MSR/SBSE Tools and Infrastructures

By Diomidis Spinellis (rapporteur), Tse-Hsun (Peter) Chen, Yasutaka Kamei, Masanari Kondo, Neil Walkinshaw, Xin Xia, and Shin Yoo

Outline

- Tools
- Infrastructures and Data
- Practices
- Four key readings
- More readings
- Future


CHROME VANADIUM
13
15
CHROME VANADIUM
15
CHROME VANADIUM
17
CHROME VANADIUM
17
CHROME VANADIUM
8
8

Generic Tools

- Git (CLI)
- Shell scripts
  - Flexible
  - Collect and structure ready for analysis
  - Easy to interface from Git and to R/Python
  - Easy to construct pipeline incrementally

Text Processing Tools

- Mallet
- TwitterLDA

Interaction Data Collection

- ActivitySpace
- Mylyn
- Hackystat

Evolutionary Computation Tools

- Evolutionary Computation Framework — http://ecf.zemris.fer.hr/
- jMetal (Java, MOEAs) https://jmetal.github.io/jMetal/
- DEAP — Distributed Evolutionary Algorithms in Python — https://github.com/DEAP/deap
- ECJ — Evolutionary Computation in Java — https://cs.gmu.edu/~eclab/projects/ecj/
- Apache Commons Math — http://commons.apache.org/proper/commons-math/
- MOEA (Multi-objective evolutionary algorithms) Framework — https://moeaframework.org

Search-Based Software Testing Tools

- EvoSuite — unit testing for Java http://www.evosuite.org
- CAVM — structural testing generation for C using AVM https://bitbucket.org/teamcoinse/cavm/src
- FLUCCS — Fault Localisation, works with Defects4J, GP-based localisation https://bitbucket.org/teamcoinse/fluccs
- GunPowder — Code instrumentation for C
- PyGGI — genetic improvement at line level

Code Analysis Tools and Frameworks

- ASM (Java bytecode)
- BCEL
- SOOT (Unmaintained)
- JavaParser (Less powerful than SOOT)
- SrcML — Provides XML representation of code
- ckjm — Chidamber and Kemerer Java Metrics
- qmcalc — calculate quality metrics from C source code

Repository Analysis Tools

- Commit-Guru — http://commit.guru/
- reaper — Project selection https://github.com/RepoReapers/reaper
- RepoDriller — Java Framework https://github.com/mauricioaniche/repodriller
- Boa — a DSL for querying software repositories http://boa.cs.iastate.edu/
- GrimoireLab — Data gathering, enrichment, and visualization from diverse data sources http://grimoirelab.github.io/

Data Collections

- PROMISE – tera-PROMISE – SeaCraft (on Zenodo)
- MSR Challenge data sets
- MSR data showcase papers
- awesome-msr — https://github.com/dspinellis/awesome-msr
  - Links to all data sets that follow

Product Data

- **AndroZoo** — a growing collection of Android Applications
- **Boa** — a domain-specific language and infrastructure that eases mining software repositories
- **GHTorrent** — an effort to create a scalable, queriable, offline mirror of data offered through the Github REST API
- **GitHub on Google BigQuery —** GitHub data accessible through Google's BigQuery platform
- **RepoReapers** Data Set — A data set containing a collection of engineered software projects from GHTorrent.
- **Maven metrics —** a collection of software complexity & sizing metrics for the Maven Repository
- **Unix history —** a Git repository with 46 years of Unix history evolution

Fault and Failure Data

- **Bug Prediction Dataset —** a collection of models and metrics from Eclipse JDT Core, PDE UI, Equinox Framework, Lucene, Mylyn, and their histories
- **CoREBench** — a collection of 70 realistically Complex Regression Errors that were systematically extracted from the repositories and bug reports of four open-source software projects: Make, Grep, Findutils, and Coreutils
- **Defects4J** — a collection of 395 reproducible bugs collected with the goal of advancing software testing research
- **Findbugs-maven** — a set of FindBugs reports for the Java projects of the Maven repository
- **SIR** — Software-artifact Infrastructure Repository — Java, C, C++, and C# software together with test suites and fault data

Process Data

- **Code Reviews —** Code reviews of OpenStack, LibreOffice, AOSP, Qt, Eclipse
- **KaVE** — developer tool interaction data
- **mzdata** — Multi-extract and Multi-level Dataset of Mozilla Issue Tracking History
- **Stack Exchange —** an anonymized dump of all user-contributed content on the Stack Exchange network.
- **TravisTorrent** — TravisTorrent provides free and easy-to-use Traivs CI build analyses.

Other Data

- **Enron Spreadsheets and Emails —** all the spreadsheets and emails used in the paper 'Enron's Spreadsheets and Related Emails: A Dataset and Analysis'
- **STAMINA** — (STAte Machine INference Approaches) data are used to benchmark techniques for learning deterministic finite state machines (FSMs)

Practices

- Ensure reproducibility
- Separate the data from its processing
- Adopt easily shareable data formats
- Use Git porcelain format, terminate records with blank (-z)
- Provide SBSE algorithm parameters
- Statistical rigor
  - Use appropriate sample size
  - Include descriptive statistics
  - Provide random number seeds
- Adopt scientific computing best practices
- Use a systematic process for data set selection

OPEN ACCESS Freely available online

PLOS | BIOLOGY

Community Page

Best Practices for Scientific Computing

Greg Wilson[1]*, D. A. Aruliah[2], C. Titus Brown[3], Neil P. Chue Hong[4], Matt Davis[5], Richard T. Guy[6¤], Steven H. D. Haddock[7], Kathryn D. Huff[8], Ian M. Mitchell[9], Mark D. Plumbley[10], Ben Waugh[11], Ethan P. White[12], Paul Wilson[13]

PLOS | COMPUTATIONAL BIOLOGY

PERSPECTIVE

Good enough practices in scientific computing

Greg Wilson[1☯]*, Jennifer Bryan[2☯], Karen Cranston[3☯], Justin Kitzes[4☯], Lex Nederbragt[5☯], Tracy K. Teal[6☯]

1. Write programs for people, not computers.

(a) A program should not require its readers to hold more than a handful of facts in memory at once.
(b) Make names consistent, distinctive, and meaningful.
(c) Make code style and formatting consistent.

2. Let the computer do the work.

(a) Make the computer repeat tasks.
(b) Save recent commands in a file for re-use.
(c) Use a build tool to automate workflows.

3. Make incremental changes.

(a) Work in small steps with frequent feedback and course correction.
(b) Use a version control system.
(c) Put everything that has been created manually in version control.

4. Don't repeat yourself (or others).

(a) Every piece of data must have a single authoritative representation in the system.
(b) Modularize code rather than copying and pasting.
(c) Re-use code instead of rewriting it.

5. Plan for mistakes.

(a) Add assertions to programs to check their operation.
(b) Use an off-the-shelf unit testing library.
(c) Turn bugs into test cases.
(d) Use a symbolic debugger.

6. Optimize software only after it works correctly.

(a) Use a profiler to identify bottlenecks.
(b) Write code in the highest-level language possible.

7. Document design and purpose, not mechanics.

(a) Document interfaces and reasons, not implementations.
(b) Refactor code in preference to explaining how it works.
(c) Embed the documentation for a piece of software in that software.

8. Collaborate.

(a) Use pre-merge code reviews.
(b) Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems.
(c) Use an issue tracking tool.

On Reproducibility

- Provide script to extract data
- Fork the repository from which data was extracted
- Provide data set
- Archive data set (e.g. on Zenodo) and cite its DOI
- Provide a way to run the processing
  - Provide an interactive notebook
  - Shell script with dependency installation
  - Docker file
  - VM Image file
- Provide statistical analysis scripts
- For double-blind reviewing
  - Anonymised read-only data sharing: Zenodo, OSF.io

Five Key Readings

- Bird, Christian, Tim Menzies, and Thomas Zimmermann, eds. The Art and Science of Analyzing Software Data. Elsevier, 2015.
- Kuhn, Max, and Kjell Johnson. Applied predictive modeling. Vol. 810. New York: Springer, 2013.
- P. McMinn. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 14(2):105–156, June 2004.
- J. Petke, S. Haraldsson, M. Harman, w. langdon, D. White, and J. Woodward. Genetic improvement of software: a comprehensive survey. IEEE Transactions on Evolutionary Computation, PP(99):1–1, 2017.
- Wilson, Greg, Dhavide A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven HD Haddock et al. "Best practices for scientific computing." PLoS biology 12, no. 1 (2014): e1001745.

More Articles

- G. Fraser and A. Arcuri. Whole test suite generation. IEEE Trans. Softw. Eng., 39(2):276–291, Feb. 2013.
- Just, René, Darioush Jalali, and Michael D. Ernst. "Defects4J: A database of existing faults to enable controlled testing studies for Java programs." In Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp. 437-440. ACM, 2014.
- Rosen, Christoffer, Ben Grawi, and Emad Shihab. "Commit Guru: analytics and risk prediction of software commits." In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp. 966-969. ACM, 2015.
- Bao, Lingfeng, Deheng Ye, Zhenchang Xing, Xin Xia, and Xinyu Wang. "Activityspace: a remembrance framework to support interapplication information needs." In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pp. 864-869. IEEE, 2015.
- Gousios, Georgios, and Diomidis Spinellis. "GHTorrent: GitHub's data from a firehose." In Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, pp. 12-21. IEEE Press, 2012.
- Krishna, Rahul, Tim Menzies, and Wei Fu. "Too much automation? The bellwether effect and its implications for transfer learning." In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 122-131. ACM, 2016.

What we would do in a month

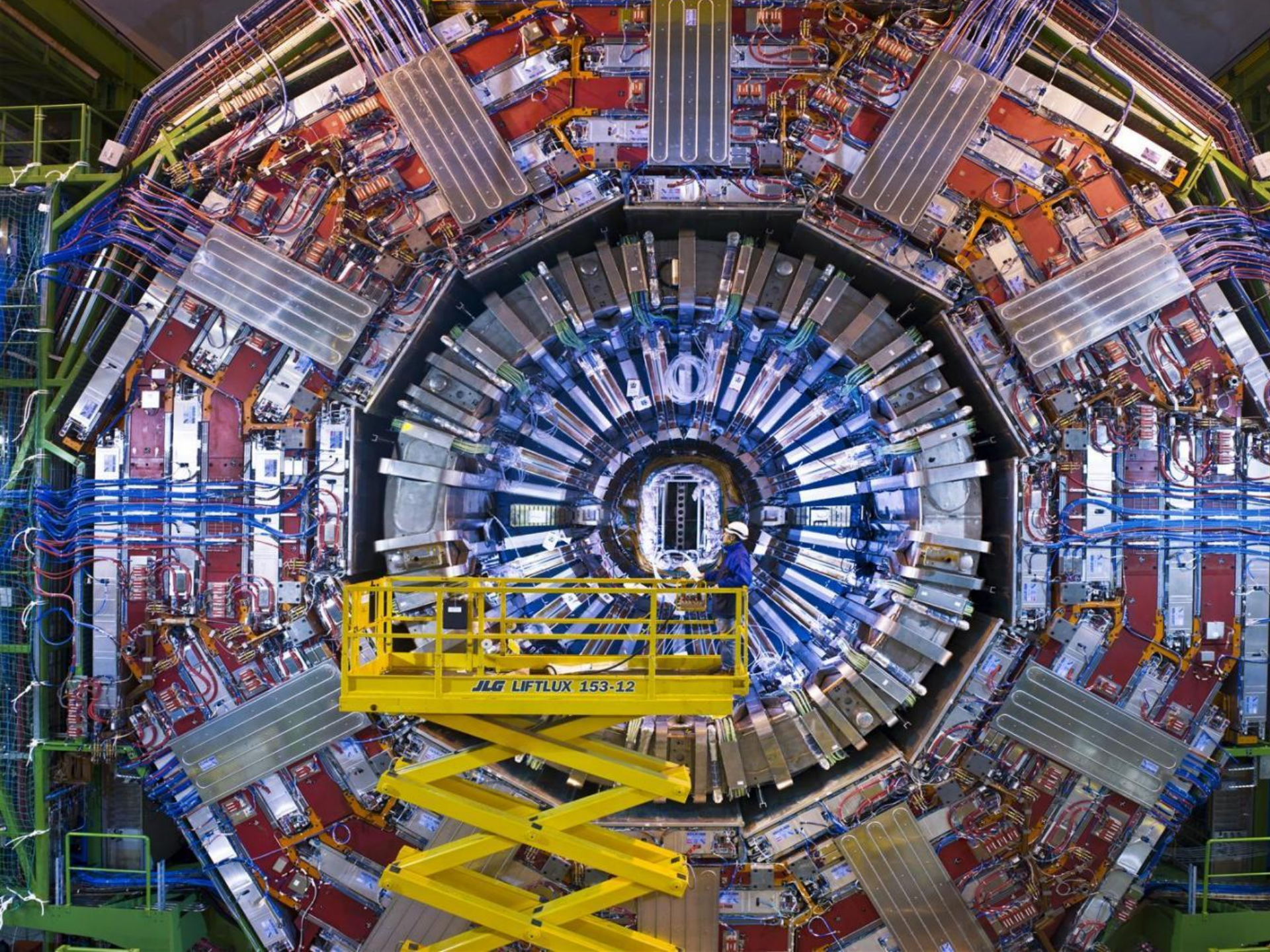
- Cookbook guide to various techniques
- Cross reference between tools/data sets and published papers

What we should do as a community

- Shared data infrastructure
  - On a cloud provider
- Product and process data from proprietary projects
- Greater emphasis to replicate experiments of significantly larger scale data sets
  - This might be difficult to publish
  - Specialist tracks for things that are difficult to publish
    - Reproducibility
    - Negative results
    - (As is done RENE-track in SANER 2018)

Product and process data from proprietary projects

- Difficult
  - Legal, regulatory, reputation, problems.
  - Can leak proprietary data
- Old projects might be candidates
  - Existing examples
    - Microsoft Word 2.0
    - Unix Research Editions
    - Microsoft DOS
    - Apple MacPaint
    - Apple iOS kernel
- Provide and obtain aggregate process data


JLG LIFTLUX 153-12

Acknowledgements

- The meeting’s financial supports from Japan's National Institute of Informatics (NII) is gratefully acknowledged.
- The research described has been partially carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 732223.