Industry can get any empirical research it wants

(Publish open source data, and some example scripts.)

Tim Menzies

prof, cs, **ncstate**, usa
acm-ieee-ase fellow; eic ASEj
timm@ieee.org
http://timm.fyi

Oct3'25

[Men07]: Data Mining Static Code Attributes to Learn Defect Predictors, TSE'07
[Men25] T. Menzies, "Retrospective: Data Mining Static Code Attributes, TSE'25

The Portland Context

- Born from open source culture in Portland, Oregon
- *"We wore no suite and tie in our photos. We did not comb our hair"*
- Philosophy: `svn commit -m "share stuff"` will change SE research
- But unhappy with SOTA data mining in SE
- **Key Insight**: Walking around Chicago's Grant Park (2004)
  - **Tim Menzies** and **Jelber Sayyad** lamented: *"Must do better... Why don't we make conclusions reproducible?"*

The Radical Idea

- In 2025 hard to believe "reproducible SE" was radical
- **Lionel Briand** (2006): *"no one will give you data"*
- Yet we persisted...

What do I do? How can I help you?

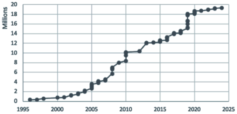
papers | pre-prints | dblp | timm@ieee.org | +1-304-376-2859 | rm 3304, EB2 (map)

So I seek **talented** grad students & **industrial partners** to find + fix the **problems in real-world AI/ML**.
Is that you? Maybe "yes" if **you want to be a leader in AI** (and not just another follower).

Schedule an Appointment

videos |

total funding
(in 2025 dollars, assuming 3% inflation)



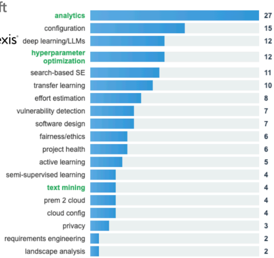
Currently:
- **Teaching:** Graduate Intro to SE
- **Editor-in-chief:** Automated Software Engineering journal
- **Associate Editor:** IEEE Trans SE
- **Editorial Board:** Communications of the ACM (opinions)
- **Track co-chair:** ESEM'25 (doctoral symposium)
- **Program committees:** ICSE'26, ICSE'25, FSE'25, ASE'25, SANER'25, ESEM'25, CAIN'25, AAAI(ethics)'25

Ph.D.s (current): Ph.D.s (graduated 🎓):

Much thanks and good luck to y'all!. 👏
Can't wait to see what you do next! 🚀

National Science Foundation
NASA
Microsoft
LexisNexis
NATIONAL ARCHIVES
Meta

Last decade: ~140 papers
(estimated ~5 tags per paper, 2014-2025)



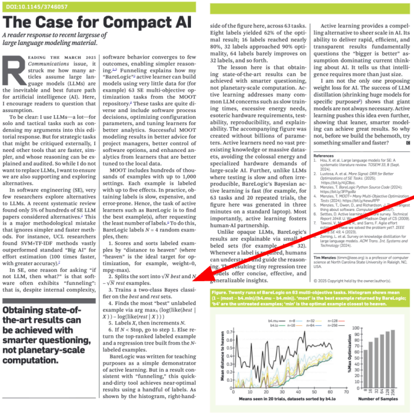
🚀 **Major Commercial Breakthrough Technologies:**
- **NASA-Proven Analytics Platform** - Mission-critical systems, space-grade reliability (27 papers)
- **Microsoft Hyperparameter Optimization** - 210X faster, enterprise validated (12 papers)
- **LexisNexis Text Mining Platform** - Legal document analysis, millions of documents
- **Cloud Configuration Systems** - Prem 2 cloud (4) + cloud config (4) solutions
- **Software Design Intelligence** - AI-driven design optimization (7 papers)
- **Project Health Analytics** - Predictive monitoring and early warning systems (6 papers)

3

Recent work: ultra-low cost active learning

https://timm.fyi/assets/pdf/cacm25.pdf

[CACM'25: Menzies, Compact AI]



DOI:10.1145/3746057

The Case for Compact AI

A reader response to recent largesse of large language modeling material.

READING THE MARCH 2025 *Communications* issue, it struck me how many articles assume large language models (LLMs) are the inevitable and best future path for artificial intelligence (AI). Here, I encourage readers to question that assumption.

To be clear: I use LLMs—a lot—for solo and tactical tasks such as condensing my arguments into this editorial response. But for strategic tasks that might be critiqued externally, I need other tools that are faster, simpler, and whose reasoning can be explained and audited. So while I do not want to replace LLMs, I want to ensure we are also supporting and exploring alternatives.

In software engineering (SE), very few researchers explore alternatives to LLMs. A recent systematic review found only 5% of hundreds of SE LLM papers considered alternatives.[1] This is a major methodological mistake that ignores simpler and faster methods. For instance, UCL researchers found SVM+TF-IDF methods vastly outperformed standard "Big AI" for effort estimation (100 times faster, with greater accuracy).[7]

In SE, one reason for asking "if not LLM, then what?" is that software often exhibits "funneling": that is, despite internal complexity, software behavior converges to few outcomes, enabling simpler reasoning.[2,5] Funneling explains how my "BareLogic"[3] active learner can build models using very little data for (for example) 63 SE multi-objective optimization tasks from the MOOT repository.[4] These tasks are quite diverse and include software process decisions, optimizing configuration parameters, and tuning learners for better analytics. Successful MOOT modeling results in better advice for project managers, better control of software options, and enhanced analytics from learners that are better tuned to the local data.

Obtaining state-of-the-art results can be achieved with smarter questioning, not planetary-scale computation.

MOOT includes hundreds of thousands of examples with up to 1,000 settings. Each example is labeled with up to five effects. In practice, obtaining labels is slow, expensive, and error-prone. Hence, the task of active learners such as BareLogic is to find the best example(s), after requesting the least number of labels.[6] To do this, BareLogic labels $N = 4$ random examples, then:

1. Scores and sorts labeled examples by "distance to heaven" (where "heaven" is the ideal target for optimization, for example, weight=0, mpg=max).
2. Splits the sort into $\sqrt{N}$ *best* and $N - \sqrt{N}$ *rest* examples.
3. Trains a two-class Bayes classifier on the *best* and *rest* sets.
4. Finds the most "best" unlabeled example via $\arg\max_X (\log(\text{like}(best \mid X)) - \log(\text{like}(rest \mid X)))$
5. Labels X, then increments N.
6. If $N <$ Stop, go to step 1. Else return the top-ranked labeled example and a regression tree built from the N-labeled examples.

BareLogic was written for teaching purposes as a simple demonstrator of active learning. But in a result consistent with "funneling," this quick-and-dirty tool achieves near-optimal results using a handful of labels. As shown by the histogram, right-hand-side of the figure here, across 63 tasks. Eight labels yielded 62% of the optimal result; 16 labels reached nearly 80%, 32 labels approached 90% optimality, 64 labels barely improves on 32 labels, and so forth.

The lesson here is that obtaining state-of-the-art results can be achieved with smarter questioning, not planetary-scale computation. Active learning addresses many common LLM concerns such as slow training times, excessive energy needs, esoteric hardware requirements, testability, reproducibility, and explainability. The accompanying figure was created without billions of parameters. Active learners need no vast preexisting knowledge or massive datasets, avoiding the colossal energy and specialized hardware demands of large-scale AI. Further, unlike LLMs where testing is slow and often irreproducible, BareLogic's Bayesian active learning is fast (for example, for 63 tasks and 20 repeated trials, the figure here was generated in three minutes on a standard laptop). Most importantly, active learning fosters human-AI partnership.

Unlike opaque LLMs, BareLogic's results are explainable via small labeled sets (for example, N = 32). Whenever a label is required, humans can understand and guide the reasoning. The resulting tiny regression tree models offer concise, effective, and generalizable insights.

Active learning provides a compelling alternative to sheer scale in AI. Its ability to deliver rapid, efficient, and transparent results fundamentally questions the "bigger is better" assumption dominating current thinking about AI. It tells us that intelligence requires more than just size.

I am not the only one proposing weight loss for AI. The success of LLM distillation (shrinking huge models for specific purposes[8]) shows that giant models are not always necessary. Active learning pushes this idea even further, showing that leaner, smarter modeling can achieve great results. So why not, before we build the behemoth, try something smaller and faster?

References

1. Hou, X. et al. Large language models for SE: A systematic literature review. *TOSEM 33*, 8 (Sept. 2024).
2. Lustosa, A. et al. *More Signal: DRR for Better Optimizations of SE Tasks.* (2025); https://bit.ly/4ljC8oc.
3. Menzies, T. *BareLogic Python Source Code* (2024); https://bit.ly/3FPgoBz
4. Menzies, T. *MOOT= Many Multi-Objective Optimization Tests* (2024); https://bit.ly/4ewuRPP
5. Menzies, T., Owen, D., and Richardson, J. The strangest thing about software. *Computer* 40, 1 (2007).
6. Settles, D. Active learning literature survey. *Technical Report 1648.* U. Wisconsin-Madison Dept of CS (2009).
7. Tawosi, V., Moussa, R., and Sarro, F. Agile effort estimation: Have we solved the problem yet?. *IEEE Trans SE 49*, 4 (2023).
8. Zeming, L. et al. Survey on knowledge distillation for large language models. *ACM Trans. Int. Systems and Technology* (2024).

Tim Menzies (timm@ieee.org) is a professor of computer science at North Carolina State University in Raleigh, NC, USA.

Figure. Twenty runs of BareLogic on 63 multi-objective tasks. Histogram shows mean (1 – (most – b4.min)/(b4.mu – b4.min)). 'most' is the best example returned by BareLogic; 'b4' are the untreated examples; 'min' is the optimal example closest to heaven.


```
x = independent values                     | y = dependent values
-------------------------------------------|----------------------
Spout_wait, Spliters, Counters,  | Throughput+, Latency-
   10,           6,       17,    |     23075,    158.68
    8,           6,       17,    |     22887,    172.74
    9,           6,       17,    |     22799,    156.83
  [Skipped],    ...,      ...,            ...,     ...
10000,           1,       10,    |    460.81,    8761.6
10000,           1,       18,    |    402.53,    8797.5
10000,           1,        1,    |    310.06,    9421
```

- Evaluate y labels and sort (say) N=4 things
- While N < 24 (say)
  - N++
  - Build a 2 class bayes classifier with
    - Class 1 = sqrt(N) best
    - Class 2 = remaining N
  - Find unlabeled thing with most like(best) / like(rest).
  - Evaluate its y labels

31

Back to 2005: Birth of PROMISE Project & Early Success

Two-Part Vision:

1. **Annual conference** on predictor models in SE (to share results)
2. **Repository** of 100s of SE datasets: defect prediction, effort estimation, Github issue close time, bad smell detection

Growth Trajectory:

- Repository grew; moved to **Large Hadron Collider** (Seacraft, Zenodo)
- Research students ran weekly sprints scouring SE conferences
- **Gary Boetticher**, **Elaine Weyuker**, **Thomas Ostrand**, **Guenther Ruhe** joined steering committee → prestige for growth

PROMISE vs MSR:

- **MSR**: Gathering initial datasets (**Devanbu [Dev15]**)
- **PROMISE**: Post-collection analysis, data re-examination **[Rob10]**

Early Results:

- Other areas struggled with reproducibility, while we swam in data
- Papers applied tool sets to COC81, JM1, XALAN, DESHARNIS etc
- First decade: Numerous successful papers using consistent data re-examination

The 2007 Paper's Core Contribution

Research Question: Can data mining algorithms learn software defect predictors from static code attributes?

Why This Matters:

- *"Software quality assurance budgets are finite while assessment effectiveness increases exponentially with effort"* **[Fu16]**
- *"Software bugs are not evenly distributed across a project"* **[Ham09]**, **[Ost04]**, **[Mis11]**
- Defect predictors suggest where to focus expensive methods

Counter-Arguments Addressed:

1. *"Specific metrics matter"* (1990s heated debates: McCabe vs Halstead)
2. *"Static code attributes do not matter"* (**Fenton & Pfleeger**, **Shepperd & Ince**)

Menzies's 1st Law: Specific metrics do not matter

1st Law: *"Specific metrics do not always matter in all data sets. Rather, different projects have different best metrics."*

Supporting Evidence:

- Feature pruning experiment on **3 dozen metrics across 7 datasets**
- Results: Pruning selected just **2-3 attributes per dataset**
- **No single attribute** selected by majority of datasets
- Different projects preferred different metrics (McCabe vs Halstead vs lines of code)
- Theoretical debates of 1990s (metric X vs metric Y) proven empirically unfounded

Menzies's Corollary:

"To mine SE data, gather all that can be collected (cheaply) then apply data pruning to discard irrelevancies."

Practical Impact:

- Changed SE data mining methodology from "careful metric selection" to "gather everything, prune later"

2nd Law: *“Static code attributes do matter. Individually, they may be weak indicators. But when combined, they can lead to strong signals that outperform the state-of-the-art.”*

Support Evidence:

- **Fenton & Pfleeger**: Same functionality, different constructs → different measurements
- **Shepperd & Ince**: Static measures often “no more than proxy for lines of code”
- **Our Response**: Stress-tested these views by documenting baselines, then showing detectors from static attributes **much better** than baselines
- **Key Finding**: Multi-attribute models outperformed single-attribute models

Key Quote: *“Paradoxically, this paper will be a success if it is quickly superseded.”*

Unprecedented Success Metrics

Citation Impact:

- **2016**: Most cited paper (per month) in software engineering
- **2018**: 20% of Google Scholar Software Metrics IEEE TSE papers used PROMISE datasets **[Men07]**
- **Current**: 1924 citations (paper) + 1242 citations (repository)

Industrial Adoption:

- **Wan et al. [Wan20]**: 90%+ of 395 commercial practitioners willing to adopt defect prediction
- **Misirli et al. [Mis11]**: 87% defect prediction accuracy, 72% reduced inspection effort, 44% fewer post-release defects
- **Kim et al. [Kim15]**: Samsung Electronics API development
  - 0.68 F1 scores, reduced test case resources

Comparative Analysis with Static Tools

Rahman et al. [Rah14] Comparison:

- **Static analysis tools**: FindBugs, Jlint, PMD
- **Statistical defect prediction**: Logistic regression models
- **Result**: *"No significant differences in cost-effectiveness were observed"*

Critical Advantage:

- Defect prediction: Quick adaptation to new languages via lightweight parsers
- Static analyzers: Extensive modification required for new languages
- **Implication**: Broader applicability across programming ecosystems

Extended Applications:

- **Security vulnerabilities [Shi13]**
- **Resource allocation** for defect location **[Bir21]**
- **Proactive defect fixing [Kam16]**, **[LeG12]**, **[Arc11]**
- **Change-level/just-in-time prediction [Yan19]**, **[Kam13]**, **[Nay18]**, **[Ros15]**
- **Transfer learning** across projects **[Kri19]**, **[Nam18]**
- **Hyperparameter optimization [Agr18]**, **[Che18]**, **[Fu17]**, **[Tan16]**

Research Evolution:

- From binary classification to multi-objective optimization
- From release-level to line-level prediction (**Pornprasit et al. [Por23]** - TSE Best Paper 2023)

The Four Phases of Repository Lifecycle

Phase Evolution:

1. *“Data? Good luck with that!”* - Resistance and skepticism
2. *“Okay, maybe it’s not completely useless.”* - Grudging acknowledgment
3. *“This is the gold standard now.”* - Required baseline, field norms
4. *“A graveyard of progress.”* - Stifling creativity, outdated paradigms

The Problem:

- Decade 2: Continued use of decades old data e.g. COC81 (1981), DESHARNIS (1988), JM1 (2004), XALAN (2010)
- **Editorial Policy Change**: Automated Software Engineering journal now desk-rejects papers based on 2005 datasets

Menzies's 3rd Law & Transfer Learning

3rd law: *"Turkish toasters can predict for errors in deep space satellites."*

Supporting Evidence:

- **Transfer learning research [Tur09]**: Models from **Turkish white goods** successfully predicted errors in **NASA systems**
- Expected: Complex multi-dimensional transforms mapping attributes across domains
- **Reality**: Simple nearest neighboring between test and training data worked perfectly
- **Implication**: *"Many distinctions made about software are spurious and need to be revisited"*

Broader Transfer Learning Success:

- Cross-domain prediction often works better than expected
- Suggests universal patterns in software defect manifestation
- Questions assumptions about domain-specific modeling requirements

4th Law: *"For SE, the best thing to do with most data is to throw it away."*

Supporting Evidence:

- **Chen, Kocaguneli, Tu, Peters, and Xu et al.** findings across multiple prediction tasks:
  - **Github issue close time**: Ignored 80% of data labels **[Che19]**
  - **Effort estimation**: Ignored 91% of data **[Koc13]**
  - **Defect prediction**: Ignored 97% of data **[Pet15]**
  - **Some tasks**: Ignored 98-100% of data **[Che05]**
- **Startling result**: Data sets with thousands of rows modeled with just **few dozen samples [Men08]**

Theoretical Explanations:

- **Power laws** in software data **[Lin15]**
- **Large repeated structures** in SE projects **[Hin12]**
- **Manifold assumption** and **Johnson-Lindenstrauss lemma [Zhu05]**, **[Joh84]**

Caveat: Applies to regression, classification, optimization

- generative tasks may still need massive data

5th law: *"Bigger is not necessarily better."*

Supporting Evidence - LLM Hype Analysis:

- **Systematic review [Hou24]**: 229 SE papers using Large Language Models
- **Critical finding**: Only **13/229 around 5%** compared LLMs to other approaches
- *"Methodological error"* - other PROMISE-style methods often better/faster **[Gri22]**, **[Som24]**, **[Taw23]**, **[Maj18]**

Trading Off Complexity:

- Scalability vs. privacy vs. performance **[Lin24]**, **[Fu17]**
- Often simpler methods provide better cost-effectiveness
- **Personal Pattern**: *"Often, I switch to the simpler."* **[Agr21]**, **[Tan16]**, **[Fu16]**

6th Law: *"Data quality matters less than you think."*

Supporting Research:

- **Shepperd et al. [She13]**: Found numerous PROMISE data quality issues
  - Repeated rows, illegal attributes, inconsistent formats
  - **Critical gap**: Never tested if quality issues decreased predictive power

Our Experiment:

- Built **mutators** that injected increasing amounts of their quality issues into PROMISE defect datasets
- **Startling result**: Performance curves remained **flat** despite increased quality problems
- **Implication**: *"There is such a thing as too much care"* in data collection

Practical Impact:

- Effective predictions possible from seemingly dirty data
- Questions excessive data cleaning efforts in SE research
- Balance needed: careful collection without over-engineering

Menzies's 7th Law: Dumb sht*t, works

7th Law: *"Bad learners can make good conclusions."*

Supporting Evidence:

- **Nair et al. [Nai17]**: CART trees built for multi-objective optimization
- **Key finding**: Models that **predicted poorly** could still **rank solutions effectively**
- Could be used to prune poor configurations and find better ones
- **Implication**: Algorithms shouldn't aim for predictions but offer **weak hints** about project data

Application of bad leaners: ultra-low cost active learning

https://timm.fyi/assets/pdf/cacm25.pdf

[CACM'25: Menzies, Compact AI]

DOI:10.1145/3746057

The Case for Compact AI

A reader response to recent largesse of large language modeling material.

READING THE MARCH 2025 *Communications* issue, it struck me how many articles assume large language models (LLMs) are the inevitable and best future path for artificial intelligence (AI). Here, I encourage readers to question that assumption.

To be clear: I use LLMs—a lot—for solo and tactical tasks such as condensing my arguments into this editorial response. But for strategic tasks that might be critiqued externally, I need other tools that are faster, simpler, and whose reasoning can be explained and audited. So while I do not want to replace LLMs, I want to ensure we are also supporting and exploring alternatives.

In software engineering (SE), very few researchers explore alternatives to LLMs. A recent systematic review found only 5% of hundreds of SE LLM papers considered alternatives.[1] This is a major methodological mistake that ignores simpler and faster methods. For instance, UCL researchers found SVM+TF-IDF methods vastly outperformed standard "Big AI" for effort estimation (100 times faster, with greater accuracy).[7]

In SE, one reason for asking "if not LLM, then what?" is that software often exhibits "funneling": that is, despite internal complexity, software behavior converges to few outcomes, enabling simpler reasoning.[2,5] Funneling explains how my "BareLogic"[3] active learner can build models using very little data for (for example) 63 SE multi-objective optimization tasks from the MOOT repository.[4] These tasks are quite diverse and include software process decisions, optimizing configuration parameters, and tuning learners for better analytics. Successful MOOT modeling results in better advice for project managers, better control of software options, and enhanced analytics from learners that are better tuned to the local data.

Obtaining state-of-the-art results can be achieved with smarter questioning, not planetary-scale computation.

MOOT includes hundreds of thousands of examples with up to 1,000 settings. Each example is labeled with up to five effects. In practice, obtaining labels is slow, expensive, and error-prone. Hence, the task of active learners such as BareLogic is to find the best example(s), after requesting the least number of labels.[6] To do this, BareLogic labels $N = 4$ random examples, then:

1. Scores and sorts labeled examples by "distance to heaven" (where "heaven" is the ideal target for optimization, for example, weight=0, mpg=max).
2. Splits the sort into $\sqrt{N}$ *best* and $N - \sqrt{N}$ *rest* examples.
3. Trains a two-class Bayes classifier on the *best* and *rest* sets.
4. Finds the most "best" unlabeled example via $\arg\max_x(\log(\text{like}(best \mid X)) - \log(\text{like}(rest \mid X)))$
5. Labels X, then increments N.
6. If $N <$ Stop, go to step 1. Else return the top-ranked labeled example and a regression tree built from the N-labeled examples.

BareLogic was written for teaching purposes as a simple demonstrator of active learning. But in a result consistent with "funneling," this quick-and-dirty tool achieves near-optimal results using a handful of labels. As shown by the histogram, right-hand-side of the figure here, across 63 tasks. Eight labels yielded 62% of the optimal result; 16 labels reached nearly 80%, 32 labels approached 90% optimality, 64 labels barely improves on 32 labels, and so forth.

The lesson here is that obtaining state-of-the-art results can be achieved with smarter questioning, not planetary-scale computation. Active learning addresses many common LLM concerns such as slow training times, excessive energy needs, esoteric hardware requirements, testability, reproducibility, and explainability. The accompanying figure was created without billions of parameters. Active learners need no vast preexisting knowledge or massive datasets, avoiding the colossal energy and specialized hardware demands of large-scale AI. Further, unlike LLMs where testing is slow and often irreproducible, BareLogic's Bayesian active learning is fast (for example, for 63 tasks and 20 repeated trials, the figure here was generated in three minutes on a standard laptop). Most importantly, active learning fosters human-AI partnership.

Unlike opaque LLMs, BareLogic's results are explainable via small labeled sets (for example, $N = 32$). Whenever a label is required, humans can understand and guide the reasoning. The resulting tiny regression tree models offer concise, effective, and generalizable insights.

Active learning provides a compelling alternative to sheer scale in AI. Its ability to deliver rapid, efficient, and transparent results fundamentally questions the "bigger is better" assumption dominating current thinking about AI. It tells us that intelligence requires more than just size.

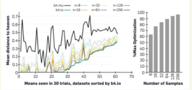
I am not the only one proposing weight loss for AI. The success of LLM distillation (shrinking huge models for specific purposes[8]) shows that giant models are not always necessary. Active learning pushes this idea even further, showing that leaner, smarter modeling can achieve great results. So why not, before we build the behemoth, try something smaller and faster?

References

1. Hou, X. et al. Large language models for SE: A systematic literature review. *TOSEM 33*, 8 (Sept. 2024).
2. Lustosa, A. et al. *More Signal: DRR for Better Optimizations of SE Tasks.* (2025); https://bit.ly/4ljC8oc.
3. Menzies, T. *BareLogic Python Source Code* (2024); https://bit.ly/3FPgoBz
4. Menzies, T. *MOOT= Many Multi-Objective Optimization Tests* (2024); https://bit.ly/4ewuRPP
5. Menzies, T., Owen, D., and Richardson, J. The strangest thing about software. *Computer 40* (2007).
6. Settles, D. Active learning literature survey. *Technical Report 1648.* U. Wisconsin-Madison Dept of CS (2009).
7. Tawosi, V., Moussa, R., and Sarro, F. Agile effort estimation: Have we solved the problem yet?. *IEEE Trans SE 49*, 4 (2023).
8. Zeming, L. et al. Survey on knowledge distillation for large language models. *ACM Trans. Int. Systems and Technology* (2024).

Tim Menzies (timm@ieee.org) is a professor of computer science at North Carolina State University in Raleigh, NC, USA.

Figure. Twenty runs of BareLogic on 63 multi-objective tasks. Histogram shows mean (1 – (most – b4.min)/(b4.mu – b4.min)). 'most' is the best example returned by BareLogic; 'b4' are the untreated examples; 'min' is the optimal example closest to heaven.



x = independent values			y = dependent values	
Spout_wait,	Spliters,	Counters,	Throughput+,	Latency-
10,	6,	17,	23075,	158.68
8,	6,	17,	22887,	172.74
9,	6,	17,	22799,	156.83
[Skipped],	...,	...,	...,	...
10000,	1,	10,	460.81,	8761.6
10000,	1,	18,	402.53,	8797.5
10000,	1,	1,	310.06,	9421

- Evaluate y labels and sort (say) N=4 things
- While N < 24 (say)
  - N++
  - Build a 2 class bayes classifier with
    - Class 1 = sqrt(N) best
    - Class 2 = remaining N
  - Find unlabeled thing with most like(best) / like(rest).
  - Evaluate its y labels

31

Menzies's 8th Law: Mud, rules

8th Law: *"Science has mud on the lens."*

Supporting Evidence:

- **Hyperparameter optimization** lessons **[Agr21]**, **[Tan16]**, **[Fu16]** on PROMISE data
- Data mining conclusions **changeable in an afternoon** by grad student with sufficient CPU
- **Critical Questions**: Are all conclusions brittle? How build scientific community on such basis?
- **Where are stable conclusions** for building tomorrow's ideas?

?Bayesian Approach Needed: Address uncertainty quantification and robust foundations

Menzies's 9th Law & Simplicity Challenge

9th Law: *"Many hard SE problems, aren't."*

Supporting Philosophy:

- **Cohen's Straw Man Principle [Coh95]**: *"Supposedly sophisticated methods should be benchmarked against seemingly stupider ones"*

Personal Experience Pattern:

- *"Whenever I checked a supposedly sophisticated method against a simpler one, there was always something useful in the simpler"*
- *"Often, I switch to the simpler."* **[Agr21]**, **[Tan16]**, **[Fu16]**

Important Caveat:

- **Not all SE problems can/should be simplified** (safety-critical; generative);
- *"Just because some tasks are hard, does not mean all tasks are hard"*

Challenge to Community: *"Have we really checked what is really complex and what is really very simple?"*
Current Focus: Minimal data approaches - landscape analysis **[Che19]**, **[Lus24]**, surrogate learning **[Nai20]**, active learning **[Kra15]**, **[Yu18]**

PROMISE Revival Strategy (**Gema Rodríguez-Pérez**):

- Data sharing now expected for almost all SE papers
- PROMISE must differentiate: accept higher quality datasets
- Focus on enhancing current data space, conducting quality evaluations

Steffen Herbold's Caution:

- Early PROMISE: Collections of metrics (not raw data)
- MSR shift: Raw data + fast tools (e.g., PyDriller, GHtorrent)
- **Risk**: *"Little curation, little validation, often purely heuristic data collection without quality checks"* **[Her22]**

Modern Data Access: 1100+ recent Github projects **[Xia22]**, CommitGuru **[Ros15]**

Current “Hot” Research Directions

Contemporary Approaches:

- **DeepLineDP** (**Pornprasit et al. [Por23]**): Deep learning for line-level defect prediction (TSE Best Paper 2023)
- **Model interpretability**: Growing research focus **[Tan21]**
- **Multi-objective optimization**: Hyperparameter selection **[Xia22]**, unfairness reduction **[Cha20]**, **[Alv23]**

Optimize CPU-Intensive Algorithms:

- MaxWalkSat **[Men09]**
- Simulated annealing **[Men02]**, **[Men07]**
- Genetic algorithms

Minimal Data Approaches:

- How much can be achieved with as little data as possible?
- Suspicion of “large number of good quality labels” assumption

Cross-Domain Success [Tur09]:

- **Turkish white goods** → **NASA systems** error prediction
- Expected: Complex multi-dimensional transforms
- **Reality**: Simple nearest neighboring between test and training data

Implication: *"Many distinctions made about software are spurious and need to be revisited"*

Power Laws & Repeated Structures:

- **Lin & Whitehead [Lin15]**: Fine-grained code changes follow power laws
- **Hindle et al. [Hin12]**: Software naturalness - large repeated structures
- **Result**: Thousands of rows modeled with few dozen samples **[Men08]**

Key Takeaways & Community Call-to-Action

Lessons Learned:

1. **Open science communities** can be formed by publishing baseline + data + scripts
2. **Reproducible research** drives field advancement when embraced collectively
3. **Simple solutions** often outperform sophisticated ones
4. **Data quality** matters less than expected for predictive tasks
5. **Transfer learning** works across surprisingly diverse domains

Call-to-Action:

- *"Have we really checked what is really complex and what is really very simple?"*
- Challenge assumptions about problem complexity
- Benchmark sophisticated methods against simpler alternatives
- Focus on stable, reproducible conclusions

References

[Agr18]: A. Agrawal and T. Menzies, "Is better data better than better data miners?: On the benefits of tuning smote for defect prediction," in *Proc. IST*, ACM, 2018, pp. 1050–1061.

[Agr21]: A. Agrawal *et al.*, "How to"DODGE" complex software analytics?" *IEEE Trans. Softw. Eng.*, vol. 47, no. 10, pp. 2182–2194, Oct. 2021.

[Alv23]: L. Alvarez and T. Menzies, "Don't lie to me: Avoiding malicious explanations with STEALTH," *IEEE Softw.*, vol. 40, no. 3, pp. 43–53, May/Jun. 2023.

[Cha20]: J. Chakraborty *et al.*, "Fairway: A way to build fair ML software," in *Proc. FSE*, 2020, pp. 654–665.

[Che19]: J. Chen *et al.*, " 'Sampling' as a baseline optimizer for search-based software engineering," *IEEE Trans. Softw. Eng.*, vol. 45, no. 6, pp. 597–614, Jun. 2019.

[Coh95]: P. R. Cohen, *Empirical Methods for Artificial Intelligence*, Cambridge, MA: MIT Press, 1995.

[Dev15]: P. Devanbu, "Foreword," in *Sharing Data and Models in Software Engineering*, T. Menzies *et al.*, Eds. San Mateo, CA: Morgan Kaufmann, 2015, pp. vii–viii.

[Fu16]: W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Inform. Softw. Technol.*, vol. 76, pp. 135–146, 2016.

References (More)

[Gon23]: J. M. Gonzalez-Barahona and G. Robles, “Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories,” *Inf. Softw. Technol.*, vol. 164, 2023, Art. no. 107318.
[Gri22]: L. Grinsztajn, E. Oyallon, and G. Varoquaux, “Why do tree-based models still outperform deep learning on typical tabular data?” in *Proc. NeurIPS*, 2022, pp. 507–520.
[Ham09]: M. Hamill and K. Goseva-Popstojanova, “Common trends in software fault and failure data,” *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 484–496, Jul./Aug. 2009.
[Has08]: A. E. Hassan, “The road ahead for mining software repositories,” *Frontiers Softw. Maintenance*, pp. 48–57, 2008.
[Her22]: S. Herbold *et al.*, “Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection,” *Empirical Softw. Eng.*, vol. 27, no. 2, p. 42, 2022.
[Hou24]: X. Hou *et al.*, “Large language models for software engineering: A systematic literature review,” *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024.
[Kam13]: Y. Kamei *et al.*, “A large-scale empirical study of just-in-time quality assurance,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.
[Kim15]: M. Kim *et al.*, “REMI: Defect prediction for efficient api testing,” in *Proc. FSE*, ACM, 2015, pp. 990–993.

[Kri19]: R. Krishna and T. Menzies, “Bellwethers: A baseline method for transfer learning,” *IEEE Trans. Softw. Eng.*, vol. 45, no. 11, pp. 1081–1105, Nov. 2019.
[Men07]: T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.
[Men24]: T. Menzies, “A brief note, with thanks, on the contributions of guenther ruhe,” *Inf. Softw. Technol.*, vol. 173, 2024, Art. no. 107486.
[Men25]: T. Menzies, “Retrospective: Data Mining Static Code Attributes to Learn Defect Predictors,” *IEEE Trans. Softw. Eng.*, 2025.
[Mis11]: A. T. Misirli, A. Bener, and R. Kale, “AI-based software defect predictors: Applications and benefits in a case study,” *AI Mag.*, vol. 32, no. 2, pp. 57–68, 2011.
[Nai17]: V. Nair *et al.*, “Using bad learners to find good configurations,” in *Proc. 11th Joint Meeting FSE*, ACM, 2017, pp. 257–267.
[Nam18]: J. Nam *et al.*, “Heterogeneous defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 874–896, Sep. 2018.
[Ost04]: T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Where the bugs are,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 29, no. 4, pp. 86–96, 2004.
[Por23]: C. Pornprasit and C. K. Tantithamthavorn, “DeepLineDP: Towards a deep learning approach for line-level defect prediction,” *IEEE Trans. Softw. Eng.*, vol. 49, no. 1, pp. 84–98, Jan. 2023.
[Rah14]: F. Rahman *et al.*, “Comparing static bug finders and statistical prediction,” in *Proc. ICSE*, ACM, 2014, pp. 424–434.

References (Last)

[Rob10]: G. Robles, "Replicating MSR: A study of the potential replicability of papers published in the mining software repositories proceedings," in *7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, IEEE Press, 2010, pp. 171–180.

[Ros15]: C. Rosen, B. Grawi, and E. Shihab, "Commit guru: Analytics and risk prediction of software commits," in *Proc. ESEC/FSE*, 2015, pp. 966–969.

[She13]: M. Shepperd *et al.*, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013.

[Shi13]: Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Softw. Eng.*, vol. 18, pp. 25–59, 2013.

[Tan16]: C. Tantithamthavorn *et al.*, "Automated parameter optimization of classification techniques for defect prediction," in *ICSE'16*, 2016, pp. 321–332.

[Tur09]: B. Turhan *et al.*, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, vol. 14, pp. 540–578, Jan. 2009.

[Wan20]: Z. Wan *et al.*, "Perceptions, expectations, & challenges in defect prediction," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1241–1266, Nov. 2020.

[Xia22]: T. Xia *et al.*, "Sequential model optimization for software effort estimation," *IEEE Trans. Softw. Eng.*, vol. 48, no. 6, pp. 1994–2009, Jun. 2022.

[Yan19]: M. Yan *et al.*, "Automating change-level self-admitted technical debt determination," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1211–1229, Dec. 2019.

References

[Agr18]: A. Agrawal and T. Menzies, “Is better data better than better data miners?: On the benefits of tuning smote for defect prediction,” in *Proc. IST*, ACM, 2018, pp. 1050–1061.
[Agr21]: A. Agrawal *et al.*, “How to”DODGE” complex software analytics?” *IEEE Trans. Softw. Eng.*, vol. 47, no. 10, pp. 2182–2194, Oct. 2021.
[Alv23]: L. Alvarez and T. Menzies, “Don’t lie to me: Avoiding malicious explanations with STEALTH,” *IEEE Softw.*, vol. 40, no. 3, pp. 43–53, May/Jun. 2023.
[Cha20]: J. Chakraborty *et al.*, “Fairway: A way to build fair ML software,” in *Proc. FSE*, 2020, pp. 654–665.
[Che19]: J. Chen *et al.*, “ ‘Sampling’ as a baseline optimizer for search-based software engineering,” *IEEE Trans. Softw. Eng.*, vol. 45, no. 6, pp. 597–614, Jun. 2019.
[Coh95]: P. R. Cohen, *Empirical Methods for Artificial Intelligence*, Cambridge, MA: MIT Press, 1995.
[Dev15]: P. Devanbu, “Foreword,” in *Sharing Data and Models in Software Engineering*, T. Menzies *et al.*, Eds. San Mateo, CA: Morgan Kaufmann, 2015, pp. vii–viii.
[Fu16]: W. Fu, T. Menzies, and X. Shen, “Tuning for software analytics: Is it really necessary?” *Inform. Softw. Technol.*, vol. 76, pp. 135–146, 2016.

References (More)

[Gon23]: J. M. Gonzalez-Barahona and G. Robles, "Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories," *Inf. Softw. Technol.*, vol. 164, 2023, Art. no. 107318.
[Gri22]: L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Proc. NeurIPS*, 2022, pp. 507–520.
[Ham09]: M. Hamill and K. Goseva-Popstojanova, "Common trends in software fault and failure data," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 484–496, Jul./Aug. 2009.
[Has08]: A. E. Hassan, "The road ahead for mining software repositories," *Frontiers Softw. Maintenance*, pp. 48–57, 2008.
[Her22]: S. Herbold *et al.*, "Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection," *Empirical Softw. Eng.*, vol. 27, no. 2, p. 42, 2022.
[Hou24]: X. Hou *et al.*, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024.
[Kam13]: Y. Kamei *et al.*, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2013.
[Kim15]: M. Kim *et al.*, "REMI: Defect prediction for efficient api testing," in *Proc. FSE*, ACM, 2015, pp. 990–993.

References (Yet More)

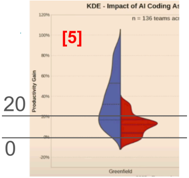
[Kri19]: R. Krishna and T. Menzies, "Bellwethers: A baseline method for transfer learning," *IEEE Trans. Softw. Eng.*, vol. 45, no. 11, pp. 1081–1105, Nov. 2019.
[Men07]: T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007.
[Men24]: T. Menzies, "A brief note, with thanks, on the contributions of guenther ruhe," *Inf. Softw. Technol.*, vol. 173, 2024, Art. no. 107486.
[Men25]: T. Menzies, "Retrospective: Data Mining Static Code Attributes to Learn Defect Predictors," *IEEE Trans. Softw. Eng.*, 2025.
[Mis11]: A. T. Misirli, A. Bener, and R. Kale, "AI-based software defect predictors: Applications and benefits in a case study," *AI Mag.*, vol. 32, no. 2, pp. 57–68, 2011.
[Nai17]: V. Nair *et al.*, "Using bad learners to find good configurations," in *Proc. 11th Joint Meeting FSE*, ACM, 2017, pp. 257–267.
[Nam18]: J. Nam *et al.*, "Heterogeneous defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 9, pp. 874–896, Sep. 2018.
[Ost04]: T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," *ACM SIGSOFT Softw. Eng. Notes*, vol. 29, no. 4, pp. 86–96, 2004.
[Por23]: C. Pornprasit and C. K. Tantithamthavorn, "DeepLineDP: Towards a deep learning approach for line-level defect prediction," *IEEE Trans. Softw. Eng.*, vol. 49, no. 1, pp. 84–98, Jan. 2023.
[Rah14]: F. Rahman *et al.*, "Comparing static bug finders and statistical prediction," in *Proc. ICSE*, ACM, 2014, pp. 424–434.

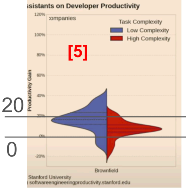
References (Last)

[Rob10]: G. Robles, "Replicating MSR: A study of the potential replicability of papers published in the mining software repositories proceedings," in *7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, IEEE Press, 2010, pp. 171–180.
[Ros15]: C. Rosen, B. Grawi, and E. Shihab, "Commit guru: Analytics and risk prediction of software commits," in *Proc. ESEC/FSE*, 2015, pp. 966–969.
[She13]: M. Shepperd *et al.*, "Data quality: Some comments on the NASA software defect datasets," *IEEE Trans. Softw. Eng.*, vol. 39, no. 9, pp. 1208–1215, Sep. 2013.
[Shi13]: Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?" *Empirical Softw. Eng.*, vol. 18, pp. 25–59, 2013.
[Tan16]: C. Tantithamthavorn *et al.*, "Automated parameter optimization of classification techniques for defect prediction," in *ICSE'16*, 2016, pp. 321–332.
[Tur09]: B. Turhan *et al.*, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Softw. Eng.*, vol. 14, pp. 540–578, Jan. 2009.
[Wan20]: Z. Wan *et al.*, "Perceptions, expectations, & challenges in defect prediction," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1241–1266, Nov. 2020.
[Xia22]: T. Xia *et al.*, "Sequential model optimization for software effort estimation," *IEEE Trans. Softw. Eng.*, vol. 48, no. 6, pp. 1994–2009, Jun. 2022.
[Yan19]: M. Yan *et al.*, "Automating change-level self-admitted technical debt determination," *IEEE Trans. Softw. Eng.*, vol. 45, no. 12, pp. 1211–1229, Dec. 2019.

Cause we need a better AI

- **Bubble** bursting in “big data” AI?
  - Unlike standard software, **exponential** costs per new user
    - Unless usage rate limited (bad for keeping new users)
    - **ChatGPT**: A mere 2% to 8% conversion **free** to paid users [2]
  - Established companies: 95% of AI apps not returning revenue [3]
  - Microsoft: Copilot costing Msoft $X00 per user [1]
- What’s **failing** [3]:
  - Support tools for groups, for negotiation
  - Integration into organizational workflows
- What’s **working**: support tools for individuals (e.g. Copilot)
  - But the improvements are modest : +-20% [5] or negative [4][6]





[1] https://www.youtube.com/watch?v=OYlQyPo-L4g AI Startups Are Bad Businesses, Sept 2025
[2] https://www.mckinsey.com/capabilities/quantumblack/our-insights/seizing-the-agentic-ai-advantage McKinsey report. , 2025
[3] https://mlq.ai/media/quarterly_decks/v0.1_State_of_AI_in_Business_2025_Report.pdf MIT NANDA, July 2025
[4] https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality, GitClear, 2024
[5] https://www.youtube.com/watch?v=tbDDYKRFjhk Does AI Boost Productivity? 2025
[6] https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/ METR July 2025

28