Submitted to the 5th IEEE International Symosium on Requirements Engineering
(ISRE 2001) Toronto, Canada, `http://www.re01.org`
WP ref: 01/isre/jane (February 22, 2001).

RE '01 Fifth IEEE International Symposium on Requirements Engineering
August 27-31, 2001
Royal York Hotel, Toronto, Canada

How to Argue Less

Tim Menzies
Dept. Electrical Engineering & Computer Engineering,
University of British Columbia, Vancouver, Canada;
`<tim@menzies.com>`

James D. Kiper
Dept. Computer Science & Systems Analysis,
Miami University, Oxford, Ohio, USA;
`<kiperjd@muohio.edu>`

Abstract

Requirements engineering can stall if all stakeholder disputes are explored; e.g. a mere 20 boolean options implies $2^{20} > 1,000,000$ *possible arguments. One method of reducing this argument space is to focus the arguments on core issues and ignore the peripheral arguments. Theoretical and experimental studies strongly suggest that, in the usual case, a space of arguments contains many irrelevant and repeated disputes. Hence, the space of all critical arguments may be dramatically smaller than the space of all arguments. It is argued here that this reduction can be implemented via* abduction *plus* induction. *Abduction can extract the consistent conclusions derivable from a requirements model. Induction can learn from that sample the attributes that most change the behaviour of the model. Experiments with this abduction-plus-induction approach have found that a very small number of critical factors can be found within seemingly huge argument spaces. A strong theoretical case can be made that this approach will apply to many domains and scale to very large models.*

1. Introduction

How cheaply can we build *useful* requirements models? Models may be deemed useful for many reasons, but our sense here is that a *useful* model is one that can be used to make definite conclusions. Many requirements engineering (RE) researchers such as van Lamsweerde [17] argue that useful RE models should be high-quality and comprise detailed and rigorously expressed product requirements. The benefit of such models is that they can be studied by sophisticated formal tools to deliver a detailed understanding of a domain. The cost of such models is their construction effort. The details required by such models may be unavailable in early life cycle or too expensive to collect. In safety-critical applications with large budgets for development, the cost of building such models can be justified. However, other domains may require cheaper alternatives to expensive and time-consuming RE.

The problem with cheaper RE is that it may generate less useful models. As the number of uncertainties grows within a model, the number of possible alternatives increases exponentially. For example, suppose an RE model is unclear on 20 issues, each with a binary value of yes/no. 20 binary choices implies $2^{20} > 1,000,000$ arguments.

In the case of multiple viewpoints RE, cheap modeling implies a potential avalanche of arguments. Viewpoints-based RE assumes that information comes from multiple stakeholders. Each set of information is maintained separately as an independent *viewpoint* (one viewpoint for each stakeholder). Each viewpoint is a partial description of some perspective on a system. Viewpoints have been used to characterize entities in a system's environment [5], to characterize different classes of users [15], to distinguish between stakeholder terminologies [16], and to partition the requirements process into loosely coupled work pieces [14]. This support for loosely coupled work pieces is a key advantage of viewpoints. We can no longer assume that software will be built by a single team in a single location using a single tool kit for a single purpose. Given recent advances in Internet technology (e.g. CORBA, the world-wide web), we should expect that software development will be geographically distributed. For such distributed development, it is pragmatic to permit the parallel development of separate 'work pieces" (a.k.a. viewpoints) that will have to be unified at some later date. Building viewpoints must be cheap lest the overall cost of multi-viewpoint RE overwhelms the budget. But cheap models incur the uncertainty problem. How are we to manage a disucssion between multiple stakeholders for our (e.g.) 2^{20} arguments?

This paper presents a novel method for arguing less. Let the union of the viewpoints be the *argument space*. It will be claimed that much of an argument space is irrelevant, redundant, or dependent on other parts of the argument space. Hence, one method of arguing less would be to focus first on the key portions of the argument space. Finding the key arguments is theoretically NP-hard [7] and, in practice, may be impractical for all but the smallest models. However, the *funnel theory* first proposed at RE99 by Menzies, Easterbrook, Nuseibeh and Waugh [10] gives new hope for a tractable search for the key arguments. This paper presents a random search engine called CHEETAH that exercises argument spaces expressed in the JANE rule-based language. A monitor called TARZAN watches from above as CHEETAH chases JANE around the argument space. TARZAN builds a log of Jane's behaviour and learns how to nudge JANE into better behaviour. Funnel theory (described in §3) predicts that such nudges are few in number and fast to find.

Technically, CHEETAH is a *randomized abductive inference engine* and TARZAN is an *inductive learner*. Therefore the core message of this paper is that we can argue less by applying randomized abduction plus induction to our combined viewpoints.

This article is structured as follows. The next section documents evidence for funnel theory. The details of that theory are then discussed, followed by an introduction to the JANE/CHEETAH/TARZAN toolkit. Finally, we will discuss the generality of this approach.

2. The Funnel Phenomena

The reader may doubt that big arguments can reduce to a much smaller set of key arguments. However this section offers some dramatic examples of such a reduction.

Menzies & Sinsel found that a space of 54 million options contained found two key variables that could most control the rest of the system [12]. In that application, a COCOMO-based tool [6] was used to evaluate the risk that a NASA software project would suffer from develop-time overrun. The tool used in that study required a guesstimate of the source lines of code (SLOC) in the system and certain internal tuning parameters which, ideally, are learnt from historical data. Lacking such data, Menzies & Sinsel used three guesses for SLOC and three sets of tunings which they took from the literature. Competing stakeholders proposed 11 changes to a project. Some of the project features were unclear and, for those features, project managers could only offer ranges for the required inputs to the COCOMO-based tool. These ranges offered 2930 possible combinations for the inputs. When combined with the other uncertainties, this generated a space of 54 million possibilities ($2930 * 2^{11}$*three guesses for SLOC * three tunings).

Faced with this overdose of possibilities, Menzies & Sinsel performed 50,000 Monte Carlo simulations where the inputs were taken from the 54 million possibilities. A machine learning program generated decision trees from the 50,000 runs. A tree query language called TARZAN then swung through the learnt trees looking for the least number of attribute ranges that had the biggest impact on the overall software development risk. TARZAN found that of the 11 proposed changes, seven had a little overall impact. Of the remaining four, two were clearly inferior in reducing the system risk. This left two attribute ranges with the clearest benefit in reducing software project risk.

In another study, Menzies, Easterbrook, Nuseibeh and Waugh [10] found that most of the choices made within a space of conflicts had the same net effect. That study compared two *abductive inference strategies*. Abduction is a method of tracking the choices made while studying a model. An abductive inference engine searches for goals while ensuring that all choices remain compatible [3]. When faced with incompatible choices, an abductive device has at least two choices. In *full worlds search*, the abductive device forks one *world of belief* for each possible resolution to the choice. In *random worlds search*, the abductive device selects one resolution at random, then continues on. Random worlds search is usually performed inside a "rest-retry" mechanism. That is, for a limited number of retries, when the random search runs out of new options, all options are retracted and the whole random worlds inference procedure runs again. In a very large case study (over a million runs), Menzies, Easterbrook, Nuseibeh and Waugh found that the average difference in reachable goals between the random worlds search and full worlds search was less than 6% (!!). That is, random conflict resolution reached as many parts of an argument space as a more rigorous method.

In yet another study, Menzies and Micheal [11] showed that random worlds search found 98% of the goals found by a full worlds search [11] (a result consistent with Menzies, Easterbrook, Nuseibeh and Waugh). More interesting from a pragmatic perspective, the full worlds search ran in time exponential to model size while the randomized abductive search cran much faster and scaled up to very large models.

3. Funnel Theory

Funnel theory is a claim that within the space of arguments, there exist a small number of key decisions that determine all others. As we shall see, funnel theory explains the above observations.

To introduce funnels, we first say that an argument space supports *reasons*; i.e. chains of reasoning that link inputs in a certain context to desired goals. Chains have links of at least two types. Firstly, there are links that clash with other links. Secondly, there are the links that depend on other links. One method of arguing less is to first debate the non-dependent clashing links. The resolutions to these arguments will have the greatest impact of reducing the subsequent argument(s). For example, suppose the following argument space is explored using the invariant $nogood(X, \neg X)$ and everything that is not a $context$ or a $goal$ is open to debate:

$$\begin{tikzcd}
 & a \arrow[r] & b \arrow[r] & c \arrow[r] & d \arrow[r] & e & \\
context1 \arrow[r] & f \arrow[r] & g \arrow[r] & h \arrow[r] & i \arrow[r] & j \arrow[r] & goal \\
context2 \arrow[r] & k \arrow[r] & \neg g \arrow[r] & l \arrow[r] & m \arrow[r] & \neg j \arrow[r] & goal \\
 & n \arrow[r] & o \arrow[r] & p \arrow[r] & q \arrow[r] & \neg e &
\end{tikzcd}$$

While all of $\{a, b, ..q\}$ is subject to discussion, in the context of reaching some specified goals from *context1* and *context2*, the only important disputes are the clashes $\{g, \neg g, j, \neg j\}$. The $\{e, \neg e\}$ clash is not exercised in the context of $context1, context2 \vdash goal$ since no reason uses e or $\neg e$. Since $\{j, \neg j\}$ are fully dependent on $\{g, \neg g\}$, then the core of this argument is one variable ({g}) with two disputed values: true and false.

The *funnel* of an argument space contains the non-dependent clashing links; e.g. $\{g\}$[1] The arguments with *greatest information content* are the arguments about the funnel variables, since these variables set the others. If the space contains *narrow funnels* then the total argument space can be greatly reduced to a small number of highly informative disputes about funnel variables. Stakeholders are still free to debate whatever they want (and they will, seemingly endlessly), but with this approach, the requirements engineer can steer the discussion towards the issues that tells us most about a domain. The net effect can be less arguments. Suppose our stakeholders agree that g is true, then in the context of arguing about how $context1, context2 \vdash goal$, the argument space reduces to:

$$context1 \longrightarrow f \longrightarrow g \longrightarrow h \longrightarrow i \longrightarrow j \longrightarrow goal$$

The reasoning starting with k has been culled since, by endorsing g, we must rejects all lines of reasoning that use $\neg g$. Also, the reasoning starting with a, n are ignored since they are irrelevant in this context; i.e. they do not participate in reaching a desired goal. Further, in this context, there is little point arguing about $\{f, h, i, j\}$ since if any of these are false, then no goal can be reached.

This small example shows how to argue less through funnel-based reasoning. Funnel-based argumentation finds the key arguments, and ignores numerous irrelevant arguments. In the above example, a argument space containing up to $2^{16} = 65536$ discussions about 16 boolean variables $\{a..q\}$ has been reduced to one discussion about one variable; i.e. "is g true or false?".

Funnel theory explains the observations seen in §2:

[1]Readers familiar with the ATMS [1] will note the similarities between the funnel and ATMS *minimal environments.* However, while both approaches rely on some *nogood* invariant, there are significant differences between the consistency-based *total envisionments* of the ATMS and the set-covering *relevant envisionments* discussed here; see [7] for details.

- The 54 million options about the software project could reduce to two since the COCOMO-based tool contained narrow funnels.
- The random worlds search used by Menzies, Easterbrook, Nuseibeh, Waugh found as many goals as the full worlds search since both searches were controlled by the same funnels.
- The random worlds search used by Menzies & Micheal ran extremely fast since it could quickly sample the funnels without all the overheads of the more rigorous search.

4. An Argument Reduction Environment

A naive approach to funnel-based reasoning is to find the funnels using some sophisticated dependency-directed backtracking tools such as the ATMS [1] or HT4 [7]. Dependency-directed backtracking is a naive approach since (1) such reasoning has been shown to be very slow, both theoretically and in practice [7]; and (2) there is no need to *find* the funnel in order to *exploit* it. This second point is the key insight that resulted in this paper. We don't need to do anything special to find the funnel since *any* reasoning pathway to goals must pass through it (by definition). Repeated application of some fast random search technique will stumble across the funnel variables (providing that search technique reaches the goals). This section describes JANE/CHEETAH/TARZAN, a general toolkit for supporting less arguments based on randomized search. In the toolkit, randomized abduction and induction are performed by CHEETAH and TARZAN respectively. Both these modules navigage a space of options, defined as rules in the JANE language.

4.1. JANE

JANE is a simple rule-based language for expressing options in a domain. Each rule and fact in JANE is stamped with the name of the author and the time and date of its creation. Rules and facts from different stakeholders can hence be stored together in one rule-base. Also, each rule and fact gets a heuristic *chances* measure (range 0 to 1) that stores the likelihood of that fact/rule. Finally, a dollar *cost* value is added to each fact/rule. In the current version of JANE, *cost* is a once-off set-up cost. Hence, if (e.g.) a fact is accessed more than once, its associated dollar cost is only incurred the first time.

$Chances$ and *cost* need not be specified exactly. JANE authors can specify a minimum and maximum value, optionally marked with some "skew". For example, a sample JANE rule base, showing contributions from two stakeholders (Tim and Bob) is shown in Figure 1. Line 6 shows an exact specification of *cost*s and $chances$ while line 17 shows a

```
 1  tim= [month=jan,day=18,year=2001
 2       ,elm='tim@menzies.com'].
 3  bob= [month=feb,day=10,year=2001
 4       ,elm='robertf@zbm.com'].
 5
 6  tim says cost = 0 and chances = 1.
 7  r1 if    rich rors healthy rors content
 8      then happy.
 9  r2 if    not tranquil then rich.
10  r3 if    tranquil     then content.
11  r4 if    no sick      then healthy.
12  r5 if    overweight   then sick.
13  r6 if    no exercise  then overweight.
14  t7 if    baseball rany running rany swimming
15           rany football then exercise
16
17  bob says cost = 1 to +4 and  chances = +0 to 1.
18  r8  if   enthusiasm rand likesSweat
19      then baseball.
20  r9  if   enthusiasm rand likesSweat
21      then running.
22  r10 if   enthusiasm rand likesSweat
23      then football.
24
25  tim says cost = 1 to +4 and  chances = +0 to 1.
26  r11 if   enthusiasm rand not likesSweat
27      then swimming.
28
29  tim says cost = 2 and chances = 1.
30  r12 if   true then enthusiasm.
31
32  run  :- prove(happy).
33  runs :- time(proves(1000,'experience.dat').
```

Figure 1. A sample JANE knowledge base.

range specification with a "+" symbol showing the skew. Internally, the skew is implemented as a $beta(X)$ distribution with mean X. $+0\ to\ 1$ means that the range is the the random variable $0 + beta(0.33) * (1 - 0)$ while $1\ to\ +4$ means that the range is a random variable $1 + beta(0.67) * (4 - 1)$. Note also that the rules $\{r1, r2, r3, r4, r5, r6\}$ all have the same $cost$ and $chances$ shown at line 6. Similarly, the $cost$ and $chances$ of $\{r8, r9, r10\}$ is set at line 17.

4.2. CHEETAH

CHEETAH is a randomized abductive inference engine that interprets rules written in the JANE syntax. CHEETAH supports *assumption-based reasoning* and *random walk*.

A core concept within CHEETAH is the *assumption*. If a JANE rule condition requires some assumption, and there is no evidence for or against that condition, then CHEETAH just makes the assumption. For example, rule $r11$ can only prove $swimming$ if it assumes $not\ likesSweat$. This assumption rules out $baseball, running$ or $football$ since those conclusions require $likesSweat$ (see line 18,20,22 in Figure 1).

Since assumptions rule out other conclusions, CHEETAH uses a *random walk* mechanism for randomly selecting which assumptions are made. This random walk is a simple adaptation of standard disjunctions and conjunctions. In standard languages, if a test is specified as $X\ and\ Y\ and\ Z$ then that test is executed left-to-right to test X before Y before Z. CHEETAH supports the standard left-to-right and and or as well as a random ordered test $rand$ and ror. If a condition is specified (e.g.)

$$swimming\ ror\ football\ ror\ baseball$$

then the order of traversal is picked randomly. Recalling the last paragraph, then CHEETAH may or may not try to prove $swimming$ before $football$ in which case assumptions about our dislike of sweating would favor swimming and rule out the other sports.

When multiple methods exist for proving something X, then our belief in $\neg X$ should decrease. This is implemented via the noX operator which sums the evidence for X into Sum, then returns $1 - Sum$.

Other random walk operators of interest are $rors$ and $rany$ (see lines 7 and 14,15 in Figure 1). $Rors$ specifies a set of goals which we desire and $rany$ specifies a set of required, but not totally desirable goals. For example, $happiness$ might result from being $rich$, $healthy$ and $content$. However, in this imperfect world it is rare that we can achieve $rich$ and $healthy$ and $content$. Hence we combine them with a $rors$ to ask CHEETAH to try and prove as many of them as possible.

$Rany$ is similar in concept to $rors$, but opposite in intent. While $exercise$ could be done via $baseball$ and $running$ and $swimming$ and $football$, we probably don't want to do all four exercises at once since this might lead to (e.g.) muscle damage. Hence we combine them with a $rany$ which must prove at least one of them, but after that, it ignores some randomly selected portion of the $rany$ goals. As with $rand$ and ror, the traversal order of the testing in $rany$ and $rors$ is picked at random.

$Rors$ and $rany$ adopts the HT0 [11] method for traversing a space of assumptions: *one shot-proofs*, *random ordering*, plus *reset-retry*. When proving a set of goals $X_1\ rors\ X_2\ rors\ X_3 ...$, then the goal X_j only gets *one-shot*. If a proof of X_j fails, then the system does not backtrack to find different solutions to prior goals $X_1 ... X_i (i = j - 1)$. Instead, X_j is marked as unproved and $rors$ skips on to the next goal X_{j+1}. One-shot is a very weak method for proving something and only it works in domains with narrow funnels (where any shot in the dark while hit something of interest). Numerous experiments [11] strongly suggest that when one-shot is combined with *reset-retry*, then one-shot greatly reduces the computational cost of searching a space of contractions. Note that if we juggled the order of the goals, then we might avoid making an assumption before search for X_j that makes X_j impossible. Such order juggling comes for free as part of the random traversal order of a $rors$, plus CHEETAH's *reset-retry* mechanism. Line 33 of Figure 1 shows that CHEETAH is called many times, with the resulting behaviour logged to the file *experience.dat*. Between each run, CHEETAH *resets* its assumption memory and *retries* its high-level goals for another time. During this later test, if the same $rors$ is accessed, then the random ordering or the $rors$ operator means that the goals may be explored in a different order. This implies that different assumptions may be made before the proof reaches X_j and, hence, X_j may be provable for some subset of CHEETAH's runs.

In the case where ordering must be preserved, CHEETAH supports the operators and, or, any, ors which are the non-random versions of $rand, ror, rany, rors$ respectively. Using an ors operator would set a preference criteria for a set of goals. For example, the goal $X\ ors\ Y\ ors\ Z$ would mean CHEETAH would attempt to prove them all in a left-to-right order, and the assumptions required for X would take precedence over the assumptions required for Y and Z. Using an and operator would set a precise ordering for how goals are proved. For example, a top-down structure chart for a project plan written in JANE would be implemented as follows:

```
r1 if   analysis and design and code and test
   then softwareProject.
r2 if   requirementsCapture and debate and
        decisions and elaborationOfDetails
   then analysis.
```

Given the goal $softwareProject$, CHEETAH would explore this rule-base depth-first, left-to-right; i.e. the and operator ensures that $requirementsCapture$ would occur before $debate$ and both of these would occur before $design$. Another use of explicit orderings in JANE might be to define restrictions on the random walk before it is executed. For example, pollution markers are a method for marking some parts of the requirements temporarily out-of-bounds [13]. In JANE, pollution markers to ignore (e.g.) x and y could be added by assuming the negation of x and y before testing for the goals. Alternatively, in order to conduct a what-if query on e.g. a and b, these assumptions could be declared prior to the goals. This would be implemented by proving the goal $done$ across the following rules:

```
r1 if setup     and goals          then done.
r2 if pollution and whatifs        then setup.
r3 if not x     and not y          then pollution.
r4 if a         and b              then whatifs.
r5 if happy rors content rors rich then goals.
```

operator	X_{cost}	$X_{chances}$
$X =$ $or(Y)$ $ror(Y)$	$first(Y).cost$	$first(Y).chances$
$X =$ $and(Y)$ $rand(Y)$	$\sum_{i=1}^{\|Y\|} cost(Y_i)$	$\prod_{i=1}^{\|Y\|} chances(Y_i)$
$X =$ $rors(Y)$ $ors(Y)$ $rany(Y)$ $any(Y)$	$Z \subseteq Y$ $\sum_{i=1}^{\|Z\|} cost(Z_i)$	$Z \subseteq Y$ $\sum_{i=1}^{\|Z\|} chances(Z_i)$
$X =$ $no(Y)$	$\sum(allY.cost)$	$1 - \sum(allY.chances)$

Figure 2. The SET1 combination rules for $costs$ and $chances$. In this table, (a) the function "$first(Y)$" returns the first proved element in Y; (b) Z is the subset of Y that is proved by the operator; (c) $allX$ finds all solutions to X; (d) the value of $chances$ is always capped to one.

4.3. Open Issues

There are many features of the Figure 1 rule base that are open to debate. For example, what exactly are the precise $cost$s and $chance$s for each rule and how are we to tally them togehter? JANE tallies $cost$s and $chances$ using a customizable set of *combination operators*. This set can easily be changed but this leaves open the question: which operators should be used?

One set of combination rules is shown are the rules known as SET1, shown in Figure 2. The rules for $\{or, ror, and, rand\}$ are simple enough. However, $\{rors, ors, rany, any, no\}$ are more complex, more open to debate. The source of the complexity is that these operators search for multiple solutions within a disjunction. It could be argued that as the amount of evidence increases, the higher the $chances$ but the greater the $cost$ (since evidence collection is expensive). Hence, for $\{rors, ors, rany, any\}$ both $cost$ and $chances$ are summed together.

(Note the absence of a not operator in SET1: JANE applies deMorgan's theorem to convert e.g. $a\ and\ not\ (b\ and\ c)$ to $a = t\ and\ (b = false\ or\ c = false)$. Hence, at runtime, not is never called.)

SET2 is another set of combination rules which is almost the same as SET1 but takes a different stance on how $cost$s are combined. In SET2, the $cost$ of finding multiple solutions within a disjunction (i.e. $\{rors, ors, rany, any\}$) is the $maximum$ of the cost of the proved parts of the disjunction.

In keeping with the whole JANE/CHEETAH/TARZAN approach, if a debate is possible, we should randomly simulate across the space of possibilities, then use induction to check which (if any) of the debate points are key. In the case study shown below, JANE simulated:

- Across the $cost$ and $chances$ range specified in Figure 1.

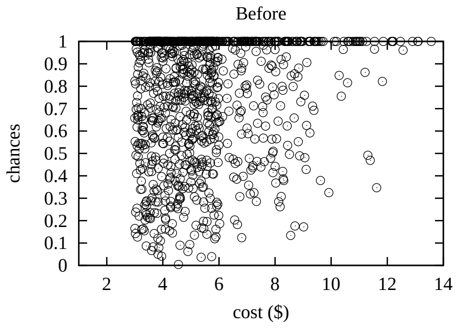

Figure 3. $Cost$ **and** $chances$ **from 1,000.**

		Cost	
		low if $\leq$ \$5	high if $>$ \$5
Chances	low if $\leq$ 0.85	24.7%	24.3%
	high if $>$ 0.85	21.7%	29.4%

Figure 4. Percentage of 1,000 runs that fall into classes that combine low/high $cost$ **and low/high** $chances$**.**

- Using either SET1 or SET2 (picked once for each run).

4.4. TARZAN

TARZAN performs induction over the logs of behaviour seen when the CHEETAH abductive inference engine explores the JANE rules. Recall from Figure 1, when CHEETAH runs JANE, a log of JANE's behaviour is stored in *experience.dat*. TARZAN searches that log looking for the *fewest* number attribute ranges that have the *largest* impact on the overall behaviour of the system.

Figure 3 shows the $cost$ and $chances$ seen in 1,000 proofs of *happy* (as defined on lines 6,7 of Figure 1). Note the wide range of possible outcomes. At this point, a formal RE researcher such as van Lamsweerde [17] could reject JANE-based models as useless. If we cannot restrict this wide range, then JANE-style modeling must be rejected since it cannot generate definite conclusions.

TARZAN's task of restricting this range of behaviour begins by dividing the output into several classes. The classifications shown in Figure 4 were chosen so as to balance the size of the different classes (classes of different sizes can bias an inductive learner). Of these classes, one is clearly inferior (low $chances$, high $cost$), and one is clearly superior (high $chances$, low $cost$). TARZAN's task is to find methods for nudging the system away from inferior and toward superior classes.

The version of TARZAN used in this study collected frequency counts of attribute ranges in the different classes. These counts were expressed as the relative measure:

$$\frac{\text{seen in superior class}}{\text{seen in other class}}$$

The results in Figure 5 shows that numerous $costs$ and $chances$, and assumptions control JANE's behaviour. And, as might be expected, the ors-$cost$-combination issue is somewhat important (witness the presence of $orsCostCombine$ in most of the cells in Figure 5). However, over-riding issues of $costs$, $chances$, and combination rules are assumptions about $likesSweat$. Observe that $likesSweat = false$ (is false) appears 3.5 times more frequently in low $cost$, high $chances$ class than in the high $cost$, high $chances$ class. Hence, to argue less, we could just try one what-if query: what-if we set "$likesSweat = false$"? The effects of that what-if query is shown in the box plots of Figure 6. Also, the variance in the $cost$ is greatly reduced (see Figure 6) and most of the $chances$ are close to one. (this explains why the the right-hand $chances$ "box" in Figure 6 is squashed flat: the 25% to 75% percentile values are all the same).

Note what has been achieved here. Without JANE/CHEETAH/TARZAN, the exact values of the $cost$ and $chances$ values could be endlessly debated. Numerous sub-committees might be formed to make contradictory conclusions about this weight vs that weight. Also, the issue of SET1 vs SET2 could be endlessly debated. Numerous PhD projects could make contradictory mathematical arguments for SET1 vs SET2.

With JANE/CHEETAH/TARZAN, we can argue less. At least in this example, debating the precise values of $cost$ and $chances$, or SET1 vs SET1, is a waste of time. Other factors, such as whether or not we believe in $likeSweat$ out-weighs the details of $cost$ and $chances$ or sum vs $maximum$.

5. Generality

The above example showed one small example of using funnel-based reasoning. In what other domains might this technique work and how well will it scale?

This technique applies in domains where three factors are true: narrow funnels are *frequent*, random search is an *adequate* method of searching for goals, and random search has a *preference* for narrow funnels over wide funnels. There is much evidence that these three factors are true in many domains.

Frequent: Menzies & Cukic discuss the average *shape* of software; i.e. how numerous and how tangled are the pathways inside a piece of software [8]. The overwhelming evidence is that most software relies a small number of frequently used straight pathways. Straight pathways are

Frequency w.r.t. to low $cost$, high $chances$	high $cost$ low $chances$	high $cost$ high $chances$	low $cost$ low $chances$
1.5	r11cost=0.99, r11cost=1.33, r11cost=1.99, r11cost=2.67, r8chances=0.89, orsCostCombine=max, likesSweat=false, swimming=t	r10chances=0, r10chances=0.44, r10chances=0.89, r8chances=0, r8chances=0.44, r8chances=0.89, r8cost=0.9996, r9chances=0, r9chances=0.44, r9chances=0.89, orsCostCombine=max, likesSweat=false, swimming=t	r10chances=0, r11chances=1.33, r11chances=2.22, r11chances=2.66, r11chances=3.11
2.0		r10chances=0, r10chances=0.44, r8chances=0.44, r9chances=0, r9chances=0.44, r9chances=0.89, orsCostCombine=max, likesSweat=false, swimming=t	
2.5		likeSweat=false, swimming=t	
3.0		likesSweat=f, swimming=t	
3.5		likesSweat=f, swimming=t	

Figure 5. Attribute ranges frequency counts seen in the different classes. Only those counts that were very different to the counts seen in the superior class (high $chances$, low $cost$) are shown.

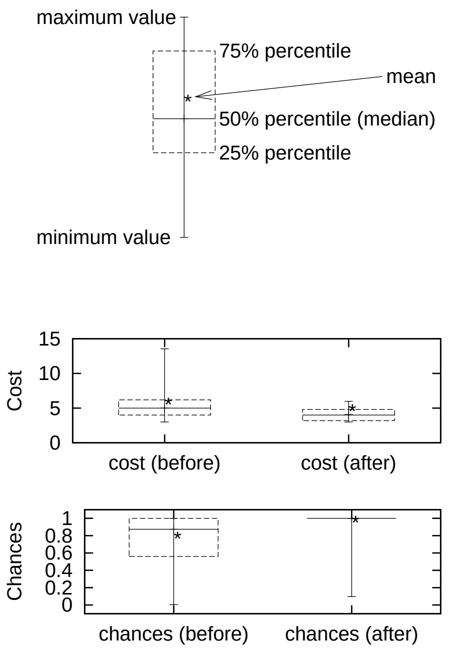

Figure 6. Box plots showing changes in the $cost$s, and $chances$ before and after "what-if likesSweat=false".

```
cmm1 if   okToProceed then reqAuthorized.
cmm2 if   peerReviewOfRequirements rany
          doFormalRequirementsInspections rany
          softwareAssuranceReviewOfRequirements
     then reviews.
cmm3 if   problemReportAndCorrectiveActionSystem
     then reqStable3.
cmm4 if   problemReportAndCorrectiveActionSystem
     then reqCompleteness3.
cmm5 if   implementFormalConfigurationManagement
          rors doFormalReviews.
     then reqStable4.
cmm6 if   doFormalReviews then formalReviews.
cmm7 if  'planAndScheduleIV&Vactivities' then''iv&v'.
cmm8 if   reqAuthorized rand reviews rand
          reqStable3 rand reqStable4 rand iv&v
     then stableRequirements.
cmm9 if   reqAuthorized rand reviews rand
          reqCompleteness3 rand formalReviews
          rand 'iv&v'
     then completeRequirements.
cmm10 if  no stableRequirements rand
          no completeRequirements rand
          invalidRequirements rand
          infeasibleRequirements rand
          unprecedentedRequirements rand
          (largeSize ror complexSystems)
     then requirementsRisk.
```

Figure 7. Some CMM level 2 knowledge in JANE format ($costs$, $chances$, and authors not shown)

prone to funnels since downstream parts of a path depend on the critical assumptions made early in the path.

Preference: Menzies & Singh explore how random search might select between narrow funnels and wide funnels. Based on known distributions of reaching part of a software system, they concluded that a random search is millions of times more likely to use narrow funnels [9].

Adequacy: A huge body of work testifies to the merits of random search, even for very hard tasks such as searching an argument space. For example, random search methods are very effective for scheduling problems and can solve hard and larger planning problems many times faster than traditional methods such as a systematic Davis-Putnam procedure [4]. This work, plus the Menzies & Micheal experiments [11] suggests that random search is both an adequate and fast inference procedure for argument spaces.

In order to test this generality argument, JANE/CHEETAH/TARZAN are being applied to several domains. Figure 7 shows part of a JANE rule base describing CMM level 2 best software practices. In other work, a model of computer hardware choice for a university department is being developed. Also, a translator from JPL's AART tool [2] into JANE is under construction to support argument reduction in the early life cycle of NASA software developments.

6. Discussion

This approach *reduces* the number of total arguments to a small number of key arguments. However, this approach does not *resolve* or *remove* those key arguments. This is quite deliberate. Arguments will not and should not go away. To be human, to be an expert, to be an individual, implies that you often take a different stance to your peers. Arguing such different stances generates heat and light and insights into a domain that may remain hidden otherwise. Arguments are an important part of viewpoints-based RE and we should orchestrate the negotiation between stakeholders by exploring their disputes.

Nevertheless, we cannot endorse arguments unless we also show how to prevent the unconstrained arguments that can stall RE. Stakeholders must be free to argue about anything they like. But in a resource-bounded situation (e.g. any software development process), we can argue less by sorting our arguments according to their information gain. Such "most informative arguments" can be quickly found in JANE rules via CHEETAH's randomized abduction followed by TARZAN's induction.

Acknowledgements

Partial funding for this work was provided through the NASA IV&V facility by the NASA Office of Safety and Mission Assurance. Helen Burgess provided invaluable editorial advice for this article.

References

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

[2] M. Feather, H. In, J. Kiper, J. Kurtz, and T. Menzies. First contract: Better, earlier decisions for software projects. In *Submitted to Proceedings of the Fifth International Symposium on Requirements Engineering*, 2001. Available from `http://tim.menzies.com/pdf/01first.pdf`.

[3] A. Kakas, R. Kowalski, and F. Toni. The role of abduction in logic programming. In C. H. D.M. Gabbay and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, pages 235–324. Oxford University Press, 1998.

[4] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, Aug. 4–8 1996. AAAI Press / MIT Press. Available from `http://www.cc.gatech.edu/~jimmyd/summaries/kautz1996.ps`.

[5] G. Kotonya and I. Sommerville. Viewpoints for requirements definition. *IEE Software Engineering Journal*, 7:375–387, 1992.

[6] R. Madachy. Heuristic risk assessment using cost factors. *IEEE Software*, 14(3):51–59, May 1997.

[7] T. Menzies and P. Compton. Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models. *Artificial Intelligence in Medicine*, 10:145–175, 1997. Available from `http://tim.menzies.com/pdf/96aim.pdf`.

[8] T. Menzies and B. Cukic. When to test less. *IEEE Software*, 17(5):107–112, 2000. Available from `http://tim.menzies.com/pdf/00iesoft.pdf`.

[9] T. Menzies and B. Cukic. Average case coverage for validation of ai systems. In *AAAI Stanford Spring Symposium on Model-based Validation of AI Systems*, 2001. Available from `http://tim.menzies.com/pdf/00validint.pdf`.

[10] T. Menzies, S. Easterbrook, B. Nuseibeh, and S. Waugh. An empirical investigation of multiple viewpoint reasoning in requirements engineering. In *RE '99*, 1999. Available from `http://tim.menzies.com/pdf/99re.pdf`.

[11] T. Menzies and C. Michael. Fewer slices of pie: Optimising mutation testing via abduction. In *SEKE '99, June 17-19, Kaiserslautern, Germany. Available from* `http://tim.menzies.com/pdf/99seke.pdf`, 1999.

[12] T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from `http://tim.menzies.com/pdf/00ase.pdf`.

[13] B. Nuseibeh. To be *and* not to be: On managing inconsistency in software development. In *Proceedings of 8th International Workshop on Software Specification and Design (IWSSD-8)*, pages 164–169. IEEE CS Press., 1997.

[14] B. Nuseibeh, J. Kramer, and A. C. W. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.

[15] D. T. Ross. Applications and extensions of sadt. *IEEE Computer*, 18:25–34, 1985.

[16] R. Stamper. Social norms in requirements analysis: an outline of measur. In M. Jirotka and J. A. Goguen, editors, *Requirements Engineering: Social and Technical Issues*, pages 107–139. Academic Press, 1994.

[17] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceedings ICSE2000, Limmerick, Ireland*, pages 5–19, 2000.