ISSN 2186-7437

NII Shonan Meeting Report

No. 2017-19

NII Shonan Meeting – Data-Driven Search-Based Software Engineering

Markus Wagner, Leandro Minku, Ahmed E. Hassan, John Clark

December 11–14, 2017

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

NII Shonan Meeting – Data-Driven Search-Based Software Engineering

Organizers:
Markus Wagner, Leandro Minku, Ahmed E. Hassan, John Clark

March 14, 2018

1 Introductory Talks

The Road Ahead for Mining Software Repositories and Software Analytics

Ahmed E. Hassan, Queen's University

Source control repositories, bug repositories, archived communications, deployment logs, and code repositories are examples of software repositories that are commonly available for most software projects. The Mining Software Repositories (MSR) field (http://msrconf.org) analyzes and cross-links the rich data available in these repositories to uncover interesting and actionable information about software systems. By transforming these static record-keeping repositories into active ones, we can guide decision processes in modern software projects. For example, data in source control repositories, traditionally used to archive code, could be linked with data in bug repositories to help practitioners propagate complex changes and to warn them about risky code based on prior changes and bugs. For additional details refer to [**?**, **?**].

A Brief Overview of Search-Based Software Engineering

Leandro L. Minku, University of Leicester

Search-Based Software Engineering (SBSE) is an approach to software engineering that re-formulates software engineering problems as search / optimisation problems, and then uses search-based optimisation algorithms to help solving such problems. For example, test suite generation can be seen as the problem of finding a set of input sequences (test cases) so as to maximise the coverage and minimise the length of the test cases. An evolutionary algorithm can then be used to solve this problem. Over the last couple of decades, SBSE has not only enjoyed a growth in the number of papers, but also an increase in the number of different stages of the software development process where SBSE approaches have been investigated. This talk gives some examples of software engineering problems that have been tackled by SBSE, including problems that have made their way into software engineering practice.

2 Keynotes / Long Talks

When Data Miners Meet Optimizers

Tim Menzies, NC State University

While MSR formulates software engineering problems as data mining problems, SBSE reformulates SE problems as optimization problems and use metaheuristic algorithms to solve them. Both MSR and SBSE share the common goal of providing insights to improve software engineering. The algorithms used in these two areas also have intrinsic relationships. We, therefore, argue that combining these two fields is useful for situations (a) which require learning from a large data source or (b) when optimizers need to know the lay of the land to find better solutions, faster.

More specifically, this talk agues that it is time to refactor our data mining algorithms and our search-based optimizer since very simple data mining algorithms can lead to spectacular improvement in optimizers. And vice versa. To show this, this talk presents numerous examples from software analytics.

- For all the slides from this talk, see tiny.cc/17search.
- For a companion paper that extends the message of this talk, see *Data-Driven Search-Based Software Engineering*, MSR'18 (arxiv.org/pdf/1801.10241).
- For a set of on-line resources that let researchers explore this area for themselves, see tiny.cc/data-se.

Search Based Software Engineering - A Tutorial

Shin Yoo, Korea Advanced Institute of Science and Technology

Search Based Software Engineering (SBSE) aims to solve various SE problems by reformulating them into optimisation problems and applying many existing metaheuristic optimisation algorithms. SBSE received much attention from the SE community, and has been applied across the entire software development lifecycle. This tutorial looks at the fundamental concepts behind the metaheuristic optimisation algorithms that are widely used in SBSE, and goes through some of the successful applications as case studies. Finally, the talk presents future research directions with an emphasis on human-computer symbiosis for SBSE.

Language Modelling from MSR + Code Mutations for SBSE

Group leads: Abram Hindle, Prem Devanbu, and Shin Yoo

Language Models

This breakout group considered how MSR and SBSE communities can collaborate around the theme of language modelling. The MSR community has been working on learning Language Models (LMs) of source code for some time: although there can be many forms of language modelling, essentially we mean the statistical/probabilistic abstraction of a large corpus of source code (i.e., repositories). One of the simplest form of LM would be n-gram models, i.e. a probability distribution that predicts the n-th token, given that $n-1$ preceding tokens have already been observed. More advanced forms may include Recurrent Neural Nets trained with the corpus, etc.

The very fact that source code exhibits *naturalness* like human languages [**?**] is one of the fundamental findings from the MSR community. A sufficiently capable language model can be used in two ways: to evaluate the naturalness of a given code (i.e., "is this code an outlier?"), and to generate (or predict) unknown code based on given context (i.e., "can we predict the next token?").

Use of Language Models in SBSE

The use of LM as generative model in SBSE has vast untapped potential, as many SBSE techniques that directly interact with source code require ways to mutate source code. Here we restrict ourselves to two of the most prominent examples.

- The entire field of Genetic Improvement [**?**] is built around the idea that it is possible to improve both functional and non-functional properties of software by applying, automatically via metaheuristics, a series of code modifications that collectively amounts to the resulting improvement. Currently, many techniques simply rely on reusing code that already exists in the given system. A generative language model that can create *previously unseen, yet realistic* code may open up entirely new directions.

- Mutation testing is a testing technique that aims to intentionally inject faults into a given system, in order to measure the adequacy of the current test suite. The majority of existing literature depends on syntactic mutations, i.e. simple and syntax-compliant replacements of tokens (e.g., from `int x = y + 1;` to `int x = y - 1;`). Mutation at the semantic level has been difficult, if only due to the lack of specifications that capture the intended semantic. A generative language model may lead to mutation testing tool that can generate something between syntactic and semantic mutants, i.e., *what the developer may have written here*.

The capability of measuring naturalness of given source code can be used as a surrogate feature in many scenarios where unnaturalness can be linked to fault proneness. It can also be connected to ease of human comprehension when comparing multiple machine generated code artefacts.

- (Un)Naturalness can be used in defect prediction, fault localisation, syntax error detection and correction [**?**] [**?**], effort prediction, bug triage, etc, as a

feature that can be correlated to fault proneness. SBSE has been applied to all of these tasks, and a strongly correlated feature can be an welcome addition to existing literature.

- Many SBSE techniques generate source code (including bug patches and test cases), and the long term maintainability of these artefacts have been questioned. Naturalness can be an additional objective in generating any of these artefacts: given two alternatives, one should always choose the machine generate artefact that are more natural with respect to human generated corpus, as estimated by a language model.

Mandatory Readings
Language Modelling: The idea of source code seen as natural language is best presented in Hindle et al. [**?**]. Tu et al. [**?**] extended this viewpoint by considering the *localness* that is perhaps unique to source code. A recent survey [**?**] provides an excellent and broad introduction.
Search Based Software Engineering: An overview of the entire field is available from Harman et al. [**?**]. Genetic Improvement, the application of metaheuristic optimisation to source code itself with the aim of improving it, is summarised in the recent survey by Petke et al. [**?**]. For mutation testing, refer to Jia and Harman [**?**].
Voluntary Readings

A good starting point for automated program repair, i.e., the *functional* Genetic Improvement, is the original GenProg [**?**, **?**, **?**]. The Plastic Surgery Hypothesis [**?**] is an attempt to distinguish what is capable of and what is not under the current approaches in automated program repair (i.e., patches obtained only by recombining existing source code). DeepFix [**?**] uses sequence to sequence neural networks to fix faults in `C` programs.
Vision

A one month collaboration between an MSR researcher with expertise on language model, and an SBSE researcher with expertise on code mutation or automated program repair, may lead to a practical prototype of code mutation operator that can be trained using a large corpus and be applied to a given program element to mutate it. For SBSE researchers, this would provide a starting point for research in many application areas, including Genetic Improvement, Automated Program Repair, and mutation testing. For MSR researchers, this would provide deeper understanding of use case scenarios that may lead to further refinement of the language models they produce.

MSR/SBSE Tools and Infrastructures

Group lead: Diomidis Spinellis

The breakout set out to put together and present important tools, infrastructures, data sets, and practices that researchers in the fields of MSR and SBSE can use in their work. The premise is to stand on the shoulders of our colleagues, rather than waste effort on duplicating groundwork.

On the generic tools front Git's command-line interface is an obvious entry-level tool. This can be usefully combined with shell scripts. Shell scripts are flexible, they can easily process Git's output and generate downstream results that can be readilly passed on to R or Python scripts for further analysis. Furthermore, such scripts allow the incremental exploratory construction of the processing pipeline.

The group then set out to outline tools that target particular types of analysis.

Text Processing Tools

- Mallet
- TwitterLDA

Interaction Data Collection Interaction data is produced by logging the interactions of users with specific applications. They can be processed to provide insight on underlying processes or optimize the corresponding interactions.

- ActivitySpace
- Mylyn
- Hackystat

Evolutionary Computation Tools

- Evolutionary Computation Framework[1]
- jMetal (Java, MOEAs)[2]
- DEAP — Distributed Evolutionary Algorithms in Python[3]
- ECJ — Evolutionary Computation in Java[4]
- Apache Commons Math[5]
- MOEA (Multi-objective evolutionary algorithms) Framework[6]

Search-Based Software Testing Tools

- EvoSuite — unit testing for Java[7]

[1] `http://ecf.zemris.fer.hr/`
[2] `https://jmetal.github.io/jMetal/`
[3] `https://github.com/DEAP/deap`
[4] `https://cs.gmu.edu/~eclab/projects/ecj/`
[5] `http://commons.apache.org/proper/commons-math/`
[6] `https://moeaframework.org`
[7] `http://www.evosuite.org`

- CAVM — structural testing generation for C using AVM[8]
- FLUCCS — Fault Localisation, works with Defects4J, GP-based localisation[9]
- 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[10]
- reaper — Project selection[11]
- RepoDriller — Java Framework[12]
- Boa — a DSL for querying software repositories[13]
- GrimoireLab — Data gathering, enrichment, and visualization from diverse data sources[14]

Data Collections

- PROMISE tera-PROMISE SeaCraft (on Zenodo)
- MSR Challenge data sets
- MSR data showcase papers
- awesome-msr[15] provides links to all data sets that follow

Product Data

[8] https://bitbucket.org/teamcoinse/cavm/src
[9] https://bitbucket.org/teamcoinse/fluccs
[10] http://commit.guru/
[11] https://github.com/RepoReapers/reaper
[12] https://github.com/mauricioaniche/repodriller
[13] http://boa.cs.iastate.edu/
[14] http://grimoirelab.github.io/
[15] https://github.com/dspinellis/awesome-msr

- 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 and 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 A number of practices can aid the efficiency and effectiveness of MSR and SBSE studies. These can be distilled in the following pieces of advice.

- Ensure reproducibility.
- Separate the data from its processing.
- Adopt easily shareable data formats.
- Use Git's porcelain format, terminate records with a null rather than a newline. (Many Unix commands offer an option to generate or process null-terminated records.)
- Provide SBSE algorithm parameters
- Strive for statistical rigor. Among other things this entails using an appropriate sample size, including descriptive statistics, and providiing the random number seeds that were used.
- Adopt scientific computing best practices.
- Use a systematic process for data set selection.

On the reproducibility front, the following can go a long way toward making the methods easily reproducible by others.

- Provide script to extract data.
- Fork the repository from which data was extracted.
- Provide the used data set.
- Archive the data set (e.g. on Zenodo) and cite its DOI.
- Provide a way to run the processing. There are a numerous ways in which this can be achieved. You can provide an interactive notebook, a shell script with automatice dependency installation, a Docker file, or a virtual machine image file.
- Provide statistical analysis scripts.
- To provide data compatible with double-blind reviewing consider using a service that supports anonymised read-only data sharing, such as Zenodo or OSF.io.

Mandatory Reading List

A book [**?**] and and a survey paper [**?**] provide a broad overview of the discussed areas. A more specialized paper reviews search-based software test data generation [**?**]. Regarding methodology, two recommended readings are Kuhn and Johnson's book on applied predictive modeling [**?**] and the advice given by Wilson and his colleagues on best practices for scientific computing [**?**].

Voluntary Reading List

- G. Fraser and A. Arcuri. Whole test suite generation. *IEEE Trans. Softw. Eng.*, 39(2):276291, Feb. 2013.
- Just, Ren, 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.

Vision

Two low-hanging fruit that came out as possible immediate actions were a cookbook-style guide to various techniques and a cross reference between tools or data sets and published papers.

On a longer term scale, there are three things we should do as a community. First, setup a shared data infrastructure, probably on a cloud provider. Second, work on making available for research purposes product and process data from proprietary projects. Third, emphasize replication experiments on significantly larger scale data sets. Although one might think that these might be difficult to publish, there are in fact tracks that cover Specialist tracks for things that are difficult to publish reproducibility and negative results, e.g. the RENE-track in SANER 2018.

Obtaining product and process data from proprietary projects will be difficult, Due to legal, regulatory, and reputation problems, but also due to the risk companies face in leaking proprietary data. Old projects might be candidates for such releases. Existing examples include Microsoft Word 2.0, Unix Research Editions, Microsoft DOS, Apple MacPaint, and Apple iOS kernel. Alternatively, companies could be encouraged to provide aggregate process data.

Combination of data mining and optimisation techniques

Group leads: Barbara Rosso Marius Lindauer

At the centre of this discussion was the term "goodness", as this needs to be defined before optimisation can take place. It must be understandable, can be multi-valued/composite, and starts and ends with business use. The correlation among objectives can affect the goodness of the SBSE; contradictory objectives can be more informative; it depends on the effect size of the correlation: very high correlation is maybe not good to have. Related to this is the question how to debug a fitness function.

Goodness must be representative: it needs to represent the search space, and also it needs to be a good representation of the reality.

Efficiency is an important consideration. For computationally expensive problems, active learning can be an efficient approach to cost minimisation. Human input is particularly inefficient, but sometimes unavoidable. To mitigate noise, the problem or function can be run many times.

The discussion of the search landscape raised many points. The landscape needs to aid search, the solutions can exhibit brittleness, and random search can be a good starting point. Domain understanding was considered helpful.

Biasing SBSE with Priors from MSR

Group lead: Abram Hindle

Tasks from SBSE that can be addressed: generate code/input, optimisation, defect prediction, test set generation, triaging.

SBSE can use priors, e.g., by maintaining probabilites rather than populations. Distributions can be learned (estimation of distribution algorithm, EDA), and surrogate models can try to estimate fitness functions. The knowledge mined can be seen as a constraint.

The discussion then focussed on langauge models: n-grams, LSTMs, trees, and graphs in general. For example, Sung Kim uses templates for bug repair that is readable, Deepfix uses LSTMs, and there are a couple of other LSTM/RNN based repair models. Language Models can be used for estimating the "naturalness" or "unnaturalness" of generated code, for ranking code snippets by similarity to a corpus, for generating tokens to fill in holes, and for aligning two language based datasets. Typiacl operators with LM: insert, replace, delete with tokens from a LM. Information Retrieval Models are an alternative TF-IDF and cosine distance to find similar "documents".

In summary, MSR mining data can be used for sanity checks/rankings for search. We should push for realism, natural answers, more humane solutions. The question is how DO we use these resources to do it? An approach is to get a corpus, search it fast and effectively.

The session ended with the question: What is the danger of MSR to SBSE?

MSR/SBSE Degree

Group lead: Meiyappan Nagappan

A lightweight version of a degree could be a MSR/SBSE School for early

PhD/MS students. There would be no prereqs other than you need to know some coding.

It was not easy to focus on a core set of content of such events/degrees: Empirical studies methodologies, Ethics, Qualitative studies, Pitfalls and bias in ESE, Data presentation and visualization, Triangulation of results, Experimental design, Stats Basic/Advanced, Machine learning for SE, SBSE Tools, Quality control for analytics Replication, Paper writing/reviewing, Source code analysis, Feature extraction, Mining unstructured data Language modelling, Tooling Latex/VCS, Data sources, Large scale analysis, Use of cloud facilities, SBSE: Single and Multi-Objectives SBSE History and theory, SBSE Representations.

The following papers are considered essential here:

- Selecting Empirical Methods for Software Engineering Research - Steve Easterbrook, Janice Singer, Margaret-Anne Storey, Daniela Damian
- Raising MSR Researchers: An Experience Report on Teaching a Graduate Seminar Course in Mining Software Repositories (MSR) Ahmed E. Hassan
- Grounded theory in software engineering research - Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald
- A practical guide for using statistical tests to assess randomized algorithms in software engineering - Andrea Arcuri, Lionel Briand
- Perspectives on Data Science for Software Engineering - Tim Menzies, Laurie Williams, Thomas Zimmermann
- Five days of empirical software engineering: The PASED experience - Massimiliano Di Penta et al.

Papers to be written

- Ethics in MSR
- What is good grounded theory no true scottsman considered harmful
- SBSE for MSR dummies *(Note: a paper addressing this was written and published as an outcome of this meeting, see "Outcomes Section")*
- A revisit of the SBSE literature survey with an MSR lens
- Mining and extraction in the new web services world

3 Outcomes

- Paper accepted for publication:

  NAIR, V.; AGRAWAL, A.; CHEN, J.; FU, W.; MATHEW, G.; MENZIES, L.; MINKU, L.; WAGNER, M.; YU, Z. . "Data-Driven Search-Based Software Engineering", 2018, accepted for publication at Mining Software Repositories, preprint: `https://arxiv.org/abs/1801.10241`.

- Abram Hindle will visit Yasukata Kamei for four months from April 2018 under a JSPS Fellowship. Their application was submitted before the Shonan Meeting, but they feel that the discussions they had in the Shonan Meeting will accelerate their research during Abram's visit.

- The presentations from the seminar have been collected and shared among the seminar attendees via a shared GoogleDrive folder.

4 Attendees

First name	Family name	Affiliation	Country	Email
Markus	Wagner	University of Adelaide	Australia	markus.wagner@adelaide.edu.au
John	Clark	University of Sheffield	UK	john.clark@sheffield.ac.uk
Ahmed	Hassan	Queen's University	Canada	ahmed@cs.queensu.ca
Leandro	Minku	University of Leicester	UK	leandro.minku@leicester.ac.uk
Yasutaka	Kamei	Kyushu University	Japan	kamei@ait.kyushu-u.ac.jp
Hironori	Washizaki	Waseda University	Japan	washizaki@waseda.jp
Shin	Yoo	Korea Advanced Institute of Science and Technology	Korea	shin.yoo@cs.kaist.ac.kr
Christoph	Treude	University of Adelaide	Spain	christoph.treude@adelaide.edu.au
Tim	Menzies	North Carolina State University	USA	tim.menzies@gmail.com
Burak	Turhan	University of Oulu	Finland	Burak.Turhan@oulu.fi
Tomas	Zimmermann	Microsoft Research	USA	tzimmer@microsoft.com
Meiyappan	Nagappan	Rochester Institute of Technology	USA	mei@se.rit.edu
Audris	Mockus	University of Tennessee	USA	audris@utk.edu
Shane	McIntosh	McGill University	Canada	shane.mcintosh@mcgill.ca
Abram	Hindle	University of Alberta	Canada	web@softwareprocess.es
Xin	Yao	University of Birmingham	UK	X.Yao@cs.bham.ac.uk
Marius	Lindauer	Freiburg University	Germany	marius.rks@googlemail.com
Hideaki	Hata	Nara Institute of Science and Technology	Japan	hata@is.naist.jp
Brad	Alexander	University of Adelaide	Spain	brad@cs.adelaide.edu.au
Barbara	Russo	Free University of Bolzano	Italy	brusso@unibz.it
Dionidis	Spinellis	Athens University of Economics and Business	Greece	dds@aueb.gr
Neil	Walkinshaw	University of Leicester	UK	nw91@le.ac.uk
Daniel	German	University of Victoria	Canada	dmg@uvic.ca
Peter Tse-Hsun	Chen	Concordia University	Canada	petertsehsun@gmail.com
Masanari	Kondo	Kyoto Institute of Technology	Japan	m7622015@edu.kit.ac.jp
Prem	Devanbu	UC Davis	USA	devanbu@cs.ucdavis.edu
Xin	Xia	University of British Columbia	Canada	xxia02@cs.ubc.ca
Yutaro	Kashiwa	Wakayama University	Japan	kashiwa.yutaro@g.wakayama-u.jp

Meeting Schedule

Check-in Day: 10 December 2017 (Sun)

- Welcome Banquet

Day1: 11 December 2017 (Mon)

- Talks and Discussions
- Group Photo Shooting

Day2: 12 December 2017 (Tue)

- Talks and Discussions

Day3: 13 December 2017 (Wed)

- Talks and Discussions
- Excursion with tea ceremony and Main Banquet

Day4: 14 December 2017 (Thu)

- Talks and Discussions
- Wrap up