Understanding Your Software Development
(through git repository data mining and code analysis)

Boyana R. Norris (she/her)
University of Oregon

Software Productivity and Sustainability track @ Argonne Training Program on Extreme-Scale Computing summer school

Contributors: Stephen Fickas (UO), Bosco Ndemeye (UO), Boyana R. Norris (UO), Jason Prideaux (UO)

https://github.com/CAT-SDK/GremCat

See slide 2 for license details
License, Citation and Acknowledgements

License and Citation

• This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

• The requested citation the overall tutorial is: David E. Bernholdt, Anshu Dubey, Todd Gamblin, Jared O’Neal, and Boyana R. Norris, Software Productivity and Sustainability track, in Argonne Training Program on Extreme-Scale Computing, St. Charles, Illinois, 2022. DOI: 10.6084/m9.figshare.20416215.

• Individual modules may be cited as Speaker, Module Title, in Better Scientific Software tutorial, ISC, 2022 ...

Acknowledgements

• This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

• This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.

• This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.

• This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001

• This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

• This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.

• This work was performed in part at University of Oregon through a subcontract with Argonne National Laboratory.
Motivation

- Software development productivity is affected by many factors, many of which are difficult or impossible to measure (accurately).

- So far in this track, the focus was on concepts and best practices based on research, experience and expert opinion.

- Here, we focus on supporting best practices through software tools that can help understand how existing or new software practices may affect these observables.

- Tools are meant to be used -- so we will show some simple ways in which anyone can begin exploring their own project’s data without significant up-front investment of time and effort.
Motivation

(Am I just making things up?)


Top three (on average):
• Job enthusiasm (F1)
• Peer support for new ideas (F2)
• Useful feedback about job performance (F11)

Top factor at Google!
Other software-related factors

A few other positively rated software-related factors (in decreasing order of average scores):

+ My project’s bug finding process is efficient and effective.
+ The software process my project uses is well defined.
+ My software reuses code, such as by using APIs, rather than duplicating it.
+ My software’s architecture mitigates risks (e.g., security vulnerability, changes in requirements, etc.).

Negative impact on productivity:

− My software requires extensive data storage.
− My software is extremely complex.
− My software's platform (e.g. development environment, software stack, hardware stack) changes rapidly.
− Extensive documentation is required to use my software at different points in its lifecycle.
“A notable outcome of the ranking is that the top 10 productivity factors are non-technical. This is somewhat surprising, given that most software engineering research tends to focus on technical aspects of software engineering, in our estimation.

Thus, a vigorous refocusing on human factors may yield substantially more industry impact for the software engineering research community. For instance, answering the following questions may be especially fruitful:

- What makes software developers enthusiastic about their job? What accounts for differences in levels of enthusiasm between developers? What interventions can increase enthusiasm? This work can extend existing work on developer happiness [24] and motivation [25].
- What kinds of new ideas are commonly expressed in software development practice? What actions influence developers’ feelings of support for those ideas? What interventions can increase support for new ideas, while maintaining current commitments?
- What kinds of job feedback do software engineers receive, and what makes it useful? What kinds of feedback is not useful? What interventions can increase the regularity and usefulness of feedback?” [emphases mine]
What we are trying to do

Motivation and enthusiasm are ephemeral things that cannot be quantified easily.

On the other hand, we have many artifacts and associated metadata that are related (if indirectly) to productivity.

So, we are betting that using some data is better than not looking at any data, as we make decisions that are aimed at improving software development productivity and code quality.

This work is part of two DOE ECP projects:

- **IDEAS-ECP**: foster and promote SE practices for better software
- **xSDK**: Extreme-scale Scientific Software Development Kit

More and better software
What this module is about

➢ We introduce a flexible, efficient, and usable software framework for acquiring, storing, manipulating, and visualizing development-related data.

➢ We demonstrate a few of its capabilities here; we continue to add new patterns and tools.
  ○ Contributions welcome! github.com/CAT-SDK/GremCat/

ECP projects that may be present in examples in this module: Spack, LAMMPS, PETSc, Nek5000 E3SM, QMCPACK, QDPXX, LATTE NAMD, fast-export, Enzo, TAU2, xpress-apex, LATTE, NWChem, FLASH, Gingko.
Part I: Mining development metadata

Part II: Analyzing code
Part I: Mining your development data

Development data:
- Git metadata: commits, forks, branches, developers
- Issues and associated discussions
- Pull requests (github, gitlab) and associated discussions
- Mailing list archives

Goal: Analyze available data to help formulate and answer questions about development processes and their impact on productivity and code quality.
Questions that can be answered (in part) with IDEAS data analysis tools

➢ If I adopt practice X, how will metric Y be affected?
➢ How active is the developer community? (git, issues, PRs, emails)
➢ What parts of the code base could benefit from review or refactoring? (git, issues)
➢ What is the project’s reliance on individual developers? (git)
➢ How are developers’ contributions split among different categories? (git, manual labeling required)
➢ How engaged are the user and developer communities? (PRs, issues, mailing lists)
➢ What are some hot topics of discussion? (issues, mailing lists)
➢ How and on what do developers collaborate? (git, issues, mailing lists)
➢ ....
Common patterns with known implications

Pluralsight book (2019)¹:

“20 patterns is a collection of work patterns we’ve observed in working with hundreds of software teams. Our hope is that you’ll use this field guide to get a better feel for how the team works, and to recognize achievement, spot bottlenecks, and debug your development process with data.”

Common patterns with known implications

Starting with these pattern descriptions, we can:

➢ Identify patterns that are relevant to HPC (open-source) software development.
➢ Characterize each pattern using data from revision control systems and developer communications.
➢ Inform decisions of the effects of adopting new SE practices.

### Example metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monthly bug fix rate</td>
<td>Computes cumulative count of bugs closed each month, starting from the beginning of the project. A higher value is not necessarily good, trends are more informative than individual values.</td>
</tr>
<tr>
<td>Monthly feature request rate</td>
<td>Number of new feature requests made by users. A higher value may indicate popularity/importance of the feature or missing functionality in the current software version.</td>
</tr>
<tr>
<td>Correlation of the number of issues with project age</td>
<td>Gives a good insight into the life cycle of the project by mapping the trend of issues raised over the lifetime of the project.</td>
</tr>
<tr>
<td>Commits, derived metrics</td>
<td>Characterizes some aspects of developer activity.</td>
</tr>
<tr>
<td>Number of issues</td>
<td>Project-related communications can be used to indicate community involvement and rates at which issues are resolved. For example, fast-growing projects can have significantly more activity in issues than more mature ones.</td>
</tr>
<tr>
<td>Issue categories</td>
<td>Identifies the types of issues that are reported in the repository. This information will be useful to derive correlation with other metrics.</td>
</tr>
<tr>
<td>Number of followers and watchers</td>
<td>Reports the code maturity and popularity. In addition, the number of followers can be correlated with the time it takes to fix reported issues.</td>
</tr>
<tr>
<td>Mailing list metrics</td>
<td>Relates to average time in discussion, most popular topic of interest, product activity based on type of emails, etc.</td>
</tr>
<tr>
<td>Number of contributors</td>
<td>Size of development community. When analyzed over time, we can estimate project turnover and identify different types of contributors.</td>
</tr>
<tr>
<td>Code complexity</td>
<td>Compute changes to code complexity metrics, e.g., cumulative size or cyclomatic complexity, over time.</td>
</tr>
</tbody>
</table>
Example pattern: Domain champion

The Domain Champion is an expert in a particular area of the codebase. They know nearly everything there is to know about their domain: every class, every method, every algorithm and pattern.

In truth, they probably wrote most of it, and in some cases rewrite the same sections of code multiple times.

The Domain Champion isn’t just “the engineer who knows credit card processing”; it’s all they ever work on. It’s their whole day, every day.

Some degree of job specialization is essential and often motivating. But even within specialized roles there can be ‘too much of one thing.’ Managers must balance enabling a team member to unilaterally own the expertise, and encouraging breadth of experience.

How to recognize it
Domain Champions will always work in the same area of code. They’ll also rewrite their code over and over, and you’ll see it in churn and legacy refactoring metrics as they perfect it.

Domain Champions are deeply familiar with one particular domain. As a result, they typically submit their work in small, frequent commits and will show a sustained above average impact.

Because no one else knows more than the Domain Champion, there’s usually very little actionable feedback that others can provide in the review process. As a result, Domain Champions will typically show low Receptiveness in incorporating feedback from reviews.

Domain Champions will seldom, if ever, appear blocked. Short-term, it’s a highly productive pattern. But it’s often not sustainable and can lead to stagnation, which of course can lead to attrition.

Why do we care?

➢ Can lead to great productivity
➢ Code quality implications
➢ Turnover effects on code -- what happens to the domain when the domain champion leaves?

“The Domain Champion is an expert in a particular area of the codebase. They know nearly everything there is to know about their domain: every class, every method, every algorithm and pattern.”
Domain champion pattern: How do we detect it?

“The Domain Champion is an expert in a particular area of the codebase.”

Can also consider other notions of “domain”, e.g., class, package, module, etc.

Can be computed over any specific period.

Can consider different “change” metrics, more on that later.

Individual developers or subteams
Domain Champion Pattern: What, if anything, should we do?

“The Domain Champion is an expert in a particular area of the codebase.”

+ Highly productive pattern in the short term.
- There’s usually very little actionable feedback that others can provide in code reviews.
- Potentially not sustainable → can lead to stagnation.

😊 Possible actions:
- Assign the DC tickets for other areas of the code.
- Make an effort to involve others (e.g., new developers) in work in that domain.
Another example pattern: Unusually high churn

Churn is a natural and healthy part of the development process and varies from project to project. However, Unusually High Churn is often an early indicator that a team or a person may be struggling with an assignment.

In benchmarking the code contribution patterns of over 85,000 software engineers, Pluralsight’s data science team identified that Code Churn levels frequently run between 13-30% of all code committed (i.e., 70-87% Efficiency), while a typical team can expect to operate in the neighborhood of 25% Code Churn (75% Efficiency).

Testing, reworking, and exploring various solutions is expected, and these levels will vary between people, types of projects, and stage in the software lifecycle. Given the variance, becoming familiar with your team’s ‘normal’ levels is necessary to identify when something is off.

Unusually high churn levels aren’t a problem in themselves. More likely, there are outside factors causing the problem.

An unusually high level of churn can be indicative of one of three behaviors:

- **Perfectionism**: When an engineers’ standards of “good enough” are not aligned with the company’s standard of “good enough.” Engineers keep going back into the code to rewrite it because they think it can and should be better but may not add much to the actual functionality of the code.

- **They’re struggling with the problem at hand**: This situation manifests differently than with Hoarding the Code (pattern #2), because in this case, the engineer initially thought they had correctly solved the problem, perhaps even sent it off for review, and then discovered it needed to be rewritten. Not just touched up. Rewritten.

- **Or most commonly, issues concerning external stakeholders.** We see this with unclear or ambiguous specs, late arriving requirements, or mid-sprint updates to the deliverables.

**How to recognize it**

This pattern is characterized by high levels of churn in the back of the sprint or project. Watch for churn rates that climb significantly above the engineer’s historical average (see the Snapshot and Spot Check reports), pairing that information with where they are in a project.

- **Could be a sign of normal productive development**
- **May indicate need for more developer resources**
- **High conflict potential**
Unusually high churn pattern: How do we detect it?

- Decide on the **granularity**. Some possibilities: modules, classes, files, functions.
- Define the **actors** -- groups of people (e.g., sub-teams), individual developers; doesn’t have to be people, it can also be milestones or other project entities.
- Choose the **time period**.
- Choose the **churn** metric. Some examples: lines of code, cos (and other) difference between code versions, number of PRs, commits, number of files.
Unusually high churn pattern: What to do?

Unusually large amount of changes to single files or components may lead to development inefficiencies.

😊 Possible actions:

➤ Consider refactoring the high-churn project components
➤ Consider involving more developers in high-churn areas that are dominated by a single person
Pattern: In the zone

When are top contributors most productive?

+ Consistent high productivity during certain times of day
- Burnout, work-life balance

How to compute?

1. Choose productivity metric (LOCC, text difference, # commits, # PRs, etc.)
2. Choose time period
3. Choose visualization
In the zone pattern: What to do?

😊 Possible actions:

➢ Acknowledge the consistently high-performing developers.

➢ Recognize and acknowledge positive change in non-top developers (e.g., junior or new contributors).

➢ Consider the timing of and number of meetings during developers’ most productive times.

When are top contributors most productive?

+ Consistent high productivity during certain times of day
- Burnout, work-life balance
Technical Details and Examples

Implementation

Python packages
- database
- patterns
- visualizer

MySQL Database

Local project cache

Git commits.
Issues, emails

Jupyter notebooks

Example analysis workflow

```python
from patterns.visualizer import Visualizer
vis = Visualizer(project_name='spack')
vis.get_data()

INFO: Loaded local cached copy of spack data.
INFO: Done computing averages.
64909 commits (code only)
```

```python
df = vis.plot_overall_project_locc(time_range=None, log=True)
```

Some currently available projects (more are being added constantly): LAMMPS, Spack, PETSc, Nek5000, NWChem, E3SM, QPMCPACK, qdpxx, NWChem, TAU,... Initial import can take up to 36 hours for some larger projects, but subsequent updates are fast and automated.

https://github.com/HPCL/ideas-uo/

Impact of different “change” estimates

Simple line counts in git commit diffs:
- Patterns such as ‘---+++’ are counted as *edited* lines, e.g., 3 in this example
- Unmatched ‘-’ and ‘+’ lines counted as lines *deleted* and *added*, respectively
- \( \text{LOCC} = \text{edited} + \text{deleted} + \text{added} \)

More “intelligent” estimate of the magnitude of changes:
- Collect the *old* and *new* strings corresponding to each commit’s diffs
- Apply text distance metrics (based on the textdistance Python package; ~30 methods)
- E.g., change-size-cos is the cos distance between the vector embeddings of *old* and *new*
Impact of different “change” estimates (cont.)

➢ Time-series git data is messy
  • Moving averages help see trends for any time period

★ Combining different “change size” metrics
  ★ Example on right: simple LOCC totals and cos distance
Significant events

Project 1

Project 2

IDEAS - Productivity

ECP - Exascale Computing Project
How do projects weather interesting times?

Project 1

Project 2

Project 3

Project 4
Where is development effort going?

Example Tool: MeerCAT
Using git data to improve developer processes and efficiency
A Draft PR is created to merge feature branch into main branch.

The MeerCat PRA triggers and analyzes file changes and reports back.

More traditional Code Quality and testing tools trigger and report results.
The MeerCat PRA provides the following analysis:

1. Are required docstrings in place and actually match the code?
2. If changes to files will cause existing test cases to become misaligned.
3. Potentially useful labels to add to the PR for other tools to use.
4. Suggests other people to add to the discussion given their prior roles related to files in this PR.

The link will take the user to the MeerCat PRA site for further refinement of the PR (see next slide).
Once the user is at the PRA site, several aids are available.

After analysis, the PRA finds that one commit has removed a parameter from a function `sub`, but there is been no change to the documentation, i.e., code and documentation are misaligned.

Here the PRA editor is highlighting the Numpy docstring code that needs to be changed and why.

The user can download changes made to a Git Patch file and then easily merge in with existing PR.
The PRA can also search the repo for test files that reference the now changed `sub` function.

It discovers one such file (now misaligned) and suggests changes in the editor. Once the user removes the now invalid cases, the PRA will add the test file (and commits) to the existing Pull Request through the Patch mechanism.

The user can download changes made to a Git Patch file and then easily merge in with existing PR.
MeerCAT: File Explorer

1. MeerCat does analysis and leaves comment in GitHub PR
2. If user clicks link, she is taken to the PRA tool (previous slides).
3. From the PRA, click on a specific file to get to the file explorer.

Benefits:
+ Problems detected early, don’t have to wait for CI failure
+ Better overall documentation, testing
Part II: Analyzing Code
Defects that code analyses can catch

- **Security**: Buffer overruns, improperly validated input.
- **Memory safety**: Null dereference, uninitialized data.
- **Resource leaks**: Memory, OS resources.
- **API Protocols**: Improper use of APIs, incomplete/incorrect implementations.
- **Exceptions**: Arithmetic/library/user-defined.
- **Encapsulation**: Accessing internal data, calling private functions.
- **Data races**: Two threads access the same data without synchronization.

Key idea: check compliance with (mostly) simple, mechanical design rules.

Standard: ISO/IEC 5055:2021(E): Information technology — Software measurement — Software quality measurement — Automated source code quality measures
General-purpose tools for code checking (bugs, style)

➢ C/C++
  - Run a bunch of general analyses with **scan-check** (wrapper around `clang --analyze`, which is uses the static analyzer below)
    - Minimally invasive, not very customizable
    - Works great with CMake and Autoconf builds
  - Clang **static analyzer** component: extensible analysis framework for bug finding
    - Can do more complex analyses (path-sensitive, inter-procedural analysis based on a symbolic execution technique)
    - Requires more compiler knowledge to extend
  - **Clang-tidy**: extensible (libTooling-based) framework for diagnosing typical programming errors or style issues
    - Checking and enforcing of simple coding conventions
    - Modular, provides API for implementing new checks
    - Relatively easy to integrate into Cmake

➢ Fortran
  - Flang (compiler front-end to LLVM)
  - Fortran-linter (limited)
Example development workflow that considers **code quality**

Example “make commit” workflow (easy in C/C++, and hopefully possible soon for Fortran):

- `clang-format` passes and reformats the code
- `clang-tidy` passes and enforces coding conventions
- `clang static analyzer` compiles debug and production builds (check errors)
- **Project-specific analysis** for debug and production build (check errors)
- debug/production builds get compiled and unit tests launched (check errors)
- production build + unit tests run under valgrind (check errors)
- production build gets compiled and unit test launched (check errors)
- production build with `--coverage` gets compiled and unit test launched against llvm-cov (write unit-test coverage stats)
Our goals and approach

Make it easy(-ish) to define and apply static and dynamic program analysis techniques to identify quality-related problems in HPC codes.

How? Two parts:
A. By integrating general static and dynamic program analyses into the HPC software development process: mainly through documentation and examples.
B. By creating easy interfaces to custom analyses, with examples.

Why?
➢ Abstraction
   ● Elide details of a specific implementation.
   ● Capture semantically relevant details; ignore the rest.
➢ Programs as data
   ● Programs are just trees/graphs!
   ● …and we have lots of ways to analyze trees/graphs
Static program analysis is...

Ensure everything is checked the same way.

**Systematic** examination of an abstraction of program state space.

Only track “important” things...

Applies to all possible executions.

Examples:
- clang-tidy
- Clang static analyzer
Dynamic program analysis is...

Partial examination of an abstraction of a single execution path at runtime.

Examples:
- Valgrind
- Clang/LLVM sanitizers (better!)

Instrumented code only.

Can capture information not available statically.

Applies to specific executions; can miss errors.
Example: Using scan-build with HYPRE

```bash
hypre/src/cmbuild$ scan-build cmake ..
hypre/src/cmbuild$ scan-build make
```

![Build log output showing successful compilation]

---

HYPRE: Scalable Linear Solvers and Multigrid Methods. [https://github.com/hypre-space/hypre](https://github.com/hypre-space/hypre)
Example: hypre (cont.)

```
hypre/src/cmbuild$ scan-view /tmp/scan-build-2021-05-27-073157-25436-1
```

Details:

```
#undef _FUNC_
define _FUNC_ "Mat_dhTranspose"

void Mat_dhTranspose(Mat_dh A, Mat_dh *Bout) {
    START_FUNC_DH
    Mat_dh B;

    if (sp_dh > 1) { SET_V_ERROR("only for sequential"); }

    Hat_dhCreate(&B); CHECK_V_ERROR;

    Calling 'Mat_dhCreate' →

    Returning from 'Mat_dhCreate' →

    Assuming 'enFlag_dh' is < 1 →

    Taking false branch →

    Assuming 'enFlag_dh' is false →

    Taking false branch →

    Assigned value is garbage or undefined

    *Bout = B;

    B->m = B->m = A->m;
    mat_dhTranspose_private(A->m, A->rp, &B->rp, A->cval, &B->cval, A->aval, &B->aval); CHECK_V_ERROR;

    END_FUNC_DH
```
What about project-specific requirements?

Do you need to be a compiler expert to implement new program checks?

Thankfully -- no!
Implementation approach: Part B

Develop custom static and dynamic checking based on \textit{project-specific} requirements.

- **C/C++**
  - Static: use and extend existing APIs (Clang static analyzer, clang-tidy); implement custom AST traversals and matchers (more details next)
  - Dynamic: use Clang sanitizer APIs; python for simplicity and easy of extensions by HPC software developers

- **Fortran**
  - Static: need to write new Flang-based checkers \textit{only} for things that Fortran developers actually care about
  - Dynamic: do we need anything?
Example: Checking for violations of PETSc developer rules

From PETSc Style and Usage Guide: https://petsc.org/release/developers/style/
## Examples of project-specific rule violations (PETSc 3.14)

<table>
<thead>
<tr>
<th>PETSc Rule</th>
<th>PETSc Construct</th>
<th>Description</th>
<th>path</th>
<th>Line</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule-2</td>
<td>Macro in the library</td>
<td>#define mpi_reduce_scatter PETSC_MPI_REDUCE_SCATTER</td>
<td>~/petsc-3.14.3/include/petsc/mpiuni/mpiunifdef.h</td>
<td>118</td>
<td>2</td>
</tr>
<tr>
<td>Rule-3</td>
<td>Function declaration in the library</td>
<td>PETSC_INTERN PetscErrorCode MatFactorFactorizeSchurComplement_Private(Mat);</td>
<td>~/petsc-3.14.3/include/petsc/private/matimpl.h</td>
<td>494</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Function declaration in the library</td>
<td>PETSC_EXTERN PetscErrorCode MatFactorFactorizeSchurComplement(Mat);</td>
<td>~/petsc-3.14.3/include/petscmat.h</td>
<td>1245</td>
<td>29</td>
</tr>
<tr>
<td>Rule-4</td>
<td>Function definition in the library</td>
<td>PETSC_EXTERN PetscErrorCode DMDAVecGetArray(DM, Vec, void *)</td>
<td>~/petsc-3.14.3/include/petscdmda.h</td>
<td>113</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Function call in the application</td>
<td>ierr = VecGetArray(y,yv)</td>
<td>~/petsc-3.14.3/include/petscvec.h</td>
<td>545</td>
<td>10</td>
</tr>
<tr>
<td>Rule-5</td>
<td>Function call in the library</td>
<td>ierr = PetscFEPushforwardGradient(fe, fegeom, 1, interpolantGrad);</td>
<td>~/petsc-3.14.3/include/petsc/private/petscfeimpl.h</td>
<td>332</td>
<td>10</td>
</tr>
<tr>
<td>Rule-6</td>
<td>If in the library</td>
<td>if (p == 0) return node;</td>
<td>~/petsc-3.14.3/src/dm/impls/plex/gmshlex.h</td>
<td>231</td>
<td>3</td>
</tr>
<tr>
<td>Rule-7</td>
<td>Macro in the library</td>
<td>ifndef PETSC4PY_COMPAT_MUMPS_H</td>
<td>~/petsc-3.14.3/src/binding/petsc4py/src/include/compat/mumps.h</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Rules from: [https://petsc.org/release/developers/style/](https://petsc.org/release/developers/style/)
Example results for a subset of the PETSc rules
## Capabilities summary

<table>
<thead>
<tr>
<th>Type of Data/Analysis</th>
<th>Database</th>
<th>Examples</th>
<th>Repository Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Git data: commits, changes (lines, files, etc.)</td>
<td>✔️</td>
<td>✔️</td>
<td><a href="github.com/CAT-SDK/GremCat/">github.com/CAT-SDK/GremCat/</a></td>
</tr>
<tr>
<td>Github and Gilab issues and associated metadata</td>
<td>✔️</td>
<td>✔️</td>
<td><a href="github.com/CAT-SDK/GremCat/">github.com/CAT-SDK/GremCat/</a></td>
</tr>
</tbody>
</table>
| Code quality checkers (dynamic & static)                   | ✗        | ✔️       | • [github.com/HPCL/code-analysis](github.com/HPCL/code-analysis) (dynamic)  
| Mailing lists                                              | ✔️       | ✔️       | Not publicly available yet, contact [norris@cs.uoregon.edu](mailto:norris@cs.uoregon.edu) |
Summary

➢ We introduced a flexible, efficient, and usable software framework for acquiring, storing, manipulating, and visualizing development-related data.

➢ We demonstrated a few of its capabilities here; a growing number of analyses and tools are continuously being developed.

● Contributions and/or requests welcome! https://github.com/CAT-SDK/GremCat

➢ Acknowledgments: DOE ECP IDEAS Productivity Project

● Carter Perkins, Bosco Ndemeye, Stephen Fickas, University of Oregon
● Armando Acosta and Kanika Sood, California State University, Fullerton
● Anshu Dubey and Lois Curfman McInnes, Argonne National Laboratory

Thank you!

ECP projects that may be present in examples in this presentation: Spack, LAMMPS, PETSc, Nek5000 E3SM, QMCPACK, QDPXX, LATTE NAMD, HYPRE, fast-export, Enzo, TAU2, xpress-apex, LATTE, NWChem