# Predicting the Future of Predictive Modeling

Tim Menzies
CS & EE, WVU, USA
tim@menzies.us

NSF Workshop: Planning Future Directions in AI &SE (AISE'12)
September 10, 2012

**ABSTRACT:** It is now well established that predictive models can be generated from the artifacts of software projects. So it is time to ask "what's next?". I suggest that predictive modeling tools can and should be refactored to address the near-term issue of *decision systems* and the long-term goal of *social reasoning*.

**INTRODUCTION:** In software engineering (SE) **there is much we are *seeing*, but little we are *learning***. The sheer volume of SE data is overwhelming. As of October 2011, 10,000 projects are monitored at http://CIA.vc- with one new commit every 17 seconds. The open source platform SourceForge.Net hosts over 300K projects, and according to Github.com 1M people host 2.9M GIT repositories. The bug database of the Mozilla Firefox projects now contains almost 700K reports according to Ohloh.Net. Yet from that data, we have extracted nearly zero general principles- the usual result is, across all this data, is that what works one project may not work on another [28, 29].

Hence there is an urgent need for better analysis of this data. Further, due to the volume of information, that analysis must be (at least partially) automated. Hence, AI research in data mining has been widely adopted in predictive modeling community with SE. In this brief note, I critique the state of the art in that field and suggest a future direction.

In summary, I think we need to move beyond mere predictive modeling. Last century, it was not known if software projects contained sufficient structure to support data mining, though some preliminary results by Porter were encouraging [37]. Now, we know better. **Many different kinds of artifacts from software projects contain a signal that can be revealed via data mining** including:

- apps store data [13];
- process data which can predict overall project effort [18];
- process models which can find effective project changes [30, 40];
- operating system logs that predict software power consumption [15];
- natural language requirements documents which can be text-mined to find connections between program components [14];
- XML descriptions of design patterns that can be used to recommend particular designs [35];
- email lists that reveal the human networks inside software teams [2];
- execution traces that generate normal interface usage patterns [10];
- bug databases that can generate defect predictors to guide inspection teams to where the code is most likely to fail [23, 27, 34, 37].

It is now well-established that predictive models can be built from software projects artifacts. **So it is now time to ask "what's next?"**.

Stepney et al. [43] assert 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 new and novel areas. According this progression, we are now leaving the age of *prediction systems* and entering the age of *decision systems*. After that, we should move to the era of *social reasoning*:

$$\underbrace{prediction}_{now} \rightarrow \underbrace{decision}_{next} \rightarrow \underbrace{social\ reasoning}_{future}$$

This progression can use the current skills of the predictive modeling community while still stepping us towards some distant grand goal. For example, consider the $W$ case-based planning system (also known as "Dub-ya" or the "the decider") [5]. 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. $W$ then applies *contrast set learning* [24] to find attribute ranges that are more common in "loved" than "hate". From those ranges, $W$ proposes a change to the current project in order to (say) push *towards* projects that are built faster and *away* from projects that have most defects.

As to the next step (from decision systems to social reasoning), that is discussed below. For now, all we need observe is that **with a little refactoring, this community has the tools and talents that can take it to the next level of research**. For example, $W$ is a decision system (about how to best change a project) that is a small change to a current prediction technology (case-based reasoning and contrast-set learning).

It turns out that, at least is this field, building decision systems is somewhat of a radical idea. To see why, we need a little history lesson:

**HISTORICAL PATTERNS:** While it is rarely stated, the original premise of predictive modeling was that predictions 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 papers about $L$ learners applied to $D$ data sets and evaluated via some $M * N$ cross-validation study. Trying every learner on every data set is not particularly insightful. Many SE data sets have limited information content- they 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 [16]. For such data, it may be a waste of time to try the latest and greatest most complex learner[1]. Hall et al. [11] and Dejaeger [8] 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 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 [44]. Hence, many researchers now explore "local learners" that eschew single global conclusions in favor of more context-dependent conclusions [1, 22, 38].

Lastly, another issue with $D * L * M * N$-style 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.

[1]Though *some* form of data mining is still essential for understanding this data. Even though most of the data is superfluous, data miners are required to isolate the essential portions of that data.

**BETTER STUDIES ON PREDICTION:** What are the alternatives to simplistic $D * L * M * N$ studies? One approach, recommended by Pat Langley [20], is to reflect on *why* different learners give different decisions on the same data set. Such *delta explanation studies* can be highly insightful. For example, once I compared learners that built models using at most $N$ attributes [19]. The performance of the $N = 1$ learner was always worse than those that used $N \geq 1$, thus showing that software projects are complex multi-dimensional entities that cannot be characterized via simplistic models (e.g. the infamous McCabe $v(g) > 10$ rule).

Another approach is to study the people who build and use the predictive models. If we focused less on the algorithms, and more on the people and processes that use them, then we might understand the black art of (i) parameter tuning [39] or (ii) how multiple learners might be effectively combined into ensemble [18]. Studying skilled practitioners is important since recent evidence suggests that even amongst supposed experts, skill levels can vary dramatically, [41]. This means that the usability of data mining tools is an open and pressing matter. Issues in this area include what are the difference between expert and novice data miners.

One way to improve the skill of non-experts might be to catalog and analyze the data mining analysis patterns used by expert data miners [21]. For an example of such a data mining analysis patterns, consider the "bad smells" detectors of Shepperd et al. [42] or (b) the "we are here" pattern used by my colleague Christian Bird when he presents results at user meetings at Microsoft. To prepare for those meetings, he:

- Writes a few slides describing his analysis and conclusions;
- Creates a spreadsheet containing interesting subsets of the data (where "most interesting" might be selected by a data miner).

At the meetings, he presents his slides then turns the spreadsheet over to the users. According to Bird, their first act is usually to check "we are here"; i.e. that they can find interesting parts of their project data in Bird's spreadsheets. If the data passes that sanity check, then they start running queries (sorts, selects, etc) to confirm (or refute) Bird's conclusions. In this way, Bird increases user engagement and user ownership of the analysis process. Tools to support this kind of analysis might include feature selection and instance selection.

**FROM PREDICTION TO DECISION:** At a recent panel on software analytics [31] 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 since they focus an inquiry onto particular issues- but **predictive models are sub-routines in a higher level *decision process*.**

We know of two definitions of such decision making processes- one very general from Brookes [6] and one very specific structure developed recently at Microsoft [7]. According to Brookes (who worked from an early model by Mintzberg [32]), the goal of a decision system is a sense of "comfort" that all problems are known and managed. He defines "comfort" as having three components:

- *Finding a problem* (i.e. detection + diagnosis);
- *Solving a problem* (i.e. find alternatives + evaluation + judgment);
- and *Resolution* (i.e. monitoring the effect of the solution).

An alternate view, more grounded in recent research, comes from Buse and Zimmermann who report a survey of 100+ managers and programmers at Microsoft [7]. They report *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?.

An open question is how (or indeed, if) we can unify and implement these two different descriptions of decision systems. Just as a thought experiment, for the purposes of this paper, I tried to combine predictive tools with the models of Brookes, Buse & Zimmermann:

- Clearly, any number of predictive technologies (e.g. classifiers, regression models) could be applied to the forecasts used in the Buse&Zimmermann model.
- As to the other parts of those models, they recommend monitoring a current solution to detect trends that lead to alerts where more action is required. Brooke's problem detection might be implemented as continually running the forecasts recommended by Buse&Zimmerman (to look for undesired outcomes).
- As to relationships in the data, the contrast sets of $W$ might offer a succinct summary of the relationships that most effect planning:
  - $W$' contrast sets could *evaluating* and *judging* alternatives via distance metrics that comment on the cost of moving from "hated" projects to all the known "liked" projects.
  - If a current project falls into a "hated" cluster then to *generate alternatives* for the Brookes model, we need only seek the contrast set of differences between this "hated" cluster and another that managers might "like" more.
- Contrast learners could also support *diagnosis* and *monitoring*:
  - To *diagnose* why suddenly a project has moved from "like" to "hate", find the contrast set between where it was liked to its current context (where it is "hated").
  - Further, to *monitor* for changes that could most hurt a project that is currently liked, build a disjunction of all the contrast between this liked context and all the nearly hated contexts.

My conclusion from this preliminary analysis is that **decision tools can be built by refactoring prediction tools**. That is, this community is well-positioned to move from prediction to decision systems.

As an aside- I note that decision systems are more than just traditional operations research since it focuses more on symbolic models (assertions of the form "do this!" or "don't do that!") rather than numeric representations. Having separated these fields, I rush to add that there is a wealth of insight that operations research can offer decision systems. Also, decision systems might help operations research- particularly for generating succinct and understandable symbolic models from numeric results.

**SOCIAL REASONING:** 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, social reasoners share, reflect, and try to improve each other's insights.

Social reasoning is 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.

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 1, 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 2, **social reasonong can be implemented as a refactoring of our current predictive technologies**. Note how, in Figure 2, 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 [36]).
- Finally, to accommodate large societies, all the above must happen very quickly so this can **scale** to large data 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 [26].

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).

- Alan Turing believed that systems of logic could execute inside silicon or carbon [9]. 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 $L_\omega$ may then be constructed in which the provable theorems are the totality of theorems provable with the help of logics L, $L_1$, $L_2$..."* [45].
- In the 1950s, Kelly proposed personnel construct theory as a methodology for using modeling to reveal previously hidden domain assumptions [17].
- 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 [25].
- 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 [46].
- 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 [47].
- 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" [12].
- 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 [33]

**Figure 1: 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 | ? |

**Figure 2: Four layers of social reasoning.**

## 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] 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`.

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

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

[14] 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.

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

[16] 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`.

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

[18] 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`.

[19] Ekrem Kocaguneli, Gregory Gay, Tim Menzies, Ye Yang, and Jacky W. Keung. When to use data from other projects for effort estimation. In *IEEE ASE'10*, 2010. Available from `http://menzies.us/pdf/10other.pdf`.

[20] Pat Langley. Editorial: On machine learning. *Machine Learning*, 1:5–10, 1986.

[21] Tim Menzies, Christian Bird, Thomas Zimmermann, Wolfram Schulte, and Ekrem Kocaganeli. The inductive software engineering manifesto: principles for industrial data mining. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, MALETS '11, pages 19–26, New York, NY, USA, 2011. ACM.

[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 Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from `http://menzies.us/pdf/03tar2.pdf`.

[25] 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`.

[26] 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`.

[27] 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`.

[28] Tim Menzies and Martin Shepperd. Special issue on repeatable results in software engineering prediction. *Empirical Software Engineering*, 17(1-2):1–17, 2012.

[29] Tim Menzies and Forrest Shull. The quest for convincing evidence. In A. Oram and G.Wilson, editors, *Making Software: What Really Works and We Believe it*. O'Reilly Books, 2010.

[30] 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`.

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

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

[33] 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.

[34] 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.

[35] 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.

[36] Fayola Peters and Tim Menzies. Privacy and utility for defect prediction: Experiments with morph. In *ICSE'12*, 2012.

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

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

[39] M. Postema, Tim Menzies, and X. Wu. A decision support tool for tuning parameters in a machine leraning algorithm. In *The Joint Pacific Asia Conference on Expert Systems/Singapore International Conference on Intelligent Systems. (PACES/SPICIS '97)*, 1997. Available from `http://menzies.us/pdf/97pakdd.pdf`.

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

[41] Martin Shepperd. It doesn't matter what you do but does matter who does it!, 2011. Available on-line at http://goo.gl/JbXcL.

[42] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect data sets, 2012. Available from http://goo.gl/OlHNh.

[43] 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.

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

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

[46] 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.

[47] 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.