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

- SI^2 meet
- email survey (400)
- MOLSSI (teleconfs)
- SGCI (teleconfs)
- Literature review
- issues with labelling

Graph showing the number of projects from May 2018 to February 2019.
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:
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

For Students
I seek talented grad students for AI+SE. Is that you?
- Why NC State?
- My projects
- My lab
- How to apply?
- All my students

For Industry
Ask me how to innovate. On time. On budget. Case studies:
- Microsoft
- Grammatetch
- LexisNexis:1,2,3
- NASA
- IBM:1,2,3,4
- CSIRO

My Funding
$10M (total). From many sources, e.g.:
- IBM
- NASA
- CSIRO

"Less, but better"
I find simple solutions to seemingly hard problems (see examples).
So what can I simplify for you?

contact me
Office: 3298, cell.map
Cell: 919-376-2859
Fax: 919-515-7846
Mail: CS, 890 Oval Dr, Raleigh, NC, USA, 27695-8206.

Feb 7: New video online: my CodeFest19 talk "After data mining, what is next? More...
Feb 3: New pre-print: Optimizers and data miners are better than ether, alone More...
Feb 2: New pre-print: How to DODGE complex software analytics (often many of my own past papers) More...
Feb 1: Invited to IEEE Fellow 2020 review board
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

So does software

<table>
<thead>
<tr>
<th>Check</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal purpose (# Developers)</td>
<td>&gt; 7</td>
</tr>
<tr>
<td>Collaboration (Pull requests)</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Issues</td>
<td>&gt; 10</td>
</tr>
<tr>
<td>Releases</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>Commits</td>
<td>&gt; 20</td>
</tr>
<tr>
<td>Duration</td>
<td>&gt; 1 year</td>
</tr>
</tbody>
</table>

10,000 “good” projects

57,000,000/10,000 = 5700
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.

Hamill, Goseva-Popstojanova, Common Trends in Software Fault and Failure Data IEEE TSE, 35(4) 2009
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

Nov 23, 1997 – Feb 12, 2019

Contributions to master, excluding merge commits
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

Table 1: Computational Scientific Projects Summary.

<table>
<thead>
<tr>
<th>Project</th>
<th>Language</th>
<th># Developers</th>
<th>Duration (Years)</th>
<th># Commits</th>
<th># Stars</th>
<th># Issues</th>
<th># Releases</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABINIT</td>
<td>Fortran</td>
<td>23</td>
<td>2.3</td>
<td>6793</td>
<td>53</td>
<td>13</td>
<td>96</td>
</tr>
<tr>
<td>LIBMESH</td>
<td>C</td>
<td>55</td>
<td>6</td>
<td>17133</td>
<td>247</td>
<td>449</td>
<td>59</td>
</tr>
<tr>
<td>LAMMPS</td>
<td>Python</td>
<td>74</td>
<td>5.13</td>
<td>3814133</td>
<td>235</td>
<td>1006</td>
<td>48</td>
</tr>
<tr>
<td>MDAnalysis</td>
<td>Python</td>
<td>70</td>
<td>3.5</td>
<td>1313</td>
<td>335</td>
<td>1094</td>
<td>48</td>
</tr>
<tr>
<td>trellis</td>
<td>Java</td>
<td>3</td>
<td>2</td>
<td>38925</td>
<td>13</td>
<td>425</td>
<td>56</td>
</tr>
<tr>
<td>yank</td>
<td>Python</td>
<td>8</td>
<td>5</td>
<td>27260</td>
<td>557</td>
<td>34</td>
<td>1</td>
</tr>
</tbody>
</table>

Check | Condition
---|---
Personal purpose (# Developers) | > 7
Collaboration (Pull requests) | > 0
Issues | > 10
Releases | > 1
Commits | > 20
Duration | > 1 year
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)
<table>
<thead>
<tr>
<th>Statement</th>
<th>Positive View</th>
</tr>
</thead>
</table>
| “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
- **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
- **HOOMD**: particle simulation toolkit. Hard particle Monte Carlo simulations of a variety of shape classes (and molecular dynamics simulations of particles)
- **AMBER**: Fast, parallelized molecular dynamics trajectory data analysis
- **LAMMPS**: Large-scale Atomic/Molecular Massively Parallel Simulator (maintained and developed by the Sandia National Laboratory).
9 Projects: Independent variables

Language agnostic

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

+ data miner
+ defect prection
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]

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, appropriate, available, bad, behavior, block, build, call, case, catch, change, character, compatibility, compile, config, configuration, context, correctly, create, currently, default, different, documentation, dump, easier, except, exist, explicitly, fail, fast, feature, format, future, good, hack, hard, header, help, include, information, init, inline, install, instead, internal, inroduce, issue, lock, log, logic, look, merge, method, necessary, new, old, operation, operations, pass, patch, previous, previously, 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

\[ G = \frac{2 \times \text{Recall} \times (1 - \text{PF})}{(\text{Recall} + (1 - \text{PF}))} \]

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

\[
G = \frac{2 \times \text{Recall} \times (1 - \text{PF})}{\text{Recall} + (1 - \text{PF})}
\]

**GREEN**: standard Empirical SE

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