Beyond Data Mining; Towards “Idea Engineering”

Tim Menzies
CSEE, WVU, USA
tim@menzies.us

***Abstract*—SE data mining tools can be reconfigured to define and explore the space of decisions made by a community.**

***Index Terms*—Data mining, software engineering, artificial intelligence**

INTRODUCTION

In his seminal “knowledge-level” keynote address to the 1980 American Association of Aritifical Intelligence, Allen Newell asked the following question: “What is knowledge?” [30]. Newell’s answer was to define a knowledge level of goals, actions, and a principle of rationality: “If an agent has knowledge that one it its actions will lead to one of its goals, then the agent will select that action”. Newell took care to separate his “knowledge-level” from an under-lying “symbol-level” that may contain logic, frames, semantic nets, or even procedural code. His challenge to the AI community at that time was to raise their thinking above the symbol-level, to look beyond the trivia of their lower-level tools, and to look towards a higher-level of generality.

It is the contention of this paper that the SE data mining community should find its own knowledge level; i.e that:

- All our current data mining tools are actually low-level primitives in a higher-level process that I will call “idea engineering”.
- That we need to look beyond and above those primitives in order to support the kind of group think that is most common in the mashed-up modern wired world.

The motivation for this paper is a sense of impatience with the SE data mining community. Last century, it was not known if software projects contained sufficient structure to support data mining, though some preliminary results by Porter were encouraging [35]. Now, we know better. Many different kinds of artifacts from software projects contain a signal that can be revealed via data mining (for a partial list of those artifacts, see Figure 1). I assert that that it is now well-established that data mining models can be built from software projects artifacts. So it is now time to move on to “what’s next?”.

Stepney et al. [38] advise that an ideal research roadmap “decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails”. Hence I propose the following progression that can refocus our exist tools and talent into this new and novel area of ‘idea engineering”.

To appear. RAISE’13: 2nd International NSF sponsored Workshop on Realizing Artificial Intelligence Synergies in Software Engineering

According to my proposed progression, we are now leaving the age of *algorithm tuning* and entering the age of *landscape mining*. After that, we should move to the era of *decision systems* and finally to *discussion systems*.

$$\overbrace{\begin{array}{c}algorithm\\tuning\end{array}}^{past} \rightarrow \overbrace{\begin{array}{c}landscape\\mining\end{array}}^{now} \rightarrow \underbrace{\begin{array}{c}decision\\systems\end{array}}_{next} \rightarrow \underbrace{\begin{array}{c}discussion\\systems\end{array}}_{future}$$

As shown below, this progression can use the current skills of the SE data mining community while still stepping us towards some distant grand goal. That is, with a little refactoring, the SE data mining community has the tools and talents that can take it to the next level of research.

This rest of this paper introduces Idea Engineering and has one section for algorithm tuning, landscape mining, decision systems, and discussion systems.

ALGORITHM TUNING

It turns out that, at least is the field of SE data mining , building decision systems is somewhat of a radical idea. To see why, we need a little history. While it is rarely stated, the original premise of SE data mining was that predictions from data mining should guide software management. That is, once upon a time, the aim of *predictions* were *decisions*. Sadly, that original aim seems to be forgotten. Too many researchers in that field are stuck in a rut, just publishing *algorithm tuning* papers about L learners applied to D data sets and evaluated via some $M * N$ cross-validation study.

- Apps store data [15];
- Process data which can predict overall project effort [20];
- Process models showing effective project changes [27], [37];
- Operating system logs that predict software power consumption [17];
- Natural language requirements documents which can be text-mined to find connections between program components [16];
- XML descriptions of design patterns that can be used to recommend particular designs [33];
- Email lists that show human networks inside software teams [2];
- Execution traces that show normal interface usage patterns [12];
- Bug databases that can generate defect predictors to guide inspection teams to where the code is most likely to fail [23], [26], [32], [35].

Fig. 1. Data mining can find signals in many SE project artifacts.

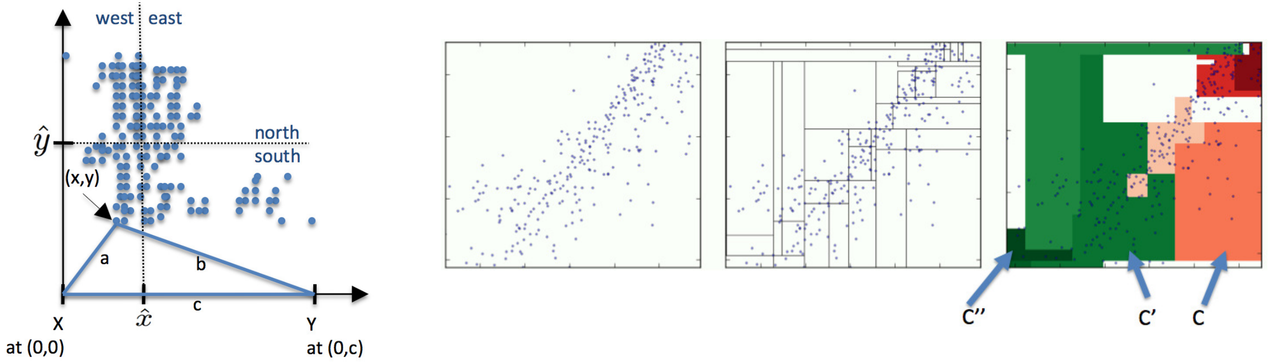

Figure 2.a: *all data in 2 dimensions found by FASTMAP.* **Fig 2.b**: *raw data* **Fig 2.c:** *leaf clusters* **Fig 2.d:** *after griding*

Fig. 2. Each dot is a 18-dimensional instance from an effort estimation data set from the PROMISE repository. The dots are placed onto into two dimensions using the FASTMAP heuristic. Each dot has distance a from the origin and b from the most remote point. The median point on the x and y axis are $\hat{x}$ and $\hat{y}$, respectively. IDEA recurses on each quadrant to generate grids. Leaf pruning then combines the smaller clusters into the colored regions shown in Figure 2.d. Dark green clusters might contain software projects with lowest effort while a dark red cluster holds projects with worst defects.

Trying every learner on every data set is not very insightful. Many SE data sets are a shallow well whose information can be thoroughly extracted by relatively simple methods. My students have found SE defect data sets with 1100 examples that can be reduced to 40 without damaging the model learned from that data [18]. For such data, it may be a waste of time to try the latest and greatest most complex learner. Hall et al. [13] and Dejaeger [9] report that for effort estimation and defect prediction, simpler data miners do just as well, or better than more elaborate ones.

$D{*}L{*}M{*}N$ results from algorithm tuning are problematic since they are highly unstable. No learner is best for all data sets [4] since data can change over time, making prior results outdated [39]. Hence, many researchers now explore "local learners" that eschew single global conclusions in favor of more context-dependent conclusions [1], [22], [36].

Lastly, another issue with $D * L * M * N$-style algorithm tuning research is that it is often driven by the data available to particular researchers, rather than an over-arching vision of the field. Such research is "driven by opportunities, not issues" (a phrase taken from the seminar outcome slides of the 2010 Dagstuhl seminar on New Frontiers for Empirical SE). Surely, as a research community, we should explore issues that are general to more than just the next data set we happen to stumble across.

Landscape Mining

One way to characterize algorithm tuning is "leap before you look"; i.e. before considering the data, throw it at a data miner then reflect on what models are generated by the learner. An opposite approach would be to "look before your leap"; i.e. before running (say) a classifier, try to understand the space of possible models.

Landscape mining is a method of looking before leaping in with data miners and is illustrated in Figure 2. Here, the N-dimensions of some data are clustered into a lower dimensional space. Each cluster is then colored red to green indicating "feared" to "envied"; i.e. a dark green cluster might contain software projects with lowest effort while a dark red cluster holds projects with worst defects.

Our IDEA algorithm [5], [21], [22] generates a dendogram (a tree of clusters) using the the FASTMAP heuristic [11]. Given N instances, we find a dimension of great variability by drawing a line drawn between the two distant points found as follows: *first* select any instance Z at random; *then* find the instance X that is furthest away from Z; and *finally* find the instance Y that is furthest away from X. The line $\overline{XY}$ is an approximation to the first component found by PCA (but is found in linear time). As shown in Figure 2.a, an orthogonal dimension to $\overline{XY}$ can be found by declaring that the line $\overline{XY}$ is of length c and runs from point $(0, 0)$ to $(0, c)$. Each instance now has a distance a to the origin (instance X) and distance b to the most remote point (instance Y). From the Pythagoras and cosine rule, each instance is at the point $x = (a^2 + c^2 - b^2)/(2c)$ and $y = \sqrt{a^2 - x^2}$. Figure 2.a shows four quadrants defined by the median values of each dimension $(\hat{x}, \hat{y})$: *NorthWest, NorthEast, SouthWest, SouthEast*. IDEA recurses on each quadrant. Theoretically, this is a $O(N.log(N))$ process since finding the median requires sorting all values. However, in practice, the algorithm's runtime is usually linear on the number of instances.

Once the data is mapped in this way, then the goals of learning can easily be visialised. Consider the three clusters labeled C, C', C'' in Figure 2.d. Suppose a manager of a project in the orange cluster C is considering how to decrease the development effort of that project (of all the neighbors of that cluster, the green cluster C' has the lowest development effort). Accordingly, that manager would learn rules over the C' data to find treatments that convert projects of type C to C' (note that such a strategy is *not* available to the manager of projects in the dark green cluster C'': no neighbor of C'' has a shorter development effort so there we would advise to maintain the status quo).

W [6] uses a standard predictive data mining technique (k-th nearest neighbor classification) but does so in such a way to generate decisions (alternate names for W are"Dub-ya" or the "the decider") W reports what needs to be changed to most improve a project:

- One way to make project estimates is to reflect on the k-th nearest neighbors to a current example. W sorts those k neighbors into l examples that is most "loves" and h examples it most "hates" (so $k = l + h$).
- For example, the "loved" examples might have least effort while the "hate" examples are most defects. W discretizes all values then counts how often value ranges occur in "loved" or "hate".
- Ranges are then sorted to find which ones are common in "loved" and rare in "hate".
- W runs what-if queries using the first i items in that sort. It returns a rule containing the first i items that most change to the current project in order to (say) push *towards* projects that are built faster and *away* from projects that have most defects.

Fig. 3. Simple constrast learning with W.

Landscape mining is silent on the nature of the learners applied to each cluster. Like Newell's knowledge level, the actual learner is a low-level detail. We prefer the IDEA algorithm shown above (since it runs in linear time) followed by some case-based reasoning tool such as the W tool (see Figure 3). Other teams have generated clusters like Figure 2.d using recursive regression methods [1]. Regardless of how the landscape is generated, the general principle is the same:

- Look before you leap.
- Cluster before running a (say) classifier in order to focus the learning of specialized regions within the data.

Decision Systems

At a recent panel on software analytics [28] at ICSE'2012, industrial practitioners reviewed the state of the art in data mining. Panelists commented "prediction is all well and good, but what about decision making?". Predictive models are useful- they focus an inquiry on particular issue. But predictive models are sub-routines in a higher level *decision process*.

Generating decisions is the task of a *decision system*. In Idea Engineering, decisions systems run as a post-processor to landscape mining. Contrast set learners are applied to neighboring clusters in order to learn the difference between each cluster. For an example of a simple contrast learner, see the W learner of Figure 3. For details on a more elaborate contrast learner, see Milton and Menzies' WHICH system [26].

Contrast sets support a range of decisions discussed in the literature. Consider a standard definition of a management support system [7], [29]. Such systems try to offer a sense of "comfort" to managers that all problems are known and managed. This "comfort" has having three components:

1) *Finding a problem* = detection + diagnosis;
2) *Solving a problem* =
   find alternatives + evaluation + judgment;
3) *Resolution* = monitoring the effect of the solution.

- *Detection* is just:
  - Find what cluster contains the current project then
  - Predict the properties of that project from the other examples in this cluster.
- As to *Diagnosing* a problem, this requires recognizing two clusters:
  - $B4$ = before; i.e. a previous cluster where a manager was content with a project's status;
  - Now = a current cluster where a manager is now worried about that status

  In this case, the diagnosis of what has gone wrong is just the contrast set of

$$Diagnosis = B4 - Now \tag{1}$$

- To *find alternative* solutions to a problem, we seek the contrast set between Now and nearby clusters with better properties; e.g. lower detects, faster development times, etc. In this case, the set of possible solutions S is any contrast set:

$$(C_i > Now) \wedge (S_i = Now - C_i) \tag{2}$$

- As to *evaluation* and *judgment*, that could be implemented many ways including some user session to browse and debate the S_i solutions found above.
- Finally, to *monitor* a resolution, we need to find the list of all things that might go wrong. Given the current cluster Now, we need to check all the neighbor clusters with worse performance than the current. The list of monitors M is then all the contrast sets that might drive us into those undesired clusters:

$$(C_i < Now) \wedge (M_i = Now - C_i) \tag{3}$$

  Note that this last expression is very similar that of Equation 2, with the sole difference of the selection criteria for C_i (here, we only look at clusters with a *worse* performance score).

Fig. 4. Decision systems via clusters+contrasts: implementing management support systems.

Landscape mining and contrast set learning can support all these activities, see Figure 4.

For another example of different inds of desicison discussed in the literature, we may turn to the ICSE'12 survey of Buse and Zimmermann who surveyed 100+ managers and programmers at Microsoft [8]. They report that that community has various *information needs* concerning

- *The past*: what trends exist over time? what relationships hold in the historical data?;
- *The present*: what alerts are raised by the current data? how does our data compare to known benchmarks? and
- *The future*: What forecasts might we generate? What is the space of the possible what-ifs in this area? How does our data compare to the end goals of this project?.

Buse and Zimmermann expand three information needs into the following nine tasks:

	Past	Present	Future
Exploration (find)	Trends	Alerts	Forecasts
Analysis (explain)	Summarize	Overlays	Goals
Experiment (what-if)	Model	Benchmarks	Simulate

- *Overlays* are predictions associated with each cluster; e.g. the mean and standard deviation of known class values in each cluster;
- *Goals* compare system performance with respect to some desired values. This is just the overlay values minus the goal values. If displayed over a diagram like Figure 2.d, he managers can quickly see how well (or how badly) different projects are performing with respect to current goals.
- *Benchmarks* compare system performance with respect to established baselines. Like goals, this is just the overlay values minus the goal values.
- If we tracked how project changes resulted in a project migrating around Figure 2.d then:
  - Past *trends* would be the track seen in historical data;
  - Forward *trends* would be an extrapolation of the past trend.
- *Forecasts* would be just be predictions resulting from mapping a project into a cluster then predicting the properties of that project from the other examples in this cluster. Future *forecasts* could be implemented by applying the forecast method to the clusters seen in the forward trend.
- *Simulations* could be implemented in two ways:
  - For *simulation via lookup*, just generate something like Figure 2.d then read off the predictions for class values seen in each cluster. In this approach, the clustering process is like a what-if query that groups the data into sets of related possibilities.
  - For *simulation via execution*, some domain model could be executed using inputs drawn from the clusters of Figure 2.d. In this approach, the clustering process divides the input space of the executable, after which we can sample different modes of the sample by sampling for different clusters.
- *Alerts* could raised if new data does not fit into the old clusters. To implement such alerts, we use the dendogram that generated Figure 2.d:
  1) For each leaf cluster, randomly select pairs of instances (say, 100 times). Record the distribution of distances found in that sample.
  2) Take new data and walk it down the dendogram to find the leaf cluster. The new data is alien if it is an outlier on the distribution generated by step 1.

  Note that if the results of step 1 are pre-computed and cached, then step 2 could report anomalies in time $O(logN)$; i.e. just the time required to map new data down the dendogram to a leaf cluster,

Fig. 5. Decision systems via clusters+contrasts: implementing the nine tasks of Buse and Zimmermann.

Figure 5 discusses the use of clustering+contrast set learning for implementing the nine tasks of Buse and Zimmermann.

Discussion Systems

Pablo Picasso once said "computers are stupid- they only give you answers". Social reasoners are not stupid- they know that while predictions and decisions are important, so to are the questions and insights generated on the way to those conclusions. Within a society of carbon and/or silicon-based agents, *discussion systems* allow those agents to share, reflect, and try to improve each other's insights. In my view:

- Discussion systems are the next great challenge for the predictive modeling community. In the digital world of the 21^{st} century, such social reasoners are essential tools. Without them, humans will be unable to navigate and exploit the ever-increasing quantity of readily-accessible digital information.
- Discussion systems can be built from decision systems (which, in turn, can be built from landscape miners).

My thesis is that social reasoners can be built by refactoring of predictive technologies. For example, the following example extends W to social reasoning:

- Consider two different cost estimates E_1 and E_2 from different contractors competing to build some software.
- Using the COCOMO effort prediction model [3], an analyst might identify different assumptions A_1 and A_2 made by each contractor.
- If we apply W's contrast set learners to those assumptions, we could then isolate the factors that separate the two estimates.
- Then, we might report "the core issue here is the difference between A_1 and A_2; here is my analysis of the probability of that difference; what do you think?".

Note the key features of this example: the outcome is not a *prediction* or a *decision* on what to change, but *questions* that focused on key issues in the domain (specifically, which assumptions were most believable).

As shown in Figure 6, the idea of improving inference by connecting human and computer and computer agents dates back to at least 1939. The new idea of this paper is that, as shown in Figure 7, **social reasonong can be implemented as a refactoring of our current predictive technologies**. Note how, in Figure 7, the underlying tools are predictive and decision systems. Apart from that, rest of a social reasoner is concerned with the discussion around those models. For example:

- A social reasoner must be able to succinctly **say** what is in the data. It is axiomatic that you cannot interact and critique and extend the ideas of another agent unless you can understand that agent. That is, social reasoning systems need a shared discussion language that is used and understood by all parties in that society. Hence, social reasoning should avoids learners that rely on arcane internal representation such as SVM, random forests, naive Bayes, neural nets, or PCA. On the other hand, social reasoning systems could use feature/instance selection tools to discard spurious details; then contrast set learners to find the deltas between the remaining data.
- Another task is to **reflect** on a model to learn how models can and should change over the space of the data.
- Social reasoners need also **share** the data and rules which means transferring the essence of the data between agents (and ensuring the shared data does not violate confidentiality [34]).
- Finally, to accommodate large societies, all the above must happen very quickly so this can **scale** to large data

- Alan Turing believed that systems of logic could execute inside silicon or carbon [10]. In his 1939 Ph.D. thesis, he discussed the value of the interactions within a society of such systems: *"The well-known theorem of Gödel (1931) shows that every system of logic is in a certain sense incomplete, but at the same time it indicates means whereby from a system L of logic a more complete system L' may be obtained. By repeating the process we get a sequence L, $L_1 = L'$, $L_2 = L_1'$, ... each more complete than the proceeding. A logic L_ω may then be constructed in which the provable theorems are the totality of theorems provable with the help of logics L, L_1, L_2..."* [40].
- In the 1950s, Kelly proposed personnel construct theory as a methodology for using modeling to reveal previously hidden domain assumptions [19].
- In the 1970s and 1980s, the knowledge acquisition community propose rapid (?rabid) construction of executable knowledge bases to reveal previously unrecognized interactions between chunks of expert knowledge [24].
- At a 2003 keynote to the ProSim process simulation conference, Walt Scacchi reported on his experience where software process models are rarely executed. Rather, their value (according to Scacchi) was as tools to help explicit domain details [41].
- Since 2009, Tao Xie has been exploring "cooperative testing schemes" where humans and algorithms interact to propose informative test cases. His framework infers likely test intentions to reduce the manual effort in specification of test intentions [42].
- In a 2010 keynote to the PROMISE conference on predictive models, Mark Harman said that modeling systems should offer more than just conclusions- rather they should also "yield insight into the trade offs inherent in the modeling choices available" [14].
- In 2012, Egyed et al. used the differences between incorrect and incomplete reasoning. They demonstrated that it is even possible to eliminate incorrect reasoning in the presence of inconsistencies at the expense of marginally less complete reasoning [31]

Fig. 6. Some related work.

	what	tasks	uses
0	**do**	predict, decide	regression, classification, nearest neighbor reasoning,...
1	**say**	summarize, plan, describe	instance section, feature selection, contrast sets
2	**reflect**	trade-offs, envelopes, diagnosis, monitoring	clustering, multi-objective optimization, anomaly detectors
3	**share**	privacy, data compression, integration old & new rules, recognize and debate deltas between competing models	contrast set learning, transfer learning
4	**scale**	do all the above, very quickly	?

Fig. 7. Four layers of social reasoning.

sets. One reason that I focus on data mining for social reasoning is that data mining methods can scale to very large tasks. The same cannot be said for other methods. Previously, I found that a purely logical method for unifying different reasoning tasks suffered from exponential runtimes [25].

In some sense, a social reasoner is the opposite of the world wide web. The web was designed for information transport and access. The web's primary goal was the rapid sharing of new information. If the web was a social reasoning system, it would be possible to (i) instantly query each web page to find other pages with similar, or disputing, beliefs; (ii) find the contrast set between then agreeing and disputing pages; (ii) then run queries that helped the reader assess the plausibility of each item in that contrast set. In the social reasoning web, most of the authoring would relate to critiquing and updating content, rather than just creating new content. Note that much of the current predictive modeling research would not qualify as a social reasoner since, in the usual case, most of that literature is still struggling with methods to create one model, let alone updating a model as time progresses.

As a final note, one fascinating open issue is how to assess social reasoners. In social reasoning, the goal of a model is to find its own flaws and to replace itself with something better- which brings to mind a quote from Susan Sontag: "the only good answers are the ones that destroy the questions". That is, we should not assess such models by accuracy, recall, precision etc. Rather, the assessment should be on the *audience engagement* they engender. For example- the audience involvement seen in the "we are here" pattern on page 2, but perhaps with more ways to assess the coverage of the options space.

ACKNOWLEDGMENTS: All that is sensible in this paper comes from the careful comments made by Alexander Egyed, Mark Harman, Abram Hindle, Ekrem Kocaguneli, Andrian Marcus & Tom Zimmermann (but any nonsense in the above should be attributed solely to me).

REFERENCES

[1] Nicolas Bettenburg, Meiyappan Nagappan, and Ahmed E. Hassan. Think locally, act globally: Improving defect and effort prediction models. In *MSR'12*, 2012.

[2] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 137–143, 2006.

[3] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches - a survey. *Annals of Software Engineering*, 10:177–205, 2000.

[4] G. Boetticher, Tim Menzies, and T. Ostrand. The PROMISE Repository of Empirical Software Engineering Data, 2007. http://promisedata.org/repository.

[5] Rcd. Borges and T. Menzies. Learning to change projects. In *Proceedings of PROMISE'12, Lund, Sweden*, 2012. Available from http://menzies.us/pdf/12idea.pdf.

[6] Adam Brady and Tim Menzies. Case-based reasoning vs parametric models for software quality optimization. In *PROMISE'10*, 2010. Available from http://menzies.us/pdf/10cbr.pdf.

[7] C.H.P. Brookes. Requirements Elicitation for Knowledge Based Decision Support Systems. Technical Report 11, Information Systems, University of New South Wales, 1986.

[8] Raymond P L Buse and Thomas Zimmermann. Information needs for software development analytics. In *ICSE'12, Industry Track*, 2012.

[9] Karel Dejaeger, Wouter Verbeke, David Martens, and Bart Baesens. Data mining techniques for software effort estimation: A comparative study. *IEEE Transactions on Software Engineering*, 38:375–397, 2012.

[10] George Dyson. *Turing's Cathedral: The Origins of the Digital Universe*. Pantheon, 2012.

[11] Christos Faloutsos and King-Ip Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD '95, pages 163–174, 1995.

[12] Natalie Gruska, Andrzej Wasylkowski, and Andreas Zeller. Learning from 6,000 projects: lightweight cross-project anomaly detection. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 119–130. ACM, 2010.

[13] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic review of fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, (PrePrints), 2011.

[14] Mark Harman. The relationship between search based software engineering and predictive modeling. In *PROMISE*, page 1, 2010.

[15] Mark Harman, Yue Jia, and Yuanyuan Zhang. App store mining and analysis: Msr for app stores. In *MSR*, pages 108–111, 2012.

[16] C.C. Hayes and M.I. Parzen. Quen: An achivement test for knowledge-based systems. *IEEE Transactions of Knowledge and Data Engineering*, 9(6):838–847, Nov/Dec 1997.

[17] A. Hindle. Green mining: A methodology of relating software change to power consumption. In *Proceedings, MSR'12*, 2012.

[18] LiGuo Huang, Daniel Port, Liang Wang, Tao Xie, and Tim Menzies. Text mining in supporting software systems risk assurance. In *IEEE ASE'10*, pages 163–166, 2010. Available from http://menzies.us/pdf/10textrisk.pdf.

[19] G.A. Kelly. *The Psychology of Persona] Constructs. Volume 1: A Theory of Personality. Volume 2: Clinical Diagnosis and Psychotherapy*. Norton, 1955.

[20] E. Kocaguneli, Tim Menzies, and J. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 2012. Available from http://menzies.us/pdf/11comba.pdf.

[21] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann. Local vs. global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*, page 1, 2012. Available from http://menzies.us/pdf/12localb.pdf.

[22] Tim Menzies, Andrew Butcher, Andrian Marcus, Thomas Zimmermann, and David Cok. Local vs global models for effort estimation and defect prediction. In *IEEE ASE'11*, 2011. Available from http://menzies.us/pdf/11ase.pdf.

[23] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, January 2007. Available from http://menzies.us/pdf/06learnPredict.pdf.

[24] Tim Menzies and L. Mason. Some prolog macros for rule-based programming: Why? how? In *Third ACM SIGPLAN Workshop on Rule-Based Programming (RULE02) Pittsburgh, PA, October 5*, 2002. Available from http://menzies.us/pdf/03datasniffing.pdf.

[25] Tim Menzies and C.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.

[26] Tim Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener. Defect prediction from static code features: Current results, limitations, new approaches. *Automated Software Engineering*, (4), December 2010. Available from http://menzies.us/pdf/10which.pdf.

[27] Tim Menzies, S. Williams, Oussama El-Rawas, B. Boehm, and J. Hihn. How to avoid drastic software process change (using stochastic stability). In *ICSE'09*, 2009. Available from http://menzies.us/pdf/08drastic.pdf.

[28] Tim Menzies and Thomas Zimmermann. Goldfish bowl panel: Software development analytics. In *ICSE*, pages 1032–1033, 2012.

[29] H. Mintzberg. The Manager's Job: Folklore and Fact. *Harvard Business Review*, pages 29–61, July-August 1975.

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

[31] Alexander Nöhrer, Armin Biere, and Alexander Egyed. A comparison of strategies for tolerating inconsistencies during decision-making. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 11–20, 2012.

[32] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Where the bugs are. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 86–96, New York, NY, USA, 2004. ACM.

[33] Francis Palma, Hadi Farzin, and Yann-Gael Gueheneuc. Recommendation system for design patterns in software development: An dpr overview. In *Third International Workshop on Recommendation Systems for Software Engineering*, 2012.

[34] Fayola Peters and Tim Menzies. Privacy and utility for defect prediction: Experiments with morph. In *ICSE'12*, 2012. http://menzies.us/pdf/12privacy.pdf.

[35] A.A. Porter and R.W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, pages 46–54, March 1990.

[36] D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In *Proceedings of ASE'11*, 2011.

[37] Daniel Rodríguez, Mercedes Ruiz Carreira, José C. Riquelme, and Rachel Harrison. Multiobjective simulation optimisation in software project management. In *GECCO*, pages 1883–1890, 2011.

[38] S. Stepney, S.L. Braunstein, J.A. Clark, A. Tyrrell, A. Adamatzky, R.E. Smith, T. Addis, C. Johnson, J. Timmis, P. Welch, et al. Journeys in non-classical computation i: A grand challenge for computing research. *International Journal of Parallel, Emergent and Distributed Systems*, 20(1):5–19, 2005.

[39] Burak Turhan. On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering*, 17:62–74, 2012.

[40] A. Turing. Systems of logic based on ordinals. *Proc. London Math. Soc*, 45:161–228, 1939.

[41] Paul Wernick and Walt Scacchi. Special issue on prosim 2003, the 4th international workshop on software process simulation and modeling, portland, or, may 2003. *Software Process: Improvement and Practice*, 9(2):51–53, 2004.

[42] Tao Xie. Cooperative testing and analysis: Human-tool, tool-tool, and human-human cooperations to get work done. In *Proc. 12th International Working Conference on Source Code Analysis and Manipulation (SCAM 2012), Keynote Paper*, September 2012.