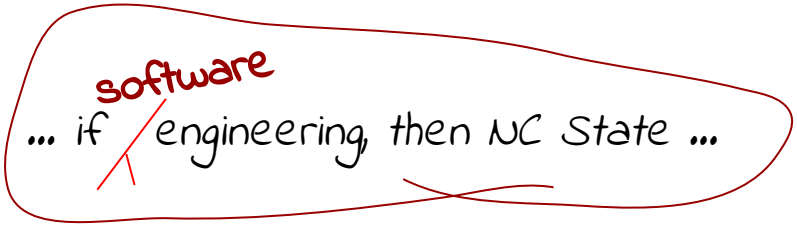


IEEE Fellows

Q: What's Wrong with Comp.Sci. Software? A: Absolutely Nothing (nearly)

Tim Menzies
IEEE Fellow
(prof, cs, se, ai, ncstate, usa)
timm@ieee.org
Feb 13, 2019

twitter: @timmenzies
home page: menzies.us
pre-prints: tiny.cc/menzies

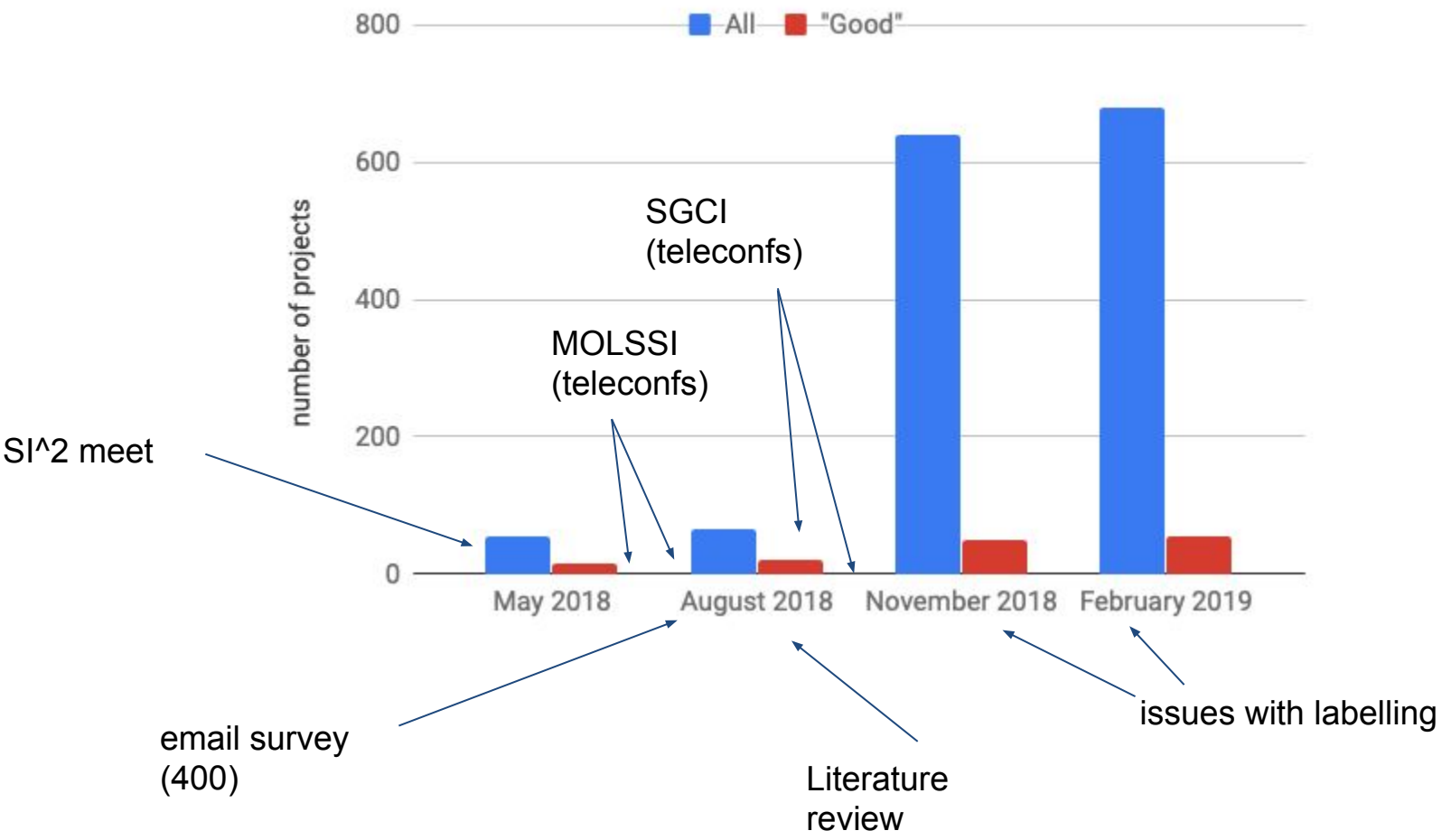
Sound bites

- Comput.Science projects are **orders of magnitude better** than convention code:
  - Higher ratio of "good code" (370 times)
  - Last much longer (e.g. decades, not years)

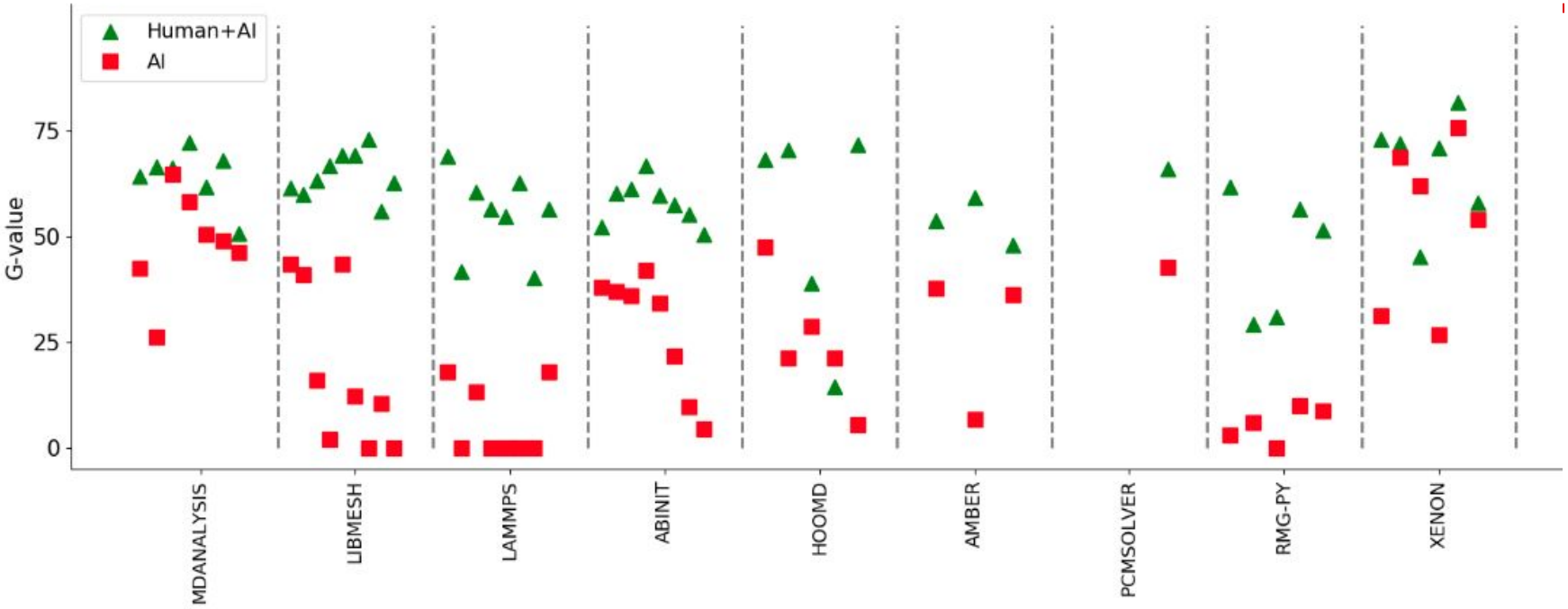
- We can **learn quality agents** for Comput.Science code:
  - Out-of-the-box, Standard empirical SE: not so good
  - But it can be successfully adapted

- Open area of research
  - **Effective Empirical SE** for Comput.Science

Data Collection



This talk's case study
Empirical SE methods tuned to Compute.Sci.



RED: standard Empirical SE
(perform badly on Compute.Sci. projects)

GREEN: After we made it better

New research direction:

- Novel empirical methods for Compute.Sci.

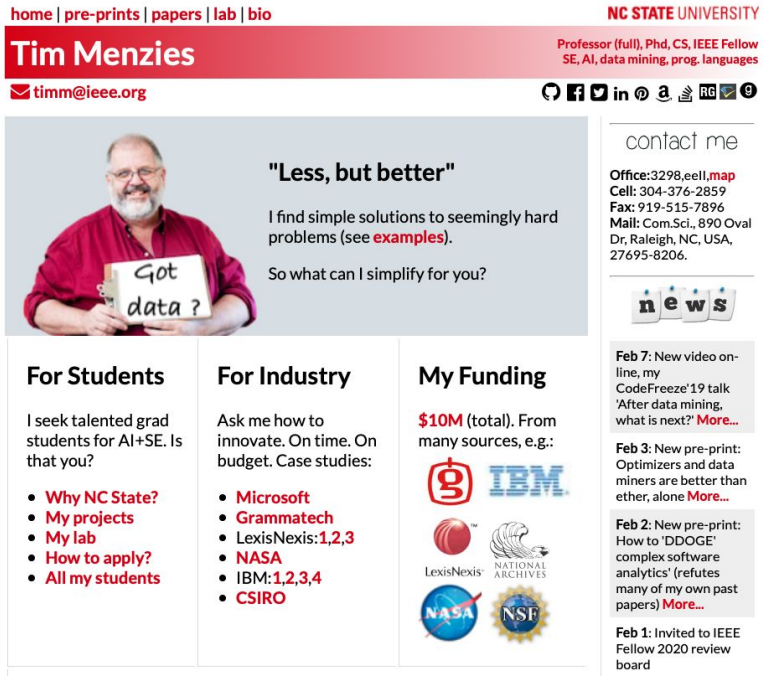
Compute.Sci

- **Compute.Sci.** = Computational Science software
  - Modeling Scientific Software Elements:
    - Often related to core physical process
  - Supporting scientific software integration (mash-ups)
  - Creating Scientific Software Innovation Institutes

- **Empirical Software Engineering** =
  - The derivation of repeated *interesting* patterns in software development projects
    - *Interesting* = insights we can use to improve software
  - Traditionally, qualitative
  - More and more, quantitative
    - Perhaps augmented with data mining algorithms

Me = menzies.us

Lab = ai4se.net



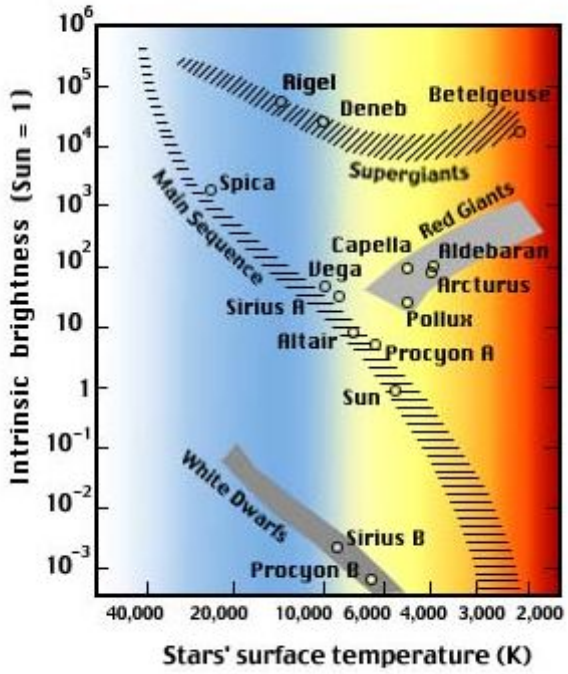
Lessons (1)

Software development can be studied to find predictable properties

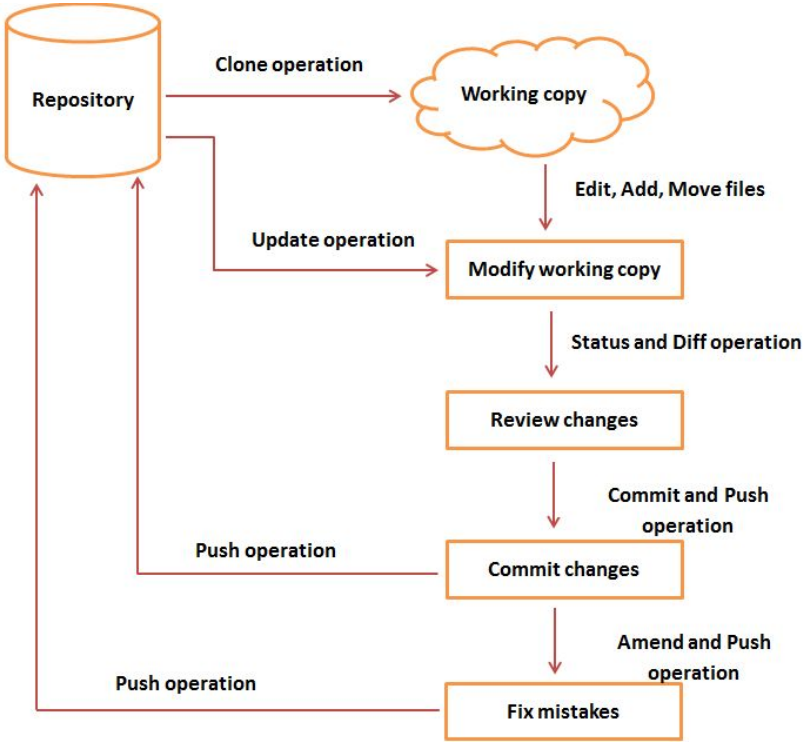
Introducing "Github"

28 million users. 57 million repositories.
Empirical SE heaven: a place to learn SE patterns

Stars have life cycles, predictable properties



So does software



Introducing "Github"

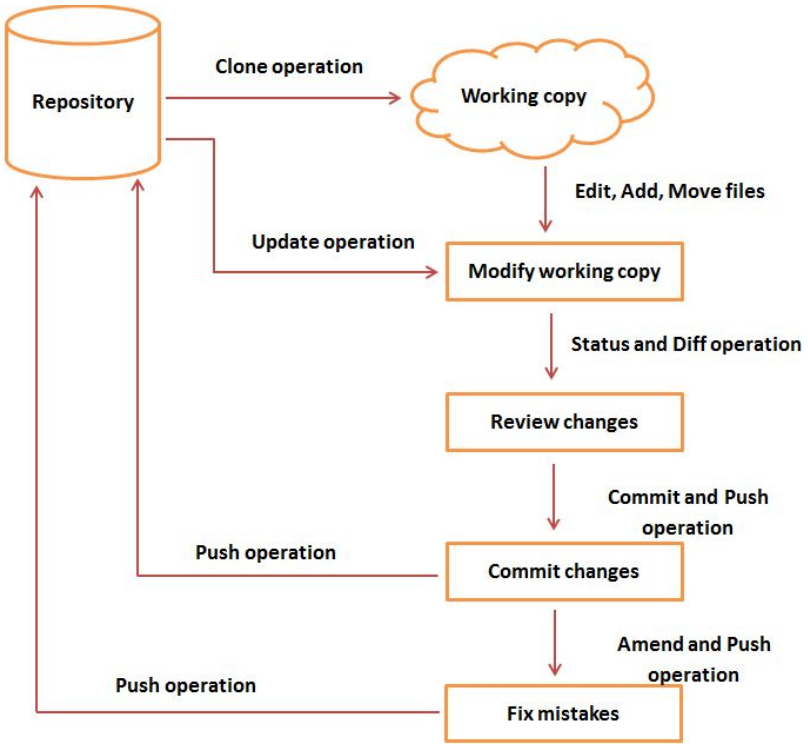
28 million users. 57 million repositories.
Empirical SE heaven: a place to learn SE patterns

Check	Condition
Personal purpose (# Developers)	> 7
Collaboration (Pull requests)	> 0
Issues	> 10
Releases	> 1
Commits	> 20
Duration	> 1 year

10,000 "good" projects

57,000,000/10,000 = 5700

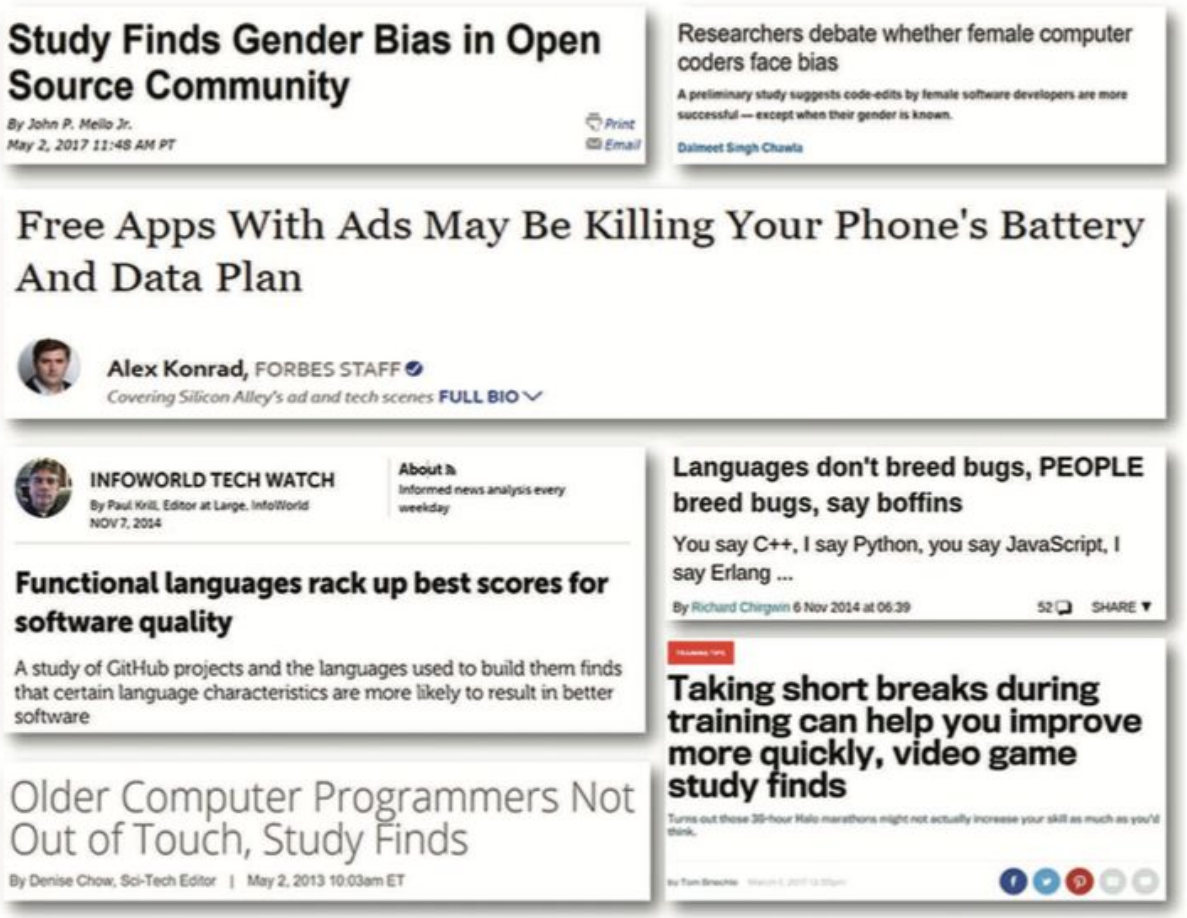
So does software



Q: What can we learn from Github?
A: Many, many things

Software analytics=

workflow that distills large amounts of low-value data into small chunks of very high-value data



e.g. Bugs live in clumps
Best bug predictor = where's the last one

- NASA software data: Most faults lie in a small proportion of the files.
- AT&T software data: about 80% of the defects come from 20% of the files
- So poke everything, everywhere
  - Rather, poke around, some
  - Where anything starts to fail,
  - Move in for a closer look

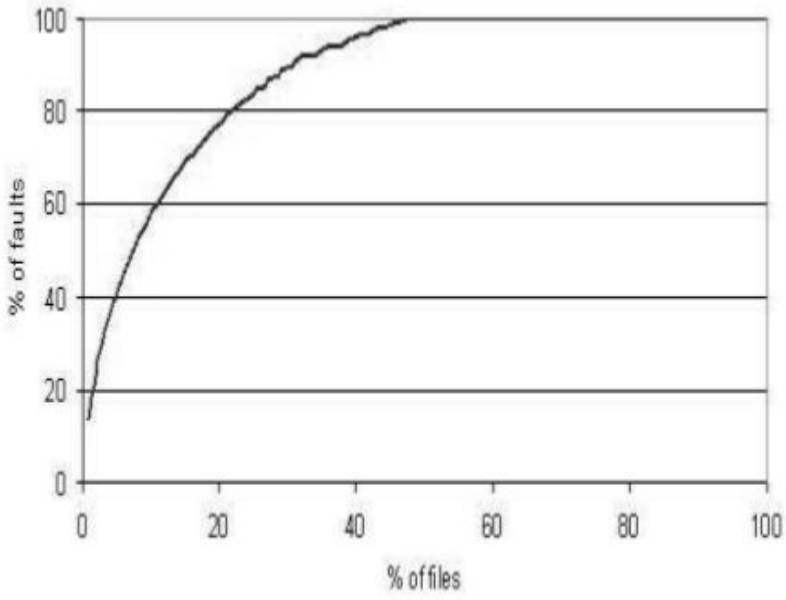


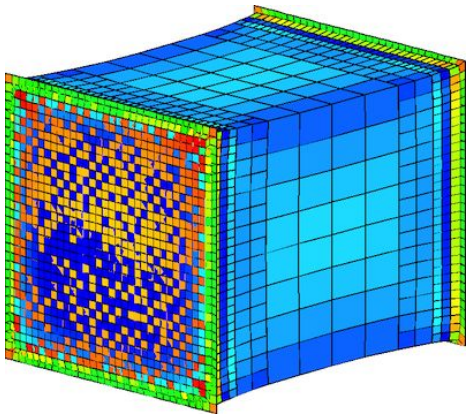
Fig. 4. Pareto diagram showing the percentage of files versus percentage of faults for GCC release 3.2.3.

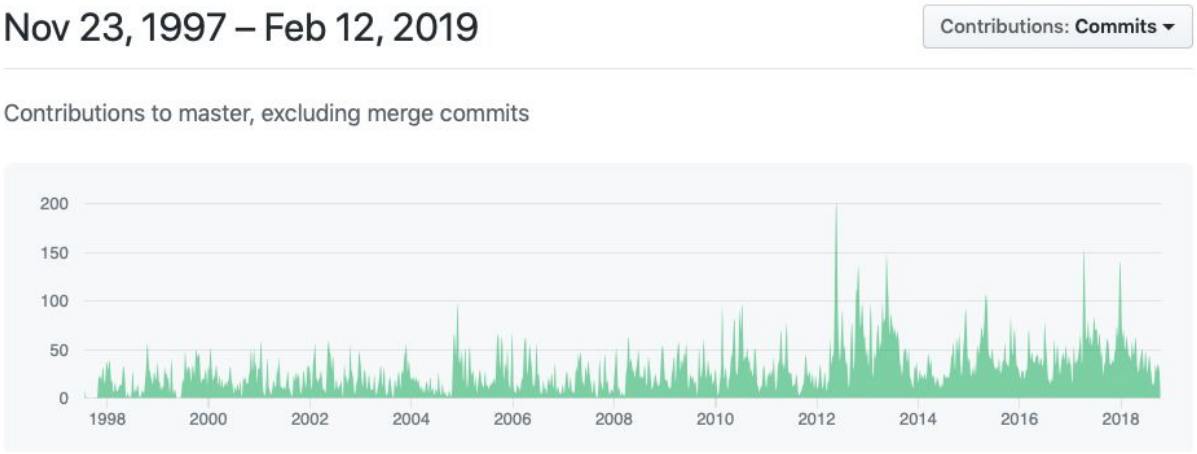
Lessons (2)

Compute.Scientists:
be proud of your software

Compute.Scientists: be proud of your software

- It is:
  - reasonable
  - controllable
  - improvable
  - maintainable, over decades
    - many examples of this; eg. the dealii finite elements toolkit





Many Compute.Sci. codes have been maintained, successfully, for years

- Slaps:
  - 16,000+ commits from 80 developers (since 2012)
- Trillions (is a more recent package)
  - 80,000 commits from over 200 developers.
- Elastic search (over 8 years old)
  - 40,000+ commits from over 1100 developers
- Dealii (maintained and extended since 1990)
  - 40,000+ commits from 100 active developers.

Some (E.g. dealii) larger (has more) longevity than many open source projects. When we talked to the developers of these 40+ packages (particularly, post-docs),

- we found developers well versed in current coding practices (Github, Travis, etc).

Many of those systems were written in modern languages (e.g. Python) or used modern programming tools (e.g. version control)

Sample of 678 Projects (for Compute.Sci. codes)

55 projects

- down selected from 678
- 678/55 = 12.5
- 370 times more common than standard SE

Check	Condition
Personal purpose (# Developers)	> 7
Collaboration (Pull requests)	> 0
Issues	> 10
Releases	> 1
Commits	> 20
Duration	> 1 year

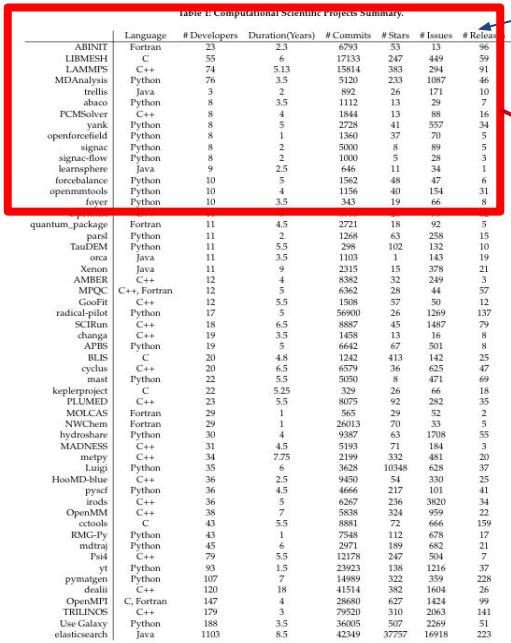


Table 1: Computational Scientific Projects Summary.

	Language	# Developers	Duration(Years)	# Commits	# Stars	# Issues	# Releases
ABINIT	Fortran	23	2.3	6793	53	13	96
LIBMESH	C	55	6	17133	247	449	59
LAMMPS	C++	74	5.13	15814	383	294	91
MDAnalysis	Python	76	3.5	5120	233	1087	46
trellis	Java	3	2	892	26	171	10
abaco	Python	8	3.5	1112	13	29	7
PCMSolver	C++	8	4	1844	13	88	16
yank	Python	8	5	2728	41	557	34
openforcefield	Python	8	1	1360	37	70	5
signac	Python	8	2	5000	8	89	5
signac-flow	Python	8	2	1000	5	28	3
learnsphere	Java	9	2.5	646	11	34	1
forcebalance	Python	10	5	1562	48	47	6
openmmtools	Python	10	4	1156	40	154	31
foyer	Python	10	3.5	343	19	66	8
quantum_package	Fortran	11	4.5	2721	18	92	5
parsl	Python	11	2	1268	63	258	15
TauDEM	Python	11	5.5	298	102	132	10
orca	Java	11	3.5	1103	1	143	19
Xenon	Java	11	9	2315	15	378	21
AMBER	C++	12	4	8382	32	249	3
MPQC	C++, Fortran	12	5	6362	28	44	57
GooFit	C++	12	5.5	1508	57	50	12
radical-pilot	Python	17	5	56900	26	1269	137
SCIRun	C++	18	6.5	8887	45	1487	79
changa	C++	19	3.5	1458	13	16	8
APBS	Python	19	5	6642	67	501	8
BLIS	C	20	4.8	1242	413	142	25
cyclus	C++	20	6.5	6579	36	625	47
mast	Python	22	5.5	5050	8	471	69
keplerproject	C	22	5.25	329	26	66	18
PLUMED	C++	23	5.5	8075	92	282	35
MOLCAS	Fortran	29	1	565	29	52	2
NWChem	Fortran	29	1	26013	70	33	5
hydroshare	Python	30	4	9387	63	1708	55
MADNESS	C++	31	4.5	5193	71	184	3
metpy	C++	34	7.75	2199	332	481	20
Luigi	Python	35	6	3628	10348	628	37
HooMD-blue	C++	36	2.5	9450	54	330	25
pyscf	Python	36	4.5	4666	217	101	41
irods	C++	36	5	6267	236	3820	34
OpenMM	C++	38	7	5838	324	959	22
cctools	C	43	5.5	8881	72	666	159
RMG-Py	Python	43	1	7548	112	678	17
mdtraj	Python	45	6	2971	189	682	21
Psi4	C++	79	5.5	12178	247	504	7
yt	Python	93	1.5	23923	138	1216	37
pymatgen	Python	107	7	14989	322	359	228
dealii	C++	120	18	41514	382	1604	26
OpenMPI	C, Fortran	147	4	28680	627	1424	99
TRILINOS	C++	179	3	79520	310	2063	141
Use Galaxy	Python	188	3.5	36005	507	2269	51
elasticsearch	Java	1103	8.5	42349	37757	16918	223

Caveat Emptor (buyer beware)

- Sample bias

- We report on the code we can access
  - Open source Github repositories
- So only a slice of Compute.Sci software
- That said, this sample is growing in size
  - For sociological reasons (see below)

"We don't get paid to write software"	• Well, actually, some of you do. • See the Gateway project
"You gotta say, the code's a mess"	• Yes, and no. • We find 687 projects of widely varying quality. • But in 55 "serious" projects, much that is exemplary
"We don't know much about SE"	• But your grads and postdocs do. And they know that post-NSF funding, the can get $X00,000/year jobs if they use state-of-the-art software tools. • Also, you seen the CERN software plans? Which run decades into the future? CERN maintains and plans it software better than Microsoft.
"Most of our code is bad."	• Compared to what? You maintainable usable software for years while much commercial code has a 2 year half-life.
"We don't use state of the art tools."	• In your best-of-breed you do.
"Most of our software is never used"	• Get used to it. Welcome to Darwinian selection

Lessons (3)

Empirical SE methods work for Compute.Sci. (after being adapted)

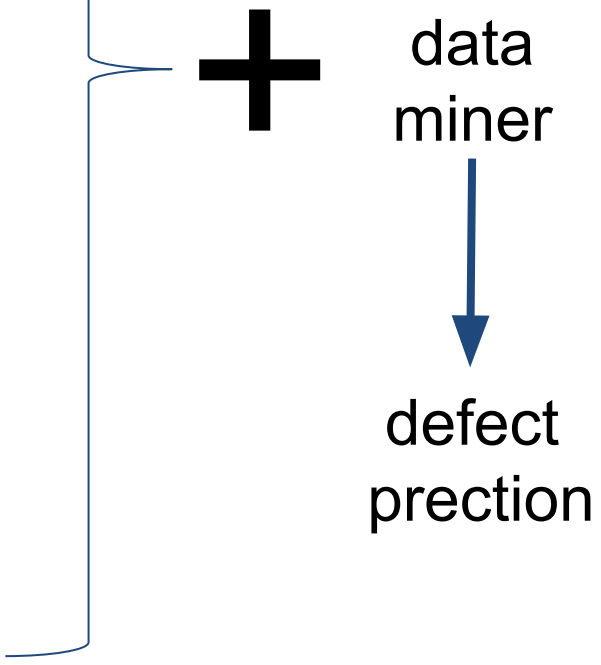
Sample projects (selected to cover a range of languages)

Fortran	**ABINIT**:	an atomic-scale simulation software suite
Python	**MDANALYSIS**:	analyze molecular dynamics trajectories generated from simulation packages
Python	**RMG-PY:**	Reaction Mechanism Generator, a tool for automatically generating chemical reaction mechanisms for modeling reaction systems including pyrolysis, combustion, atmospheric science, and more
Java	**XENON**:	A middleware abstraction library for compute and storage resources
C	**LIBMESH:**	numerical simulation of partial differential equations using arbitrary unstructured discretizations. Supports for adaptive mesh refinement (AMR) computations in parallel
C++	**PCMSOLVER**:	API , Polarizable Continuum Model
C++	**HOOMD**:	particle simulation toolkit. Hard particle Monte Carlo simulations of a variety of shape classes (and molecular dynamics simulations of particles)
C++	**AMBER**:	Fast, parallelized molecular dynamics trajectory data analysis
C++	**LAMMPS:**	Large-scale Atomic/Molecular Massively Parallel Simulator (maintained and developed by the Sandia National Laboratory).

9 Projects: Independent variables

Language agnostic

Dimension	Name	Definition
Diffusion	NS	Number of modified subsystems
	ND	Number of modified directories
	NF	Number of modified Files
	Entropy	Number of modified subsystems
Size	LA	Lines of code added
	LD	Lines of code deleted
	LT	Lines of code in a file before the changes
Purpose	FIX	Whether the change is bug-fixing?
History	NDEV	Number of developers that changed the modified files
	AGE	The average time interval between the last and the current change
	NUC	Number of unique changes to the modified files before
Experience	EXP	Developer experience
	REXP	Recent developer experience
	SEXP	Developer experience on a subsystem



Q: What is the dependent variable?
A1: Use a Keyword List

Look at project releases R1,R2,R3,....

- In release[i]
  - Label the comments that refer to errors?
  - What code was changed before that? (SZZ algorithm)
    - Mark that code as “defective”
  - Build a predictor for that target
- Apply that predictor to release[i+1]

Q: What is the dependent variable?
A1: Use a Keyword List

Look at project releases R1,R2,R3,....

- In release[i]
  - **Label the commits, commenting on errors**
  - What code was changed before that? (SZZ algorithm
    - Mark that code as "defective"
  - Build a predictor for that target
- Apply that predictor to release[i+1]

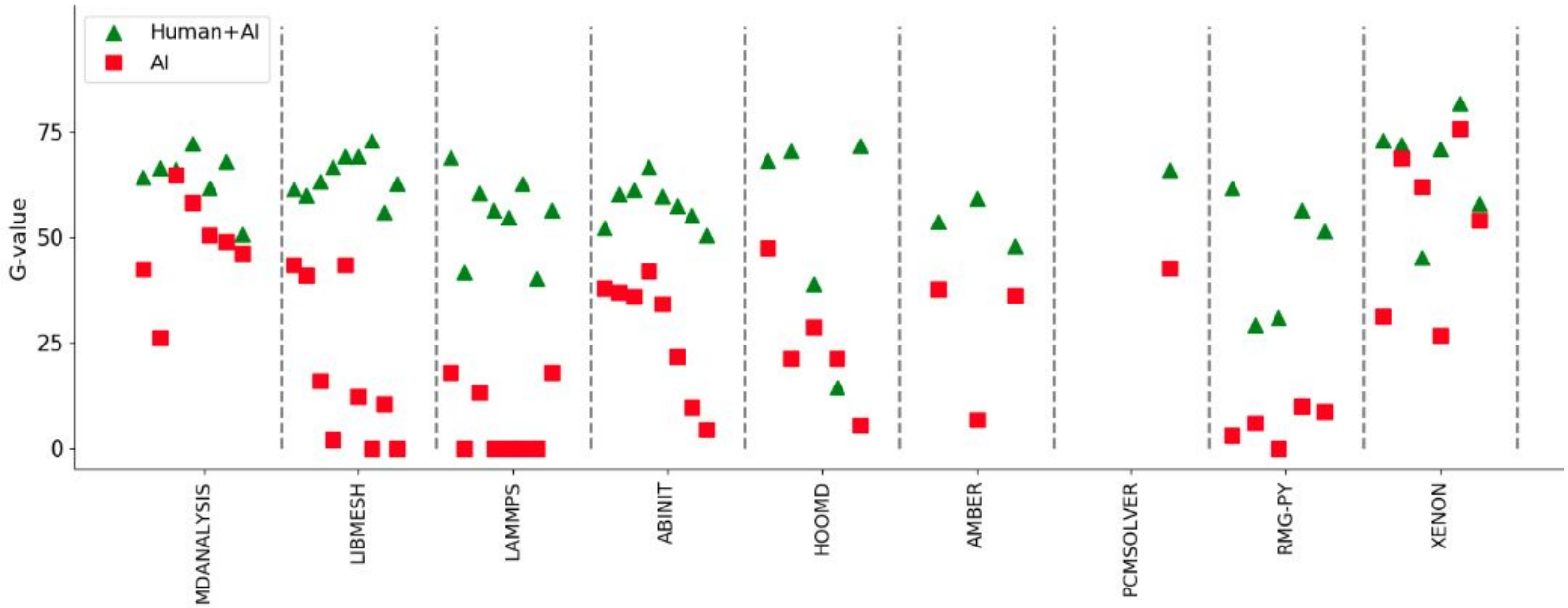
standard method

Corrective: active, against, already, bad, block, bug, build, call,case, catch, cause, character, compile, correctly, create, different,dump, error, except, exist, explicitly, fail, failure, fast, fix, format, good, hack, hard, help, init, instead, introduce, issue, lock, log,logic, look, merge, miss, null, oops, operation, operations,pass, previous, previously, probably, problem, properly, random,recent, request, reset, review, run, safe, set, similar, simplify, special,test, think, try, turn, valid, wait, warn, warning, wrong

Adaptive: active, add, additional, against, already, appro-priate, available, bad, behavior, block, build, call, case, catch,change, character, compatibility, compile, config, configura-tion, context, correctly, create, currently, default, differ-ent, documentation, dump, easier, except, exist, explicitly, fail,fast, feature, format, future, good, hack, hard, header, help,include, information, init, inline, install, instead, internal, in-troduce, issue, lock, log, logic, look, merge, method, necessary,new , old, operation, operations, pass, patch, previous, pre-viously, probably, properly, protocol provide, random, recent,release, replace ,request, require, reset, review, run, safe,security, set, similar, simple, simplify, special, structure,switch, test, text, think, trunk, try, turn, useful, user,valid, version, wait

Perfective: cleanup, consistent, declaration, definition,header, include, inline, move, prototype, removal, static,style, unused, variable, warning, whitespace

Q: How'd that go?
A: Not so good



$$G = \frac{2*Recall*(1 - PF)}{(Recall + (1 - PF)}$$

RED: standard Empirical SE

Q: What is the dependent variable?
A1: Use a Keyword List

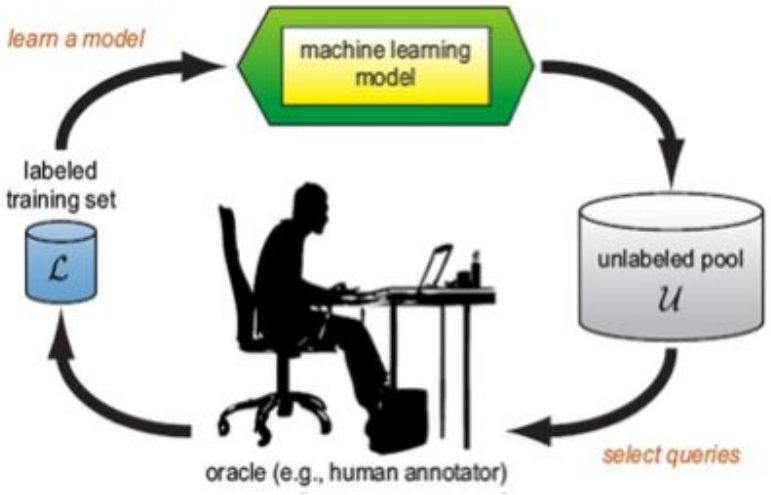
Look at project releases R1,R2,R3,....

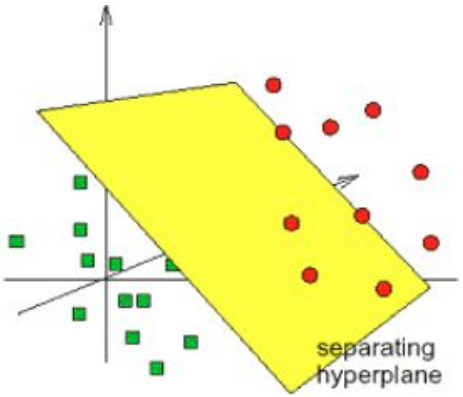
- In release[i]
  - Label the commits, commenting on errors
  - What code was changed before that? (SZZ algorithm)
    - Mark that code as “defective”
  - Build a predictor for that target
- Apply that predictor to release[i+1]

Q: What is the dependent variable?
A1: Human+AI Interaction to Learn Commit Classifier

Look at project releases R1,R2,R3,....

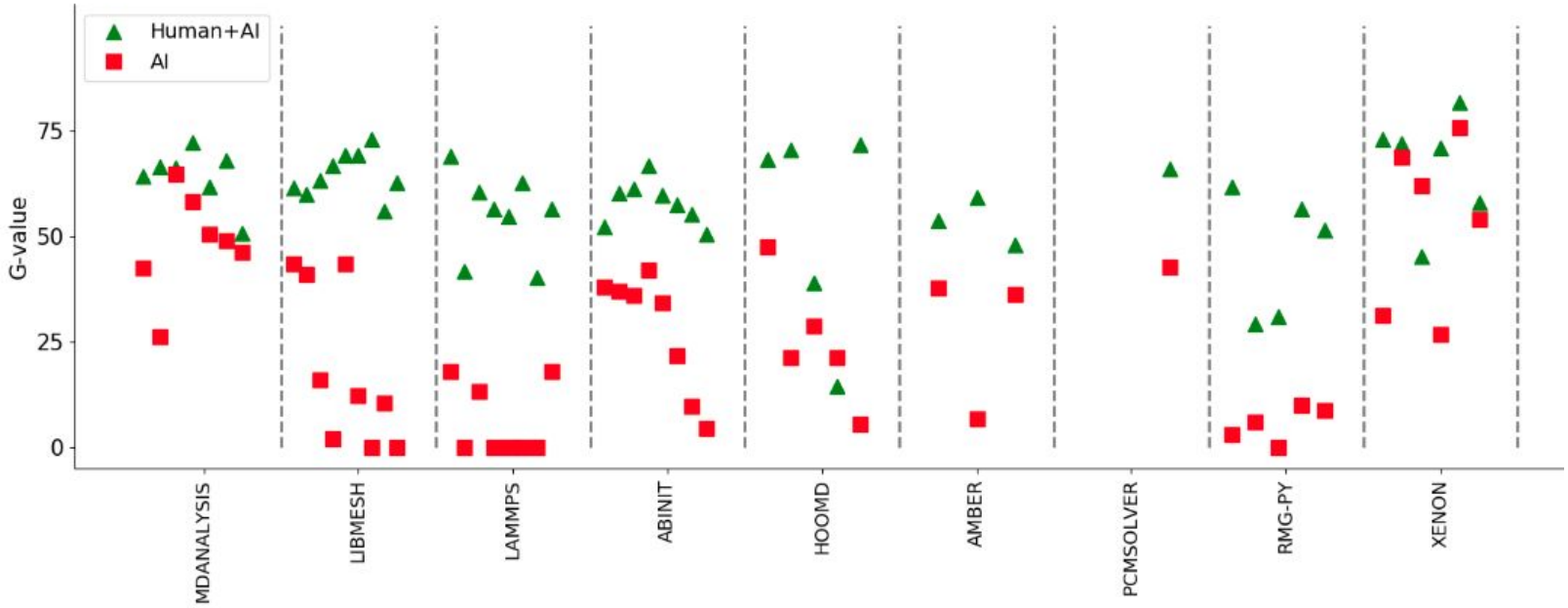
- In release[i]
  - **Label the commits, commenting on errors**
  - What code was changed before that? (SZZ algorithm)
    - Mark that code as "defective"
  - Build a predictor for that target
- Apply that predictor to release[i+1]





- So not a trite list of words, learned from other domains
- But a commit classifier specialized for Compute.Science

Q: How'd that go?
A2: Much better



$$G = \frac{2*Recall*(1 - PF)}{Recall + (1 - PF)}$$

GREEN : standard Empirical SE

- So not a trite list of words, learned from other domains
- But a commit classifier specialized for Compute.Science

Conclusions

Sound bites

- Comput.Science projects is **orders of magnitude better** than convention code:
  - Higher ratio of "good code" (370 times)
  - Last much longer (e.g. decades, not years)

- We can **learn quality agents** for Comput.Science code:
  - Out-of-the-box, Standard Empirical SE is not so good
  - But it can be successfully adapted (see above)

- Open area of research
  - **Effective Empirical SE for Comput.Science**

Acknowledgement

Individuals Wolfgang Bangerth and Rajiv Ramnath
MOLSSI Paul Saxe, Jessica Nash, and Eliseo Marin-Rimoldi
SCGI Nancy Wilkins-Diehr, Maytal Dahan, and Michael Zentner
CERN Peter Elmer