# On the Value of Stochastic Abduction (if you fix everything, you lose fixes for everything else)

Tim Menzies,
Oussama Elrawas, Dan Baker
Lane Dept. CSEE West Virginia University, USA
tim@menzies.us.
oelrawas@mix.wvu.edu
danielryanbaker@gmail.com

Jairus Hihn
Karen Lum
Jet Propulsion Laboratory
California, USA
jairus.hihn@jpl.nasa.gov
karen.t.lum@jpl.nasa.gov

## 1. INTRODUCTION

Back in the 1980s, the model-based diagnosis (MBD) community explored qualitative representations [42]. Since they are not overly-specific, such representations can be quickly collected in a new domain. Indeed, in domains where information is limited, there may be no alternative to qualitative representations since their is not enough information to build precise quantitative theories.

Qualitative theories are inherently nondeterministic [18] but, argued MBD researchers, they can generate a wider range of options for diagnosis and repair. Creative solutions can sometimes be found in larger space of possible qualitative behaviors than in the tighter space of precise quantitative behaviors. To put that another way:

*If you fix everything, you lose fixes for everything else.*

Previosuly, we have explored tools for searching models containing uncertainities [19]. Standard quantitative sensitivity analysis [16] may be inappropriate for non-linear theories or theories containing inconsistency predicates. Motivated by work on qualitative simulation and non-montonic logics, we defined those tools in terms of logical abduction [20]. Given a theory $T$ and a goal $G$, the abductive problem is to find assumptions $A$ that lead to the goal, without causing inconsistencies ($\perp$):

$$T \wedge A \quad \vdash \quad G \tag{1}$$

$$T \wedge A \quad \nvdash \quad \perp \tag{2}$$

Informally, equation (1) is saying "do something" and equation (2) is saying "but don't do dumb things".

In the general case, equations (1) and (2) leads to the generation of multiple worlds $W$ of belief where each world is characterized by incompatible assumptions (and no world is a subset of a large world w.r.t. size). Worlds are also called *extensions* in Reiter's default logic [39], *scenarios* in Poole's THEORIST system [35], or envisionments in the ATMS [8].

In our HT4 abductive algorithm [19], a preference predicate $best$ selects the subset of preferred worlds. Elsewhere [21], we have argued that a wide-range of knowledge-level tasks can be implemented via different $best$ predicates. For example:

- Diagnosis prefers worlds that contain the greatest number of faults, with fewest root causes (ideally, one).
- Planning prefers worlds with the least cost to traverse;
- Explanation prefers the worlds with the largest intersection with concepts the user has already learned;
- Monitoring requires caching the generated worlds, pruning any world that contradicts any incoming data. In this framework, at any time, the remaining worlds contain the set of possible remaining plans.
- For other applications of abduction, see [14,21,22,25,26,28, 34–38,41,43].

That is, once you have a good way to explore uncertainty, you have an engine that can handle a very large number of tasks

While the generality of the above framework was compelling, that research floundered on the computational cost of abduction. With no inconsistency checks, abduction takes linear time for propositional theories [9]. However, the inconsistency check makes abduction computation NP-hard [6]. Experimentally, HT4's runtimes proved to be exponential on model size and never scaled to anything larger than medium-sized models.

Recent success with stochastic methods have made us revisit abduction. We offer one case study here where a heuristic preprocessor reduces a potentially exponential time process to linear time. The results are preliminary (from just one set of models) but they are promising enough to motivate further exploration.

## 2. "COLLARS" VARIABLES

Surprisingly, our abductive reasoner produce surprisingly few worlds. Or, to be more precise, it produces worlds that contain similar sets of goals. For example, in one study, the number of goals reached in any randomly selected world was nearly the same as the maximum number of goals found in any world [28].

We first suspected this was a quirk of the model used in that study. However, it turns out to be a general result. A repeated conclusion in AI is that the behavior of a large system is determined by a very small number of key variables, which we call the *collar* [30] variables. Collars have been discovered and rediscovered in AI many times, and given different names including *variable subset selection* [17], *narrows* [2], *master-variables* [7], *back doors* [44], minimal envisionments [8], just to name a few. Note that if the collar contains just a few variables, then few worlds will be generated: just one for every consistent setting to the collar variables.

Analogous results have been reported in software engineering literature. Observe that if a system is controlled by its collar, and the collar variables are few in number, then a stochastic sampling

This research was conducted at West Virginia University, Univerity of Southern California, and NASA's Jet Propulsion Laboratory under a NASA sub-contract. 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.

IWLU'07 Atlanta, Georgia, USA

of the system will soon *saturate*; i.e. no new states will be found after exploring the combinations of the settings to the collar variables, Saturation has been observed in the knowledge engineering and software engineering literature:

- Horgan & Mathur [13] document systems were most program paths get exercised early with little further improvement as testing continues.
- With Owen & Cukic [29] and Goa et.al. [12], we have shown that the number of new states found in a stochastic search of a formal model quickly saturates.
- Mutation testers often report that a small sample of their mutations[1] perform just as well as a much much larger number of mutations [1, 5, 32, 45].
- Druzdzel [10] studied a diagnosis application for monitoring patients in intensive care. Although the software had 525,312 possible internal states, the application reached few of them at runtime: one of the states occurred 52 percent of the time, and 49 states appeared 91 percent of the time.

Not only is saturation a repeated result, it can be shown mathematically that it is the expected behavior of any system of variables [10, 31]. Druzdel observes that if software has $n$ variables, each with its own assignment probability distribution of $p_i$, then the probability that software will fall into a particular state is

$$p = p_1 p_2 p_3 ... p_n = \prod_{i=1}^{n} p_i.$$

By taking logs of both sides, this equation becomes

$$\ln p = \ln \prod_{i=1}^{n} p_i = \sum_{i=1}^{n} \ln p_i.$$

The asymptotic behavior of such a sum of random variables is addressed by the central limit theorem. In the case where we know very little about software, $p_i$ is uniform and many states are possible. However, the *more* we know about software the more varied the individual distributions. Given enough variance in the individual priors and conditional probabilities or $p_i$, the expected case is that the frequency with which we reach states will exhibit a log-normal distribution; i.e. a small fraction of states can be expected to cover a large portion of the total probability space; and the remaining states have practically negligible probability [10]. If the reachable states are few, then the deltas between them (the collar settings) are also few.

# 3. EXPLOITING THE COLLARS

In summary, while abduction is *theoretically* slow, it need not be slow *in practice*. The trick is to reverse the usual abductive procedure of, e.g. the ATMS [8]. The ATMS was designed as a working memory that stores the conclusions of a separate inference engine. Inferences are passed to the ATMS along with a *justification* (a conjunction) for each inference. These justifications were then woven into a subsumption network, the roots of which are the assumptions that select the different set of reachable beliefs (and we would call those root assumptions the collar settings). In that approach, the collars are found after the inference.

$$inference \rightarrow collars \rightarrow worlds$$

An alternative is to find the collars *before* the inference.

$$peek \rightarrow collars \rightarrow worlds \rightarrow inference \quad (3)$$

[1]A *mutant* of a program is a syntactically valid but randomly selected variation to a program; e.g. swapping all plus signs to a minus sign.

The results of §2 suggest that a stochastic sampling method could *peek* at the reachable states (via some Monte Carlo simulation). If each of the $n \in N$ peeks is scored by some domain-specific predicate, then the collars can then be identified as the variables with *very different frequency in the high scoring samples than in the low scoring samples*.

One world exists for each consistent set of settings to the collars. Since worlds are internally consistent, inference within them can be very fast. Williams et.al report that setting the collar variables (which they call *backbones* [44]) can reduce exponential time inferences to polynomial time.

If inference takes time $k$, then after $peek$ing, Equation 3 takes time $O(k * 2^C)$ where $C$ is the number of ranges the collars. Since the number $C$ is often much less than the total number of ranges in the entire theory, this is a win. However, it is still an exponential time process.

We show below that, given a certain kind of $best$ predicate, it is possible to reduce Equation 3 to $O(k)$. If $peek$ing sorts the collar settings from best to worst, the worlds and inference can be explored using a greedy search that stops when the next world scores little better than the previous one.

# 4. CASE STUDY

This case study is based on the STAR algorithm. A summary of STAR is offered below. For full details, see [27]. STAR's current implementation explores three software process models:

- The COQUALMO software defect predictor [4, p254-268];
- The COCOMO software effort predictor [4, p29-57];
- The THREAT predictor for software project effort & schedule overrun [4, 284-291].

In standard practice, these models are tuned to local data. Unfortunately, the data required for such local tuning is difficult to obtain, especially across multiple organizations [24]. This is due to the business sensitivity associated with the data as well as differences in how the metrics are defined, collected and archived. In many cases the required data has not been archived at all. For example, after two years we were only able to add 7 records to our NASA wide software cost metrics repository.

After decades of research, the space of possible local tunings is well defined. The above three models contain relationships between (e.g.) software process decisions and defect removal. These are linear equations, defined by a slope $m$, and features learned from the domain $\{a, b\}$. Varying $\{a, b, m\}$ across the space of known tunings will sample the space of possible behaviors.

Another source of variability in the model behavior are the input features. Figure 1 shows the range of those inputs for the four case studies explored in this paper:

- OSP is the GNC component of NASA's 1990s *Orbital Space Plane* experiment;
- OPS2 is a later version of OSP;
- Flight and ground systems reflect typical ranges seen at NASA's Jet Propulsion Laboratory.

The inputs of Figure 1 are explained in Figure 2. All these features can range over $1 \leq x \leq 6$.The input features either *ranges* (a space of options) or are fixed to one *value*; e.g. line one of Figure 1 says:

- the $prec$ input is constrained to $prec \in \{1, 2\}$
- the $data$ input is fixed to $data = 3$.

Figure 1 does not mention all the possible inputs. For example, COQUALMO has inputs for use of *automated analysis*, *peer*

| | ranges | | | values | |
|---|---|---|---|---|---|
| project | feature | low | high | feature | setting |
| OSP: Orbital space plane | prec | 1 | 2 | data | 3 |
| | flex | 2 | 5 | pvol | 2 |
| | resl | 1 | 3 | rely | 5 |
| | team | 2 | 3 | pcap | 3 |
| | pmat | 1 | 4 | plex | 3 |
| | stor | 3 | 5 | site | 3 |
| | ruse | 2 | 4 | | |
| | docu | 2 | 4 | | |
| | acap | 2 | 3 | | |
| | pcon | 2 | 3 | | |
| | apex | 2 | 3 | | |
| | ltex | 2 | 4 | | |
| | tool | 2 | 3 | | |
| | sced | 1 | 3 | | |
| | cplx | 5 | 6 | | |
| | KSLOC | 75 | 125 | | |
| OSP2 | prec | 3 | 5 | flex | 3 |
| | pmat | 4 | 5 | resl | 4 |
| | docu | 3 | 4 | team | 3 |
| | ltex | 2 | 5 | time | 3 |
| | sced | 2 | 4 | stor | 3 |
| | KSLOC | 75 | 125 | data | 4 |
| | | | | pvol | 3 |
| | | | | ruse | 4 |
| | | | | rely | 5 |
| | | | | acap | 4 |
| | | | | pcap | 3 |
| | | | | pcon | 3 |
| | | | | apex | 4 |
| | | | | plex | 4 |
| | | | | tool | 5 |
| | | | | cplx | 4 |
| | | | | site | 6 |
| JPL flight software | rely | 3 | 5 | tool | 2 |
| | data | 2 | 3 | sced | 3 |
| | cplx | 3 | 6 | | |
| | time | 3 | 4 | | |
| | stor | 3 | 4 | | |
| | acap | 3 | 5 | | |
| | apex | 2 | 5 | | |
| | pcap | 3 | 5 | | |
| | plex | 1 | 4 | | |
| | ltex | 1 | 4 | | |
| | pmat | 2 | 3 | | |
| | KSLOC | 7 | 418 | | |
| JPL ground software | rely | 1 | 4 | tool | 2 |
| | data | 2 | 3 | sced | 3 |
| | cplx | 1 | 4 | | |
| | time | 3 | 4 | | |
| | stor | 3 | 4 | | |
| | acap | 3 | 5 | | |
| | apex | 2 | 5 | | |
| | pcap | 3 | 5 | | |
| | plex | 1 | 4 | | |
| | ltex | 1 | 4 | | |
| | pmat | 2 | 3 | | |
| | KSLOC | 11 | 392 | | |

**Figure 1: Four case studies.**

*reviews*, and *execution-based testing tools*. For all inputs not mentioned in Figure 1, values are picked at random $1 \leq x \leq 6$.

STAR's simulated annealer explores the possible inputs and $\{m, a, b\}$ values looking for constraints to these ranges that minimizes effort ($Ef$) and defects ($De$) and threats ($Th$). More precisely, the simulated annealer tries to minimize the *energy function*:

$$E = \left(\sqrt{\overline{Ef}^2 + \overline{De}^2 + \overline{Th}^2}\right) / \sqrt{3} \quad (4)$$

Here, $\overline{x}$ is a normalized value $0 \leq \frac{x - min(x)}{max(x) - min(x)} \leq 1$. Hence, our energy ranges $0 \leq E \leq 1$ and *lower* energies are *better*.

The simulated annealing creates a log showing what attribute are associated with higher or lower energies. STAR's forward select algorithm sorts those ranges in increasing order of associated energy.

| | |
|---|---|
| scale factors (exponentially decrease effort) | prec: have we done this before?<br>flex: development flexibility<br>resl: any risk resolution activities?<br>team: team cohesion<br>pmat: process maturity |
| upper (linearly decrease effort) | acap: analyst capability<br>pcap: programmer capability<br>pcon: programmer continuity<br>aexp: analyst experience<br>pexp: programmer experience<br>ltex: language and tool experience<br>tool: tool use<br>site: multiple site development<br>sced: length of schedule |
| lower (linearly increase effort) | rely: required reliability<br>data: secondary memory storage requirements<br>cplx: program complexity<br>ruse: software reuse<br>docu: documentation requirements<br>time: runtime pressure<br>stor: main memory requirements<br>pvol: platform volatility |

**Figure 2: The COCOMO II scale factors and effort multipliers.**

Then, for the controllable input features $C$, STAR imposes the top $1 \leq X \leq C$ ranked ranges and re-runs COCOMO, COQUALMO, and THREATS 1000 times. The median energy seen in those 1000 runs is collected. Let $Min$ be some value $1 \leq Min \leq C$ associated with minimum median energy seen in the above forward select experiment. A *policy* are all the ranges see in $1..Min$.

It is insightful to compare the effects of STAR's policies with more traditional methods. SCAT and 2CEE are effort estimation tools developed in-house by the authors at JPL. Both SCAT and 2CEE accept inputs like Figure 1 and output a range of effort estimates. SCAT uses fixed $\{a, b\}$ values (derived from historical logs of JPL systems) while 2CEE derives ranges for $\{a, b\}$ based on historical data. Neither of these models vary $m$; rather, they reuse the $m$ values proposed from Boehm [4].

Figure 3 compares the median effort estimate predictions found by in SCAT, 2CEE, and STAR (defect and threat estimates are not shown since SCAT and 2CEE only contain effort models). In

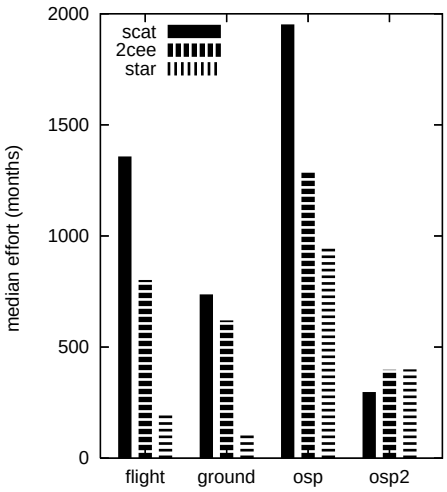


**Figure 3: Effort estimations from two standard methods (SCAT, 2CEE) and STAR.**

| id | features | relative weight |
|---|---|---|
| 1 | Personnel/team capability | 3.53 |
| 2 | Product complexity | 2.38 |
| 3 | Time constraint | 1.63 |
| 4 | Required software reliability | 1.54 |
| 5 | Multi-site development | 1.53 |
| 6 | Doc. match to life cycle | 1.52 |
| 7 | Personnel continuity | 1.51 |
| 8 | Applications experience | 1.51 |
| 9 | Use of software tools | 1.50 |
| 10 | Platform volatility | 1.49 |
| 11 | Storage constraint | 1.46 |
| 12 | Process maturity | 1.43 |
| 13 | Language & tools experience | 1.43 |
| 14 | Required dev. schedule | 1.43 |
| 15 | Data base size | 1.42 |
| 16 | Platform experience | 1.40 |
| 17 | Arch. & risk resolution | 1.39 |
| 18 | Precedentedness | 1.33 |
| 19 | Developed for reuse | 1.31 |
| 20 | Team cohesion | 1.29 |
| 21 | Development mode | 1.32 |
| 22 | Development flexibility | 1.26 |

**Figure 4: Relative effects on development effort. Data from a regression analysis of 161 projects [3].**

one case, all the estimates were similar (OSP2) but for the others, STAR's forward select found input ranges that greatly reduced the developed effort: e.g. for ground systems, STAR's preferred inputs yielded estimates that were 6.4 and 5.4 times smaller than SCAT or 2CEE, respectively.

When we first saw these results, we suspected a bug. Then we realized that unlike SCAT and 2CEE, STAR doesn't just make estimates. STAR also tries to minimize those estimates. The impact of altering the input ranges can be quite dramatic. Figure 4 shows the known relative productivity effects of changing project features. The product of those productivity effects shows the total impact of decisions regarding inputs on effort estimation:

$$11{,}022.4 = 3.53 * 2.38 * 1.63 * 1.54 * 1.53 * 1.52 * 1.51 * 1.51 * 1.5 * 1.49 * 1.46 * 1.43 * 1.43 * 1.43 * 1.42 * 1.4 * 1.39 * 1.33 * 1.31 * 1.29 * 1.32 * 1.26.$$

That is, if projects were free to do anything at all, the process decisions of Figure 2 can change total effort by three orders of magnitude. STAR was only able to reduce the cost of building (e.g.) JPL's ground systems by a factor of $\approx 6$ since Figure 1 constrained the space of possible choices.

# 5. DISCUSSION

Figure 3 illustrates the same effect reported in the introduction by the model-based diagnosis community. Creative solutions can sometimes be found in a large space of possible behaviors (STAR) than in a tighter space of more precisely defined behaviors (SCAT, 2CEE).

These results show how the premature fixing of the options can prevent an analysis from proposing novel fixes. For example, observe that:

- In Figure 1, OSP2's fixed values are more extensive that the other three cases.
- In Figure 3, STAR reduced the effort by the least amount of the other case studies.

We speculate that if STAR had been applied earlier in the life cycle of OSP2, then a different and cheaper project could have been selected.

STAR is an abductive inference engine. The theory $T$ is the combined COCOMO, COQUALMO, THREAT models. The assumptions $A$ of Equations 1 and 2 are the *policies* found by STAR's forward select. The $best$ predicate that selects the preferred worlds is the energy function of Equation 4 that ranks the ranges. If $k$ is the time required to run COCOMO,COQUALMO, and THREAT, then STAR runs in $O(k)$ time using its greedy search down the ranked list of ranges. Hence, STAR runs fast (under 10 seconds for the simulated annealing and forward select for each of the test cases in Figure 1).

Our previous implementation of this approach was called the TAR3 *treatment learner* [23]. A treatment learner finds a minimal contrast set that distinguishes desired from undesired outcomes. With Feather, we have used treatment learning [11] to find the key decisions within requirements models. Treatment learning assumes that there is very little control of the device generating the training data. STAR was designed assuming a fine-grained integration between the data generator and the learner.

There is much to recommend STAR over TAR3. STAR generated all our case study results in less than a minute. A similar range ranking, by TAR3, would require 20 to 30 minutes to generate feature range rankings. whose $peeks$ rank the rank collar settings in a best to worst order.

STAR looks nothing like a logical theorem prover. In this case study, for example, STAR watched over a procedural system and not a system represented as a set of clauses. As observed by Kakas et.al. [15] abduction is a *knowledge-level* inference procedure [33] which, under the hood, can be implemented by any number of symbol-level heuristic methods.

# 6. CONCLUSION

Traditionally, uncertainty is viewed as a bad thing. To be sure, there are times when unceratinty is dangerous. The software controller of the ascent stage of a manned spacecraft should be a deterministic algorithm with guaranteed performance properties. Using anything else at this stage of the mission is as crazy as *not* using (say) random search to assist in on-board diagnosis when the craft is (a) in deep space and (b) in deep trouble and (c) it takes too long to ask for help from ground control.

Previously, we have explored uncertainty analysis in an abductive framework but that work floundered on the compuational cost of abduction. Recent success with a stochastic pre-processor to find the "collar" variables suggest that we can return to researching abduction as a general framework for software engineering and knowledge engineering.

For future work, we need to check the generality of the STAR algorithm. COCOMO, COQUALMO, and THREAT are simple models and a more general STAR framework might require more intricate forward selection methods or more some alternative to simulated annealing. For example, Kakas has explored linear optimization as a symbol-level engine for abduction. Another candidate method might be the cross-entropy method [40].

# 7. REFERENCES


[1] A. Acree. *On Mutations*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1980.

[2] S. Amarel. Program synthesis as a theory formation task: Problem representations and solution methods. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach: Volume II*, pages 499–569. Kaufmann, Los Altos, CA, 1986.

[3] B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from

http://www.computer.org/certification/beta/Boehm_Safe.pdf.

[4] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[5] T. Budd. *Mutation analysis of programs test data*. PhD thesis, Yale University, 1980.

[6] T. Bylander, D. Allemang, M. Tanner, and J. Josephson. The Computational Complexity of Abduction. *Artificial Intelligence*, 49:25–60, 1991.

[7] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.

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

[9] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

[10] M. Druzdzel. Some properties of joint probability distributions. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pages 187–194, 1994. Available from http://www.pitt.edu/AFShome/d/r/druzdzel/public/html/abstracts/uai94.ht%ml.

[11] M. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002. Available from http://menzies.us/pdf/02re02.pdf.

[12] J. Gao, M. Heimdahl, D. Owen, and T. Menzies. On the distribution of property violations in formal models: An initial study. In *COMPSAC '06*, 2006. Available from http://menzies.us/pdf/06compsac.pdf.

[13] J. Horgan and A. Mathur. Software testing and reliability. In M. R. Lyu, editor, *The Handbook of Software Reliability Engineering*, pages 531–565, McGraw-Hill, 1996.

[14] 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.

[15] A. C. Kakas, B. V. Nuffelen, and M. Denecker. A-system: Problem solving through abduction. In *IJCAI*, pages 591–596, 2001.

[16] J. Kliijnen. Sensitivity analysis and related analyses: a survey of statistical techniques. *Journal Statistical Computation and Simulation*, 57(1–4):111–142, 19987.

[17] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[18] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:229–338, 1986.

[19] T. Menzies. *Principles for Generalised Testing of Knowledge Bases*. PhD thesis, University of New South Wales, 1995. Ph.D. thesis. Available from http://menzies.us/pdf/95thesis.pdf.

[20] T. Menzies. Situated Semantics is a Side-Effect of the Computational Complexity of Abduction. In *Australian Cognitive Science Society, 3rd Conference*, 1995. Available from http://menzies.us/pdf/cogsci95.pdf.

[21] T. Menzies. Applications of abduction: Knowledge level modeling. *International Journal of Human Computer Studies*, 45:305–355, 1996. Available from http://menzies.us/pdf/96abkl.pdf.

[22] T. Menzies. Applications of abduction: A unified framework for software and knowledge engineering. Asian-Pacific Workshop on Intelligent Software Engineering, 1998. Available from http://menzies.us/pdf/98apwise.pdf.

[23] T. Menzies. $21^{st}$ century ai: proud, not smug. *IEEE Intelligent Systems*, 2003. Available from http://menzies.us/pdf/03aipride.pdf.

[24] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, November 2006. Available from http://menzies.us/pdf/06coseekmo.pdf.

[25] T. Menzies, R. Cohen, S. Waugh, and S. Goss. Applications of abduction: Testing very long qualitative simulations. *IEEE Transactions of Data and Knowledge Engineering*, pages 1362–1375, November/December 2003. Available from http://menzies.us/pdf/97iedge.pdf.

[26] 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://menzies.us/pdf/96aim.pdf.

[27] T. Menzies, O. Elwaras, J. Hihn, F. n. B. B. M, and R. Madachy. The business case for automated software engineerng. In *IEEE ASE*, 2007. Available from http://menzies.us/pdf/07casease-v0.pdf.

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

[29] T. Menzies, D. Owen, and B. Cukic. Saturation effects in testing of formal models. In *ISSRE 2002*, 2002. Available from http://menzies.us/pdf/02sat.pdf.

[30] T. Menzies and J. Richardson. Making sense of requirements, sooner. *IEEE Computer*, October 2006. Available from http://menzies.us/pdf/06qrre.pdf.

[31] T. Menzies and H. Singh. Many maybes mean (mostly) the same thing. In M. Madravio, editor, *Soft Computing in Software Engineering*. Springer-Verlag, 2003. Available from http://menzies.us/pdf/03maybe.pdf.

[32] C. Michael. On the uniformity of error propagation in software. In *Proceedings of the 12th Annual Confererence on Computer Assurance (COMPASS '97) Gaithersburg, MD*, 1997.

[33] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87–127, 1982.

[34] P. O'Rourke. Working notes of the 1990 spring symposium on automated abduction. Technical Report 90-32, University of California, Irvine, CA., 1990. September 27, 1990.

[35] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[36] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[37] H. Pople. On the mechanization of abductive logic. In *IJCAI '73*, pages 147–152, 1973.

[38] J. Reggia. Abductive inference. In *Proceedigns of the Expert Systems in Government Symposium*, pages 484–489, 1985.

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

[40] R. Rubinstein and D. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, 2004.

[41] A. Russo. On the use of logical abduction in software engineering. In C. Chung, editor, *Handbook of Software and Knowledge Engineering*, volume 1, 2001.

[42] B. Williams and J. DeKleer. Qualitative reasoning about physical systems: a return to roots. *Artificial Intelligence*, 51:1–9, 1991.

[43] C. P. Williams. Analytic Abduction from Qualitative Simulation. In B. Faltings and P. Struss, editors, *Recent Advances in Qualitative Physics*, pages 435–450. The MIT Press, 1992.

[44] R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI 2003*, 2003. http://www.cs.cornell.edu/gomes/FILES/backdoors.pdf.

[45] W. Wong and A. Mathur. Reducing the cost of mutation testing: An empirical study. *The Journal of Systems and Software*, 31(3):185–196, December 1995.