates that the software is suitable for its intended use. In short, the Facility ensures that the software is being developed properly, and that the right software is being developed or acquired.

A recent change in NASA's funding policy for IV&V has led to much debate about appropriate software processes and tools. Previously, IV&V had to market itself to individual projects. In that model, IV&V only occurred on projects where funds could be reallocated from other development activities. That funding model has now changed. NASA headquarters now maintains a funding pool for IV&V that is separate to project funds. The existence of this funding pool means that managers of NASA's software projects no longer have to make hard decisions about cutting funds from one part of a project in order to fund IV&V. Instead, projects that need IV&V can draw on additional funding from the funding pool at headquarters. However, the new model has its administrative challenges. Before, IV&V requirements and methods could be debated project-by-project amongst individual managers. Now, IV&V's methods are audited at a national level.

These additional auditing requirements have resulted in demands for greater precision and detail in defining IV&V processes and tools. In response to this increased domain for process details, IV&V analysts at Fairmont have conducted extensive DELPHI sessions to model that group's consensus on what comprises good and bad IV&V practices. The result is the IV&V Facility *Software Integrity Level Assessment Proces*s (SILAP) [3] model.

This paper uses data mining methods to analyze SILAP. Using data mining methods, we find we can convert intricate domain models into succinct policy statements. Our analysis method is in two parts:

1. *Stochastic simulations of the model*. Input values are picked from a profile representing common types of NASA projects. Unlike standard Monte Carlo simulations, we do not require detailed profile knowledge. In fact, for one of the runs

shown here, all the variables were picked totally at random (i.e. from a completely flat profile).

2. *Data mining of the output*. Stochastic simulation can generate voluminous output. Data miners can summarize that output.

2. Digressions

Before beginning, we pause for the following digression. An opponent of our approach might argue that we will learn very little from a summary log of randomly selected behaviors of a model. One analyst put this most succinctly when she said "won't you just learn the original model?"

In reply we note that, in all our experience with our approach, that *the summary is very different* to the original models. First, the learned model is often much smaller that the original model. Real-world models often contain irrelevant, redundant attributes, or attributes that have been tuned from noisy data (i.e. data containing spurious signals not associated with variations in the domain). Our data miners quickly strip away such noisy irrelevant redundancies.

Second, our data miners make explicit relationships that are implicit or hard to find in the original models. Knowledge within models can be hard to assess. It may be expressed verbosely or hidden within redundant or useless parts of the model.

Third, using some domain model may be hard when that model is very large and slow to execute. In contrast, our learned models may be a far simpler and succinct record of the important knowledge. Since they are smaller, our learned models can execute very quickly.

3. Minimizing a Model

The central thesis of this paper is that data mining can learn succinct theories of complex business models. To put that claim another way, we are saying that the essential details of a model are quite succinct.

We have argued for this claim for some years. In many models, influences are linked in "chains" (strings of preconditions leading to effects that are preconditions for other effects). For such "chained models", a negation of any part of the chain can negate the whole the chain.

Models with these chains may still appear complex because human short-term memory is too small to process all the current chains in software and automatic tool support is required. Since the early 1990s, we have been exploring a*bductive inference* to fully exercise the space of options within a program to find the key issues.

Abduction is the logic of argument: pre-conditions are inferred for all goals and those pre-conditions become the assumptions that the whole argument space must share [7]. Often, mutually incompatible assumptions can be inferred and the space of arguments must be resolved into alternate *worlds of belief*- each world comprises a maximal consistent set of beliefs. In the literature, these worlds are called many things including extensions [13], scenarios [12], or the environments of the a multiple worlds reeasoner like an assumption-based truth maintenance system (ATMS) [4].

Worlds contain assumptions, and assumptions form chains of argument. The assumptions the start of a chain are the *core assumptions* since they set the rest of the assumptions (in the language of the ATMS, these are called the *minimal environments*). Refuting *any* assumption in the chain refutes the whole chain.

Previously [9], we have shown mathematically that on average, the core assumptions in any model are few in number. We call this the funnel assumption; i.e. most models channel their inference through a small funnel comprising the core assumptions. In models where the funnel assumption holds, then the complexity of large models reduces to just the complexity of the variables in the funnel. When the funnel is small, this means that large complex models can be controlled via the appropriate settings to the funnel variables.

There is much evidence for these theoretical results. A surprising and repeated result is that, in the usual case, a very small number of variables control the rest of the models [9]. Other evidence comes from the theories of Herbert Simon. Simon rebelled against traditional models of human decision where decisions reflect optimizations to some goal function. His preferred model [14] for human rationality was *satisficing*, not optimizing[1]. The

[1] Satisficing is a behavior which attempts to achieve at least some minimum level of a particular variable, but which does not strive to achieve its maximum possible value.

distinguishing feature of a satisficing decision is that is sub-optimum. Somehow, humans following such a sub-optimum strategy still manage to make effective decisions about something as complex as the world. Our explanation for the surprising success of such a non-optimal strategy is the funnel assumptions. That is, making a few decisions about just the most important variables (i.e. those in the core) suffice for producing effective control strategies.

Lest we oversell our case, we hasten to add that a premise to the above optimism is that the core assumptions can be found. Traditionally, this has *not* been the case since complete abductive inference can be impractically slow. For example, the ATMS's runtimes were exponential on problem size [2].

However, recent results suggest that an alternative approach may be useful. If a model contains narrow funnels then, by definition, the funnel variables will be used very frequently. Consider the case of a model with narrow funnels *and* some oracle scored model output as (say) *good* and *bad*. In this model, certain ranges of the funnel variables will occur at very different frequencies in *good* than *bad*. The TAR3 *treatment learners* tested if using those ranges with very different frequencies was enough for building minimal models.

TAR3's rule generator favors the *smallest* rules that *most* change the output of a simulation [10] (in the language of machine learning, treatment learning combines *minimal contrast set* learning [1] with a *feature subset selection* mechanism) [6]. TAR3 orders variable ranges based on their frequency in *good* divided by their frequency in *bad*[2]. It then builds rules by randomly combining attribute range (here, "random" means that it favors attribute ranges with a high good/bad rating).

This approach can be shown to simplify the task of understanding complex models. When benchmarked against standard learners, we find that TAR3 often produces smaller theories. Those smaller theories can perform just as well as those found by a more complete search. For example, in comparisons of our data miners vs. more complete search algorithms such as simulated annealing [11] and genetic algorithms [5], Feather and Menzies found that their data miners heuristically generated alert points yielding solutions very close to the solutions generated by more complete methods.

Figure 1 shows an example of TAR3 simplifying the analysis of a complete model. In that example, experts on satellite design at NASA's Jet Propulsion Laboratory use this method to control their discussions. Given a requirements model, a simulator samples the space of design options to find the least number of critical decisions that most influence the design. The impact on the design debates can be quite dramatic. For example, from a design model with 2^{99} options, 30 key decisions could be found that dominated and rendered irrelevant the other 69 decisions. Applying those 30 key decisions, the benefits and additional costs associated with a particular satellite design changed dramatically. Figure 1 illustrates the results. Note that the average cost has greatly reduced while the average benefit has increased [5]. Note further, that our JPL experts can now argue faster since our tools have removed at least 69/99 of the disputes.

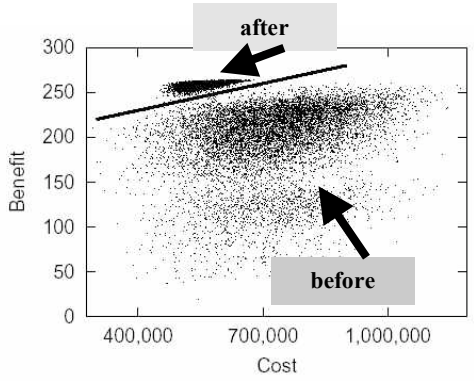

Fig 1: *Each dot is one project plan for JPL satellite design; i.e. one* possible setting to nearly 100 design possibilities. The x axis = cost shows the sum of the cost of selected design decisions (lower is better). The y-axis shows coverage of requirement, based on different design decisions (more is better)

The rest of this paper applies the TAR3 technology to the IV&V modeling problem.

4. The SILAP Model

The SILAP model has two major attributes, Consequence and Error Potential. The combination of these attributes, whose numerical ranges are between 1 and 5, is used to characterize the software development project under assessment and define the tasks that IV&V shall perform and the software modules that shall be within IV&V's scope. Each

[2] TAR3 is not restricted to binary classes. In the N-ary class case, TAR3 computes the weighted sum of a range frequency in class X times the utility of X.

attribute has a set of criteria that are used to further define the Consequence and Error Potential.

Consequence is composed of the following criteria;
- Human Safety,
- Asset Safety, and
- Performance.

Each criterion has an associated weight and can take on values from 1 to 5. Once the weights are applied to these criteria, their scores are summed and represent the Consequence score.

Error Potential has more criteria than Consequence and is further broken down into 3 sub-categories:
- Software Development,
- Software Process, and
- Software Characteristics.

The Software Development sub-category is comprised of the following criteria:
- Developer's Experience, and
- Development Organization's Communication Structure.

The Software Process sub-category is comprised of the following criteria:
- Developer's use of standards,
- Developer's use of configuration management,
- Developer's CMM level,
- Developer's use of formal reviews,
- Developer's use of a defect tracking system,
- Developer's use of a risk management system,
- Developer's reuse approach, and
- Maturity of the development artifacts.

The Software Characteristics sub-category is comprised of the following criteria;
- Complexity of the software,
- Degree of innovation,
- Size of the system.

Each criterion is assessed and given a score between 1 and 5. The associated weight is applied to the score and then summed for each sub-category. The sub-categories have their weights applied to them and they too are summed for the resultant Error Potential score.

The SILAP model was first written in C so that stochastic simulations could be ran. The model contains 13 input variables and 3 internal variables (the sub-categories of "Consequences" which, in turn, get broken down to the other 13 variables). Each variable has a two parts: a "weight" and a setting from the user. The model calculates Error Potential (EP) as follows:

```
//Weighted sub-categories and their criteria
 //pre-defined as a result of the Delphi sessions
// 13 weights for the 13 input variables:
    1. real CMM_weight=0.0764;
    2. real Complexity_weight=0.547;
    3. real ConfigManagement_weight=0.0962;
    4. real DefectTracking_weight=0.0873;
    5. real Experience_weight= 0.828;
    6. real FormalReviews_weight=0.1119;
    7. real Innovation_weight=0.351;
    8. real Maturity_weight=0.242;
    9. real Organization_weight= 0.172;
    10. real Reuse_weight=0.226;
    11. real RiskManagement_weight=0.0647;
    12. real SoftwareSize_weight=0.102;
    13. real Standards_weight=0.0955;
// 3 weights for the 3 input variables
    14. real SoftwareCharacteristics_weight=0.172;
    15. real SoftwareDevelopment_weight = 0.579;
    16. real SoftwareProcess_weight=0.249;

//Compute sub-category scores
real DV = (Experience_weight * devpExperience) +
(Organization_weight * devpOrganization)

real SW = (Complexity_weight * swComplexity) +
(Innovation_weight * swInnovation) +
(SoftwareSize_weight * swSize)

real PR = (Standards_weight * prUseOfStandards) +
(ConfigManagement_weight * prUseOfConfigMgt) +
(CMM_weight * prCMMLevel) +
(FormalReviews_weight * prUseOfFormalReviews) +
(DefectTracking_weight * prUseOfDfctTracking) +
(RiskManagement_weight * prUseOfRiskMgt) +
(Reuse_weight * prReuseApproach) +
(Maturity_weight * prArtifactMaturity)

//Compute final Error Potential Score
real EP = (SoftwareDevelopment_weight * DV) +
(SoftwareProcess_weight * PR) +
(SoftwareCharacteristics_weight * SW)
```

The model seems simple enough, but in practice is surprisingly difficult to reason about. After numerous fruitless sessions arguing the merits of different combinations of inputs, we turned to a three part automatic method to simplify those discussions:

1. Sampling studies;
2. Stability studies;
3. Specialization studies;

This three-part procedure is discussed below.

5. Sampling Studies

Before we can learn from the model, we need to know whether we have exercised the model sufficiently to uncover all of its nuances. Hence, the first step in our kind of studies is a *sampling study* to check that we are executing the model enough.

In a sampling study, we compare the results from N runs and 10*N runs. If the distributions in the model output change, then we run again at 100*N. This repeats till the model's output stabilizes.

We chose to compare the results of 500, 5,000, and 50,000 runs of the model. We compared the results using simple visualization capabilities provided by the WEKA tool [15]. Figures 2 through 4 show the affects that each criterion, within Error Potential, has on the final Error Potential score as a result of 500, 5,000, and 50,000 simulations.

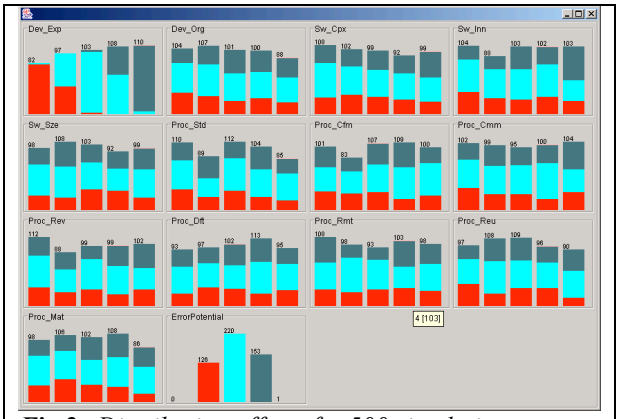

***Fig 2**. Distribution affects for 500 simulations.*

***Fig 3**. Distribution affects for 5,000 simulations*

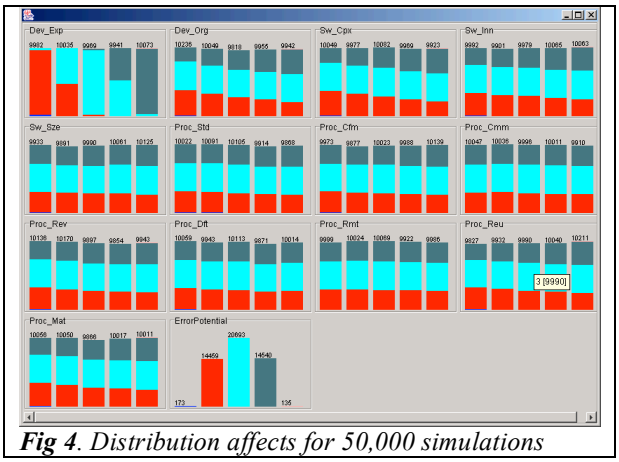

***Fig 4**. Distribution affects for 50,000 simulations*

As an example, for all three figures, the second bar chart in the first row represents the criterion "Developer's Organization". The last bar chart in the last row represents the distribution of Error Potential scores, red represents an error potential score of 2, turquoise represents an error potential of 3, et cetera. In Figure 2 we see that roughly 25% of all the scores for "Developer's Organization" (values range from 1-5) belongs to an Error Potential score of 2. Although it does fluctuate somewhat, a score of 1 contributes more to an Error Potential score of 1 than a score of 3 does. What we are looking for is stability across the simulation runs. Also, we see that roughly 50% of all the scores belong to an Error Potential score of 3 and then the remaining 25% belong to an error potential score of 4.

A visual comparison of these figures shows that there were differences between 500 and 5,000 simulations. For example, the second bar chart in the second row in all the figures represents the criterion "Developer's Use of Standards". Comparing all the figures, you can see how the distribution of red scores (error potential of 2) for the criterion "Developer's Use of Standards" goes from being very erratic to more stable, actually there is a large difference between Figure 2 and Figure 3 but not much between Figure 3 and Figure 4. Performing the same comparisons for all the criterion allow us to conclude that 5000 runs are adequate for sampling our models and that there is not any difference between 5,000 simulations and 50,000 simulations.

6. Stability Studies

TAR3 learned dozens of small rules describing the key factors that most changed the performance of the model. A very small example of that output is shown here:

```
Baseline: [as is]
       bad: ~~~                  [  2 =  11%]
      good: ~~~~~~~~~~~~~~~~~~   [ 16 =  89%]

Treatment:[Performance=2]
       bad:                     [  0 =   0%]
      good: ~~~~~~~~~~~~~~~~~   [  6 = 100%]

Treatment:[HumanSafety=2]
       bad:                     [  0 =   0%]
      good: ~~~~~~~~~~~~~~~~~   [  9 = 100%]
```

This example has been simplified for the purposes of exposition. The key features of this output are:

- The *first rule* describes the baseline as is situation. In this tiny example, 18 examples were run through the model and 11% of them were scored *bad* by some oracle.
- The *subsequent rules* describe strategies for changing that baseline. In this example, TAR3 has been told to try to avoid bad and seek more good results. Hence its control rules try to reduce *bad* and increase *good*. Here, TAR3 is proposing two treatments: setting "performance" to 2 or setting "human safety" to 2. In either case, all the bad disappears and 100% of the outputs are good.

A concern with TAR3's rule generation is that the generated rules are just some of the rules which might be generated from the space of possible models. Hence we conduct a *stability study* to see if the treatments are stable; i.e. occur in multiple sub-samples of the data.

Since our sampling study showed that 5000 runs are adequate for this domain, we designed our stability study as follows:

- Ran the simulations 5,000 times,
- Ten times, randomly selected 90% of that data for learning. The models learned in this way were assessed using the remaining 10% of the data,
- Only report the treatments found in the majority of the sub-samples (here, we declared that "majority" meant 7 or more of the sub-samples), and
- Sort the majority treatments according to "worth".

In this study, we used a negative definition of "worth" and asked TAR3 for combinations of inputs that select for highest Error Potential (which, in this model, was an Error Potential of 5). That is, we were looking for the management actions with the worst possible effects.

Initially, inputs were selected completely at random. This was changed later (see next section).

The sorted results from the stability study are listed below. Recall that the top-ranked treatment is the worst thing we can do:

1. Score the Developer's Experience a 5, which means the developer has hardly any experience in building the system, or
2. Score the Developer's Experience a 5 and the Developer's Reuse approach a 1. This means the developer has hardly any experience in building the system but they are reusing software, their approach is well-established, and the software was originally built with reuse as an objective, or
3. Score the Developer's Experience a 5 and Software Innovation a 1, which means the developer has hardly any experience in building the system but similar software has been flown on previous mission, or
4. Score the Developer's Experience a 5 and Software Complexity a 1, which means the developer has hardly any experience in building the system but the software does not have any logical conditions or intense numerical solutions, or
5. Score the Developer's Experience a 5 and the Developer's Organization a 1, which means the developer has hardly any experience in building the system but they are not subcontracting any of the work and the entire software development team resides in the same location.

These 5 treatments show that the greatest affect we can have on a Project's Error Potential score is when the Project has very little experience in building similar systems. This simply means, if you execute the SILAP model for Project A, and you execute the SILAP model for Project B, and then you execute the model for Project C, the criterion that has the biggest affect on their Error Potential scores is whether or not they have experience in building similar systems.

7. Specialization Studies

An assertion that we make is that similar NASA software projects would yield different treatments. This is a *specialization effect*; i.e. that the treatments found for one special set of inputs are different to the treatments found from another. For example, Project A and Project B are both Human Space Flight missions. The criterion or criteria that would have the

Criterion	Value	Explanation
Experience	1	The developer's have built these systems before and have several years of domain experience.
Development Organization	4	Usually more than one NASA Center is involved with Human Space Flight missions.
Degree of Innovation	1	Normally, the software is not doing anything that has not been tested during a previous flight.
Use of Standards	1	Developers incorporate NASA standards as well as accepted industry standards.
Use of Configuration Management	1	Tools, as well as established methods, for configuration management are integrated into the development effort.
CMM Level	3	Methods and processes are characteristic of a Level 3 organization.
Use of Formal Reviews	1	Formal reviews are essential for the Human Space missions and they are followed and have predefined criteria.
Use of a Defect Tracking System	1	Defect tracking tools are well established at the software level and in place for the development efforts.
Use of a Risk Management System	3	Risk management tools are established at the Project level but they are not consistently used at the software level.
Artifact Maturity	1	The majority of the software artifacts are logically in a state that is similar to the schedule.

Fig 5. *Criteria that are set to constant values during the simulation to reflect Human Space Flight missions*

biggest affect on their Error Potential score would be different than a robotic mission to Mars.

To test our assertion, we conducted a *specialization study*. In this study, we stopped picking inputs purely at random. Rather, we constrained those variables to typical values seen in Human Space Flight missions (see Figure 5). For those specialized inputs, we ran 5,000 simulations and conducted a stability study on the output. For the stability study, it revealed that changing the software complexity, for all the samples, would have the biggest affect on the error potential.

```
Baseline: [No Treatment]
       1: ~~~                  [ 698 = 14%]
       2: ~~~~~~~~~~~~~~~~~~   [4302 = 86%]

Treatment:[Complexity = 3]
       1:                      [   0 =   0%]
       2: ~~~~~~~~~~~~~~~~~~   [1071 = 100%]

Treatment:[Complexity = 5]
       1:                      [   0 =   0%]
       2: ~~~~~~~~~~~~~~~~~    [ 971 = 100%]
```

The results of our treatment learner on the set of 5,000 runs are significantly different that when we previously executed the model using randomly selected inputs. Specifically, the criterion that has the biggest affect on the Error Potential score for Human Space Flight missions was Software Complexity. For all the samples, changing the software complexity score for the Project has the biggest influence on the Error Potential, as opposed to the Developer's Experience.

8. Related Work

A standard method for understanding models is some sort of *sensitivity analysis*. Sensitivity analysis is a huge field that includes many techniques (some of which have overlapping definitions). One survey [8] argues that what is called "sensitivity analysis" divides up into the tasks shown in Figure 6; (i.e. *validation*, *screening, true sensitivity analysis*, *uncertainty analysis* and, finally, finding and generating some *optimization* policy). Validation and screening usually precede the other tasks but it seems to be a matter of personal style whether or not a sensitivity analysis follows uncertainty analysis.

Traditional sensitivity analysis is a labor-intensive, time-consuming task. Researchers in this area can spend their whole career working in just one of the areas shown in Figure 6. Often, extensive knowledge is required about the internal features of the model being simulated. Unless managers have access to specialists in sensitivity analysis, then they may not be able to understand all the implications of their models that they execute.

A unique feature of this work is the use of data mining to reduce the effort and skill-level required for sensitivity analysis. In our approach, managers just need to tag some of the model output variables with "utilities"; i.e. their evaluation of how important certain variables are to them. Our data miners would then perform many random simulations, cache the results, and then automatically learn what input parameters lead to preferred outputs.

Validation: e,g. check that data models generated from the model matches known domain values

Screening: e.g. dimensionality reduction via, say, ignoring input variables that are not highly correlated to outputs

Sensitivity analysis: e.g. execute using the minimum and maximum of all input values

True uncertainty analysis: e.g. treat each input as a random variable with a mean and standard deviation, then perform Monte Carlo simulations

Optimization: e.g. find some combination of inputs that improve the output values.

Fig 6*: Sensitivity analysis methods*

9. Discussion

Our goal was optimizing debates; i.e. the earlier termination of debates and terminating on better decisions.

TAR3 is a method for such an optimization. Our results show us the *least number* of critical decisions that *most influence* the Error Potential score. There is a wide range of opinion on how to best reduce that error score. For example, someone might advocate that software complexity has the greatest affect on the Error Potential score, while someone else may assert that a combination of software complexity and the size of the system have the biggest influence on the Error Potential score. Arguments such as these are common when using models to make decisions. Our treatment learner is able to minimize the argument space and reveal those input variables that have the biggest influence on the score. In our case, which input variables (when changed) have the biggest affect on the Error Potential score.

TAR3 optimizes debates another way. One curious feature of the above results was that nearly half the variables in the model *never appeared in any treatment*. This is an important observation since it means efforts to collect or specialize or refine those variables is possibly a waste of time and should be avoided.

Significantly, we show above that the SILAP model supports specialization effects where the learned conclusions vary depending on the class of software used to constrain the input conditions. This is another important observation since it means that we need to search for classes of NASA software development projects (i.e. Human Space Flight missions versus Robotic missions) that yield different treatments to the input space that ultimately affect the assessed Error Potential score for the Project. This in turn affects the different levels of IV&V recommended for different classes of missions. That is, SILAP is not a one-size fits all *model*, and care should be taken to avoid applying the wrong conclusions from different classes of software to the current domain.

While the domain explored here is quite specific (process and tool options for IV&V), our claim is that the technology used here is quite general. Given an executable model generated from a DELPHI session, the data mining methods used here can generate succinct summaries of even quite complex business knowledge. When applying this approach to other domains, we highly recommend sampling, stability, and specialization studies to better understand what the data miners are revealing.

The reader might doubt that tools like TAR3 are required for models as simple as the SILAP model shown in Section 4. If so, they are invited to repeat our analysis by some other means. To be a fair comparison, that alternative method must:

- Produce results that are demonstrable stable across a wide range of inputs,
- Those stable conclusions must be as simple to express as TAR3's minimal rules, and
- That analysis must be fully automated and as simple to run as using TAR3.

For our future work, we are interested in exploring the stability of our conclusions across instabilities in SILAP's weighting factors. SILAP's conclusions are critically dependent on those factors (which are generated in DELHPI sessions). As shown in Section 4, some of those figures are specified in great detail (up to four significant figures). If we could identify which factors were most "brittle" (i.e. small variations had largest impacts on the output) then we would hold further DELPHI sessions to better define those critical brittle factors.

10. References

[1] Bay, S.B., M.J. Pazzani, Detecting Change in Categorical Data:s Mining Contrast Sets, *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining, 1999*

[2] Bylander, T., D. Allemang, M. C. Tanner and J. R. Josephson (1991): The computational complexity of abduction. In Artificial Intelligence 49

[3] Costello, K., Software Integrity Level Assessment Process (SILAP), NASA IV&V Facility, 2005

[4] DeKleer, J. An Assumption-Based TMS. Artificial Intelligence, 28:163--196, 1986

[5] Feather M.S., Menzies T., Converging on the Optimal Attainment of Requirements (2002), IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany. Available from http://menzies.us/pdf/02re02.pdf

[6] Hall, M. and Holmes, G. . Benchmarking attribute selection techniques for discrete class data mining.

IEEE Transactions on Knowledge and Data Engineering. 15(3), May/June 2003.

[7] Kakas, A. C., Kowalski, R. A., and Toni, F., Abductive Logic Programming, Journal of Logic and Computation 2(6):719—770, 1992.

[8] Kliijnen, J.P.C., Sensitivity Analysis and Related Analyses: a Survey of Statistical Techniques, Journal Statistical Computation and Simulation, pages 111-142, Number 1--4, Volume 57, 1987.

[9] Madravio, M., Menzies T., Singh, H., Many Maybes Mean (Mostly) the Same Thing; Soft Computing in Software Engineering; 2003; Springer-Verlag; Available from http://menzies.us/pdf/03maybe.pdf

[10] Menzies T., Hu, Y., Data Mining for Very Busy People; IEEE Computer; October 2003; Available from http://menzies.us/pdf/03tar2.pdf

[11] Menzies T., Kiper, J., Feather M., Improved software engineering decision support through automatic argument reduction tools (2003), SEDECS'2003: the 2nd International Workshop on Software Engineering Decision Support (part of SEKE2003). http://menzies.us/pdf/03star1.pdf

[12] Poole, D.L., Goebel, R., Aleliunas, R., "Theorist: a logical reasoning system for defaults and diagnosis", in N. Cercone and G. McCalla (Eds.) The Knowledge Frontier: Essays in the Representation of Knowledge, Springer Varlag, New York, 1987, pp. 331-352.

[13] Reiter, R. 1980. A Logic for Default Reasoning. Artificial Intelligence 13:81--132

[14] Simon, H, The Science of the Artifical, MIT Press, 1996 (second edition).

[15] WEKA, University of Waikato, http://www.cs.waikato.ac.nz/~ml/weka/

Acknowledgements

This research was conducted at Portland State University and the IV&V Facility under partial funding from the NASA Office of Safety and Mission led by the NASA IV\&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.