I. Case Study: Studying the CMM

Our first case studies explored reasoning over a symbolic model containing much uncertainty. The second studied explored a numeric model where all the influences were precisely specified. This third study takes a numeric model and adds a large degree of uncertainty in the numerics.

This study uses a cost/benefits model of CMM level 2 CMM2 [1, p125-191]. We elected to study CMM2 since, in our experience, many organizations can achieve at least this level. CMM2 is less concerned with issues of (e.g.) which design pattern to apply, than with what overall project structure should be implemented. Improving CMM2-style decisions is important since in early software life cycle, many CMM2-style decisions effect the resource allocation for the rest of the project.

CMM2 was encoded using the JANE propositional rule-based language [2]. JANE's rules take the form $Goal if SubGoals$. Each $SubGoal$ contributes some $Cost$ and $Chances$ to the $Goal$. JANE's $Chances$ define the extent to which a belief in one vertex can propagate to another. $Cost$s let an analyst model the common situation where some of the $Cost$ of some procedure is amortized by reusing its results many times. Hence, the *first* time we use a proposition, we incur its $Cost$ but afterwards, that proposition is free of charge.

The $Cost$ and $Chances$ of a proposition are either provided by the JANE programmer or computed at runtime via a traversal of the rules:

- When searching *X if not A*, the $Chances$ of *X* are *1-Chances(A)* and $Cost(X) = Cost(A)$.
- When searching *X if A and B and C*, the $Cost$ and $Chances$ of *X* is the product of the cost and chances of *A,B,C*.
- When searching *X if A or B or C*, then the $Cost$ and $Chances$ of *X* is taken from the first member of *A,B,C* that is satisfied.

These and, or, not operators can be insufficient to capture the decision making of business users. For examples, in our experience, business users select CMM-2 options, often in a somewhat arbitrary manner. To model this, JANE includes a $rany$ operator:

- The $rany$ operator is like or except that (e.g.) *X if A rany B rany C* succeeds if some random number of *A,B,C* (greater than one) succeeds. Unlike and, or which explore their operands in a left-to-right order, $rany$ explores its $SubGoals$ is a random order. If at least one succeeds, then the $Cost$ and $Chances$ of *X* is summed from the satisfied members of *A,B,C*.

$Rany$ is useful when searching for subsets that contribute to some conclusion. For example, the JANE rule in Figure 1 offers several essential features of *stableRequirements* plus several optional factors relating to monitoring change in evolving projects. The essential features are and-ed together while the optional factors are $rany$-ed together.

Figure 1 shows 11 propositions. Our model of CMM2, written in JANE, has 55 proposition ($range = \{t, f\}$) and is available from the authors. Of those 55 propositions, 27 were identified as management actions that could be changed by managers (see Figure 2).

```
stableRequirements
    if   effectiveReviews
    and requirementsUsed
    and sEteamParticipatesInPlanning
    and documentedRequirements
    and sQAactivities
    and (reviewRequirementChanges
          rany softwareConfigurationManagement
          rany baselineChangesControlled
          rany workProductsIdentified
          rany softwareTracking
         ).
```

Fig. 1. Part of CMM-2, encoded in the JANE language.

Apart from $rany$, JANE supports one other mechanisms for exploring the space of possibilities within CMM-2. When defining $Cost$s and $Chances$, the programmer can supply a *range* and a *skew*. For example:

```
goodUnitTesting and cost = 1 to +5
```

defines the *cost* of *goodUnitTesting* as being somewhere in the range 1 to 5, with the mean skewed slightly towards 5 (denoted by the "+").

Similarly, while all the $Chances$ values were based on expert judgment, their precise value is subjective. Hence, each such $Chances$ value X was altered to be a range

```
chances = 0.7*X to 1.3*X
```

During a simulation, the *first* time a $Cost$ or $Chance$ is accessed, it is assigned randomly according to the range and skew. The assignment is cached so that all subsequent accesses use the same randomly generated value. After each simulation, the cache is cleared. After thousands of simulations, JANE can sample the "what-if" behavior resulting from different assignments within the range and many different $rany$ choices.

Data from 2000 simulations was passed from the CMM-2 model to TAR2. Each simulation was classified into one of four classes:
Score=0: *High cost, low chance*
Score=1: *Low cost, low chance*
Score=2: *High cost, high chance*

baselineAudits, baselineChangesControlled, changeRequestsHandled, changesCommunicated, configurationItemStatus-Recorded, deviationsDocumented, documentedDevelopmentPlan, documentedProjectPlan, earlyPlanning, formalReviewsAtMilestones, goodUnitTesting, identifiedWorkProducts, periodicSoftwareReviews	*planRevised, requirementsReview, requirementsUsed, reviewRequirementChanges, risksTracked, SCMplan, SCMplanUsed, SElifeCycleDefined, SEteam-ParticipatesInPlanning, SEteamParticipatesOnProposal, SQAauditsProducts, SQAplan, SQAplanUsed, SQAreviewActivities, workProductsIdentified*

Fig. 2. Management actions in the CMM2 model. SQA= software quality assurance and SCM= software configuration management)

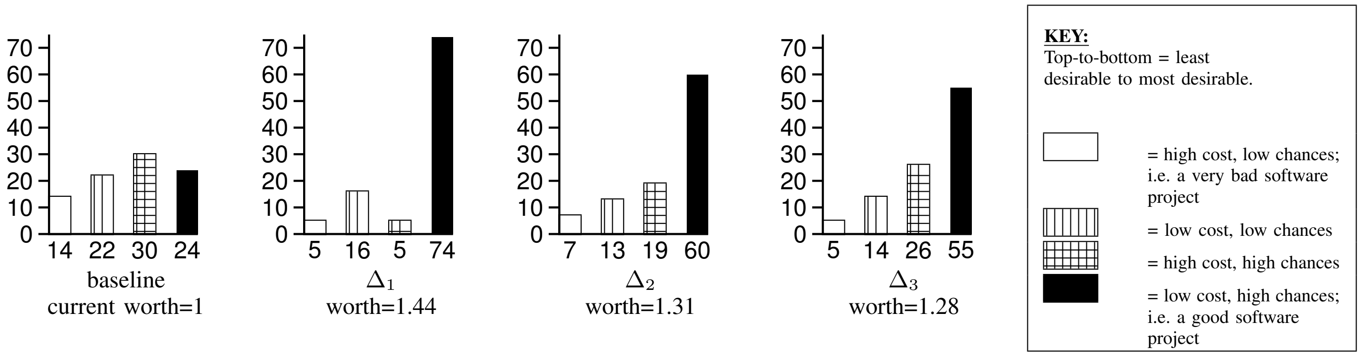

Fig. 3. Ratios of different software project types seen in four situations.

Δ_1:	*requirementsUsed.*`Cost`*=lower and not periodicSoftwareReviews and formalReviewsAtMilestones.*`Cost`*=lower*
Δ_2:	*requirementsUsed.*`Cost`*=lower and goodUnitTesting.*`Cost`*=middle and formalReviewsAtMilestones.*`Cost`*=lower*
Δ_3:	*goodUnitTesting.*`Cost`*=lower and periodicSoftwareReviews.*`Cost`*=middle and formalReviewsAtMilestones.*`Cost`*=lower*

Fig. 4. The three best treatments found in the CMM-2 model.

Score=4: *Low cost*, *high chance*.

That is, our preferred projects are cheap and highly likely while expensive, low odds projects are to be avoided.

Figure 3 shows three sets of actions learned by TAR2. The left-hand-side histogram shows the baseline distributions seen in the 2000 simulations. The other histograms show how those ratios change after applying the treatments learned by TAR2; The *worth* of each option is a reflection of the proportion of good and bad projects, compared to the baseline, i.e. ($worth(baseline) = 1$). Note that as *worth* increases, the proportion of preferred projects also increases.

Figure 4 shows the three best treatments ($\Delta_1, \Delta_2, \Delta_3$) found using this technique (and Figure 3 compared the effects of these treatments to the untreated examples). Note that the values of each attribute are reported using the tags *no*, *lower*, *middle*, or *upper*. In treatment learning, continuous attribute ranges are divided into N-discrete bands based on percentile positions. For N=3, we can name the bands *lower*, *middle*, *upper* for the lower, middle, and upper 33% percentile bands.

In Figure 4, the treatments are advising to lower the cost of:

- *Using requirements:* This could be accomplished by (e.g.) by sharing them around the development team in some searchable hypertext format
- *Performing formal reviews at milestones:* This could be accomplished by (e.g.) using ultra-lightweight formal methods such as proposed by Leveson [3].
- *Performing good unit testing:* This could be accomplished by (e.g.) hiring better test engineers.

An interesting feature of Figure 4 is what is *missing*:

- None of the treatments proposed adjusting the $Chances$ of any action. In this study, changing $Cost$ will suffice.
- Of the 27 actions listed in in Figure 2, only the four underlined actions appear in the top three treatments. That is, management commitment to undertake 27-4=23 of the actions is less useful than changing on $formalReviewsAtMilestones$, $goodUnitTesting$, $periodicSoftwareReviews$, and $requirementsUsed$
- The value *not* in Δ_1 is a recommendation against *periodicSoftwareReviews* (plus lowering the costs of using requirements and formal reviews at milestones). Note that if *periodicSoftwareReviews* are conducted, Δ_3 is saying that there is no apparent need to reduce the cost of such reviews.

More generally, in a result consistent with the prior studies, despite the uncertainties introduced by $rany$ and the $cost/chances$ ranges, TAR2 found a small number of CMM-2 process options that have a significant impact on the project.

XXX jim- some phrase about external validity. only good for the numbers we used in this study. blah blah blah

References

[1] M. Paulk, C. Weber, B. Curtis, and M. Chriss, *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.

[2] T. Menzies and J. Kiper, "Better reasoning about software engineering activities," in *ASE-2001*, 2001, available from http://menzies.us/pdf/01ase.pdf.

[3] N. Leveson, S. Cha, and T. Shimall, "Safety verification of ADA programs using software fault trees," *IEEE Software*, vol. 8, no. 7, pp. 48–59, July 1991.