Lecture 2 Machine learning in Software Engineering

- Existing approaches. Recent trends in SBSE
- Reformulate software engineering problems
- Evaluation criteria for search-based software engineering

Lecture 1 Search-based Software Engineering

- Search-based Software Engineering
- Machine learning
  - techniques
- Software Engineering
  - subdisciplines
  - activities
  - artifacts (UML, sorce code, source repository, test data, runtime data,..)

Machine learning in Software Engineering

ML algorithms can be used in tackling software engineering problems.

ML algorithms not only can be used to build tools for software development and maintenance tasks, but also can be incorporated into software products to make them adaptive and self-configuring.

A maturing software engineering discipline will be able to benefit from the utility of ML techniques.

Search Based Software Engineering

Reformulating software engineering problems as search problems

Machine learning approaches for software engineering

- Requirement engineering - AL, BBN, LL, DT, ILP
- Rapid prototyping - GP
- Component reuse - IBL (CBR4)
- Cost/effort prediction - IBL (CBR), DT, BBN, ANN
- Defect prediction – BBN, AR
- Design defect detection - AR
- Test oracle generation - AL (EBL5)
- Test data adequacy - CL
- Validation – AL
- Restructuring/Refactoring CU
- Design patterns – CU
- Data structures – ANN, SVM
- Reverse engineering – CL

Major types of learning:

- concept learning (CL)
- decision trees (DT)
- artificia neural networks (ANN)
- Bayesian belief networks (BBN)
- reinforcement learning (RL)
- genetic algorithms (GA) and genetic programming (GP)
- instance-based learning (IBL)
- inductive logic programming (ILP)
- analytical learning (AL)
- support vector machines (SVM)
- association rules (AR)
- clustering (CU)

Mark Harman - Recent Advances in Search Based Software Testing and Genetic Improvement

https://www.youtube.com/watch?v=y4xWZj1ocDo

Overview of SBSE. Details on search-based software testing

SBSE REPOSITORY

collects the work which address the software engineering problems using metaheuristic search optimisation techniques

http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/

SBSE Challenge

Participants are invited to investigate and report upon one of the following open source projects. You are free to focus on any particular version or a comparison of different versions; you can also choose to analyse, test, improve, or apply any other SBSE-based activities to either parts or the whole of a project, including source code, documentation, or any other related resources (bug database, versioning histories, online discussions, etc) that can be found freely available online.

LibreOffice https://www.libreoffice.org

LibreOffice is a large open-source productivity suite, implemented in several languages including C++, with a total of 8MLOC. The project incorporates three levels of regression testing.

SQLite https://www.sqlite.org

SQLite is arguably the most popular database in the world. It is designed for efficiency, simplicity, and can be deployed as a single C source code file. The project incorporates 338KLOC and three separately developed test suites.

Guava https://github.com/google/guava

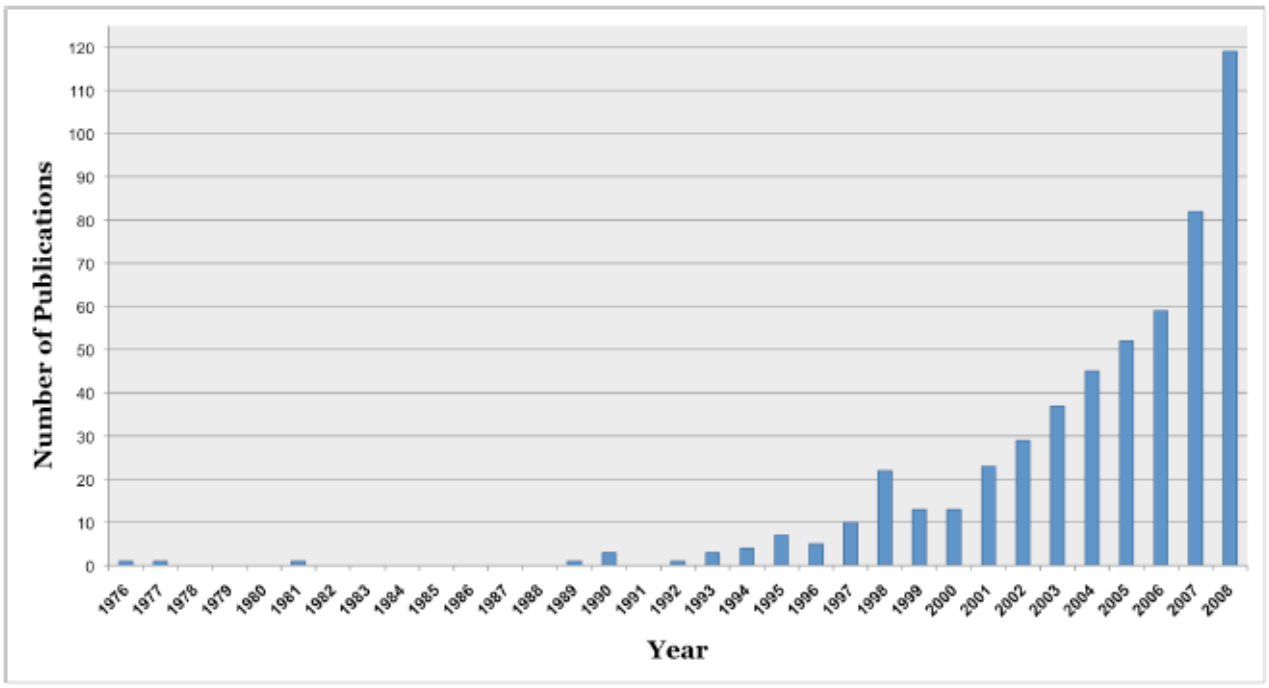
Guava is a widely adopted and extensive Java collections library developed by Google. It includes over 252KLOC and its test suite includes a thorough set of performance tests.

Flask http://flask.pocoo.org

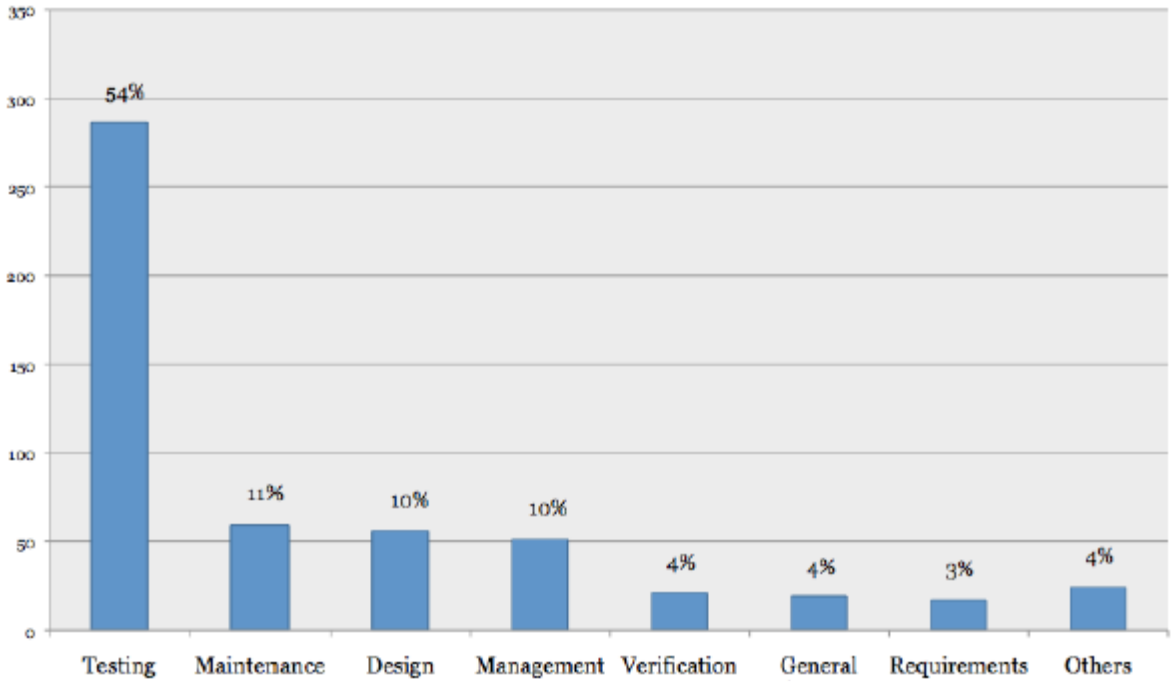
Flask is a very popular minimalist web framework for Python, with 9KLOC. It comes with a full test suite.

Search Based Software Engineering: Trends, Techniques and Applications

Mark Harman, University College London; S. Afshin Mansouri, Brunel University; Yuanyuan Zhang, University College London

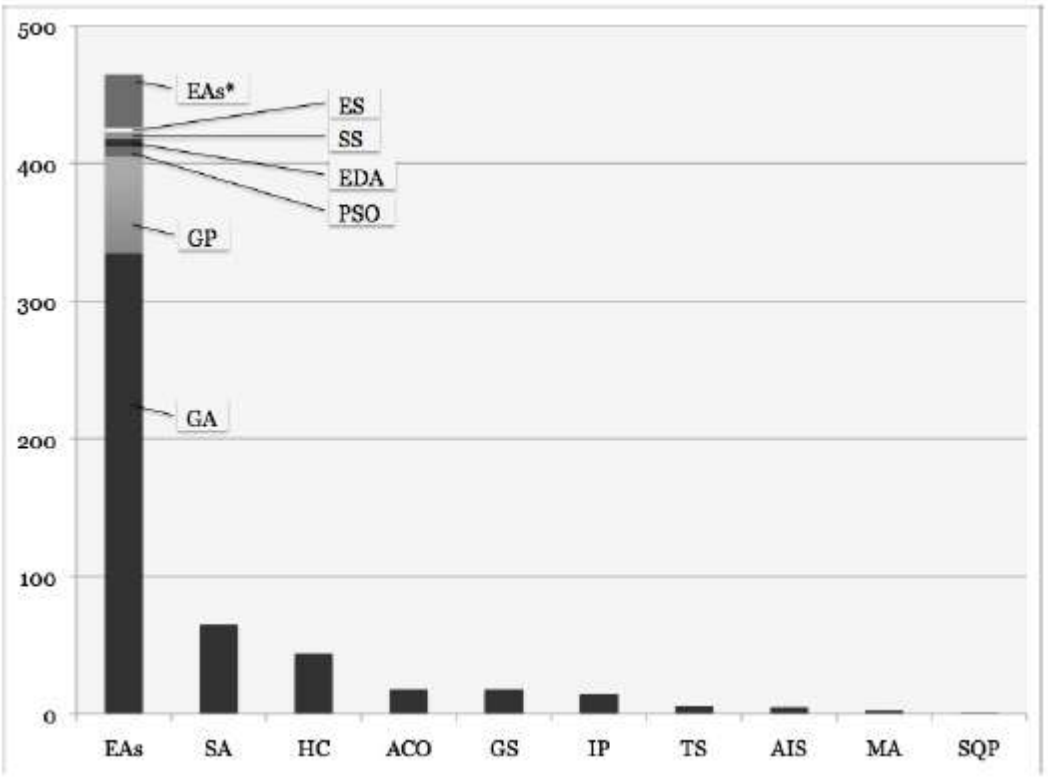


Publication numbers



Topic areas

Search Based Software Engineering: Trends, Techniques and Applications



Numbers of papers using each of the different types of SBO techniques: EAs are split into GA, GP, ES, SS, EDA, PSO and EAs. In this figure the stacked bar 'EAs' represents the general class of all Evolutionary Algorithms, while the top portion of bar labelled 'EAs*' refers to the proportion of the literature that describes itself as using an 'evolutionary algorithm', without further qualification or specification as to the type of evolutionary algorithm used.

EAs = Evolutionary Algorithms (including GAs, GP, ES, PSO, EDA and SS)
GA = Genetic Algorithms
GP = Genetic Programming
ES = Evolutionary Strategies
PSO = Particle Swarm Optimization
EDA = Estimation of Distribution Algorithms
SS = Scatter Search Algorithm

ACO = Ant Colony Optimization
AIS = Artificial Immune Systems
GS = Greedy Search
HC = Hill Climbing
IP = Integer Programming
MA = Memetic Algorithms
SA = Simulated Annealing
SQP = Sequential Quadric Programming
TS = Tabu Search

Achievements, Open Problems and Challenges for Search Based Software Testing
M. Harman, Yue Jia, Yuanyuan Zhang

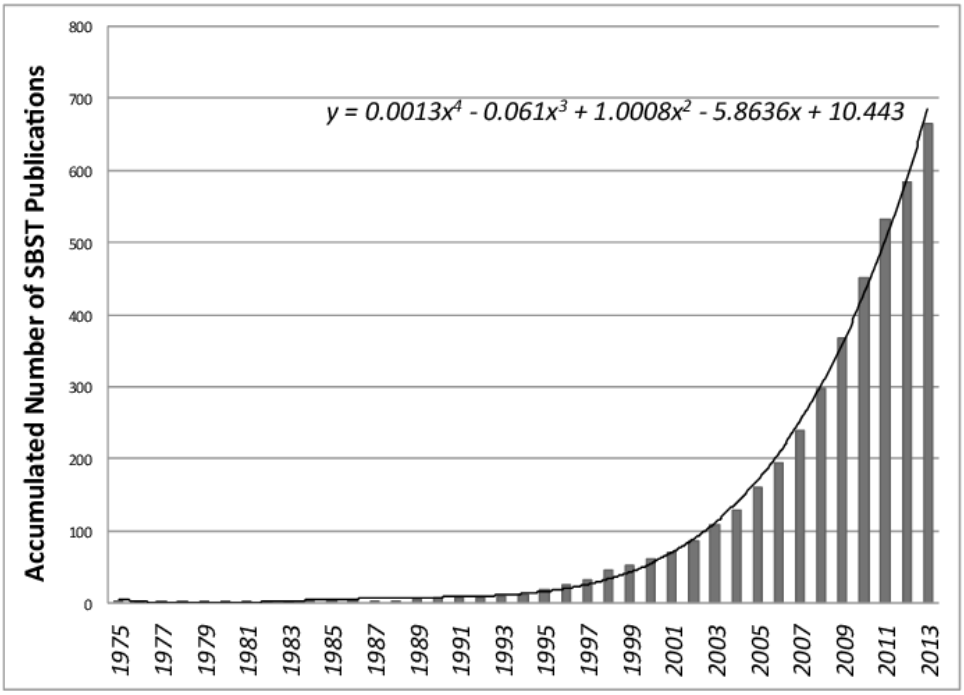
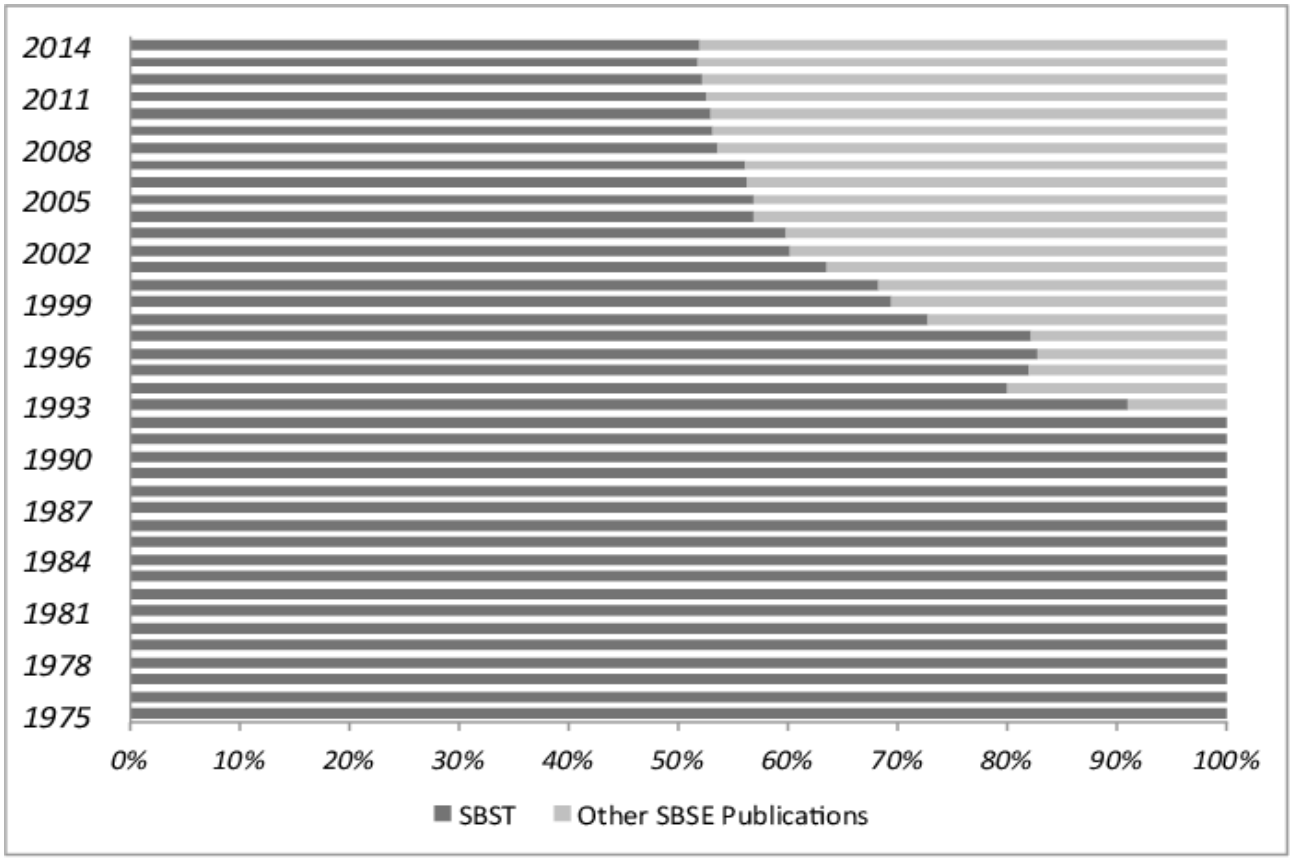


Fig. 1: Cumulative number of Search Based Software Testing

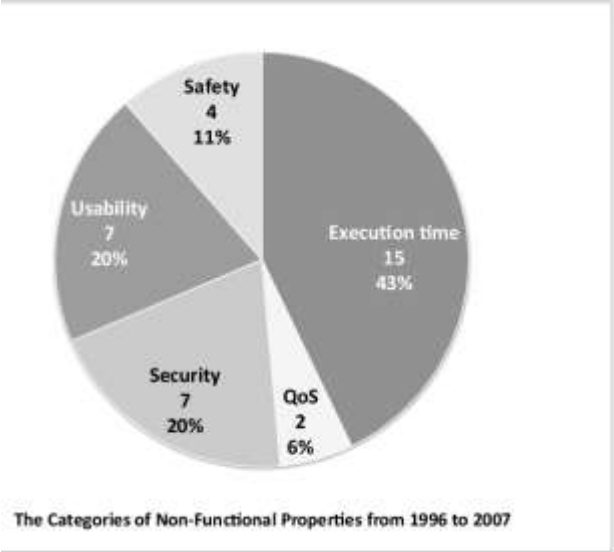
Cumulative number of Search Based Software Testing papers. As can be seen, the overall trend continues to suggest a polynomial yearly rise in the number of papers, highlighting the breadth of interest and strong health of SBST



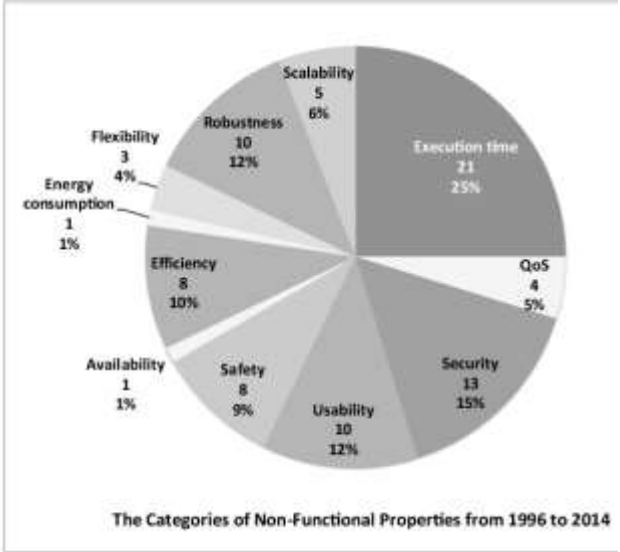
The changing ratio of SBSE papers that are SBST papers. Initially, SBST dominated SBSE. Over the years, this ratio has decreased, stabilising at around 50%. This represents the growth in non-testing related areas of SBSE rather than any decline in the number of papers on SBST

Achievements, Open Problems and Challenges for Search Based Software Testing

M. Harman, Yue Jia, Yuanyuan Zhang



The Categories of Non-Functional Properties from 1996 to 2007



The Categories of Non-Functional Properties from 1996 to 2014

Accepted Papers in SSBSE 2017

Full Papers

- *Andrea Arcuri.* Many Independent Objective (MIO) Algorithm for Test Suite Generation
- *Matteo Biagiola, Filippo Ricca and Paolo Tonella.* Search Based Path and Input Data Generation for Web Application Testing
- *José Campos, Yan Ge, Gordon Fraser, Marcelo Eler and Andrea Arcuri.* An Empirical Evaluation of Evolutionary Algorithms for Test Suite Generation
- *Byron Devries and Betty Cheng.* Automatic Detection of Incomplete Requirements using Symbolic Analysis and Evolutionary Computation
- *Gregory Gay.* Generating Effective Test Suites by Combining Coverage Criteria
- *Annibale Panichella, Fitsum Meshesha Kifetew and Paolo Tonella.* LIPS vs MOSA: a Replicated Empirical Study on Automated Test Case Generation
- *Christopher Steven Timperley, Susan Stepney and Claire Le Goues.* An Investigation into the Use of Mutation Analysis for Automated Program Repair

Short Papers

- *Vineeth Kashyap, Rebecca Swords, Eric Schulte and David Melski.* MuSynth: Program Synthesis via Code Reuse and Code Manipulation
- *Elias Khalil, Mustafa Assaf and Abdel Salam Sayyad.* Human Resource Optimization for Bug Fixing: Balancing Short-Term and Long-Term Objectives
- *Fitsum Meshesha Kifetew, Denisse Muñante, Jesús Gorroñogoitia, Alberto Siena, Angelo Susi and Anna Perini.* Grammar Based Genetic Programming for Software Configuration Problem
- *Jinhan Kim, Junhwi Kim and Shin Yoo.* GPGPGPU: Evaluation of Parallelisation of Genetic Programming using GPGPU
- *Junhwi Kim, Byeonghyeon You, Minhyuk Kwon, Phil McMinn and Shin Yoo.* Evaluating CAVM: A New Search-Based Test Data Generation Tool for C

SSBSE 2018

Mark Harman. Deploying Search Based Software Engineering with Sapienz at Facebook.

Edouard Batot and Houari Sahraoui*Injecting Social Diversity in Multi-Objective Genetic Programming: the Case of Model Well-formedness Rule Learning*(R)

Bogdan Korel, Nada Almasri and Luay Tahat*Towards Minimizing the Impact of Changes using Search-Based Approach*(R)

Wael Kessentini, Houari Sahraoui and Manuel Wimmer*Automated Co-Evolution of Meta-models and Transformation Rules: A Search-Based Approach*(R)

Marouane Kessentini. Search-based Software Refactoring.

David Issa Mattos, Erling Mårtensson, Jan Bosch and Helena Holmström Olsson*Optimization Experiments in the Continuous Space: The Limited Growth Optimistic Optimization Algorithm*(R)

Kate M. Bowers, Erik M. Fredericks and Betty H. C. Cheng*Automated Optimization of Weighted Non-Functional Objectives in Self-Adaptive Systems*(R)

Irene Manotas, James Clause and Lori Pollock*Exploring Evolutionary Search Strategies to Improve Applications' Energy Efficiency*(R)

Thomas Jones and David Ackley*Damage Reduction via White-Box Failure Shaping*(R)

Mengmei Ye, Myra B. Cohen, Witawas Srisa-An and Sheng Wei*EvoIsolator: Evolving Hardware Isolation for Software Security*(H)

William B. Langdon and Justyna Petke*Evolving Better Software Parameters*(H)

Matías Martínez. Astor: An automated software repair framework.

Jinhan Kim, Michael G. Epitropakis and Shin Yoo*Learning Without Peeking: Secure Multi-Party Computation Genetic Programming*(R)

Kabdo Choi, Jeongju Sohn and Shin Yoo*Learning Fault Localisation for Both Humans and Machines using Multi-Objective GP*(H)

Gregory Gay*Detecting Real Faults in the Gson Library Through Search-Based Unit Test Generation*(C)

Mozhan Soltani, Pouria Derakhshanfar, Annibale Panichella, Xavier Devroey, Andy Zaidman and Arie van Deursen*Single-objective versus Multi-Objectivized Optimization for Evolutionary Crash Reproduction*(R)

Annibale Panichella, Fitsum Meshesha Kifetew and Paolo Tonella*Incremental Control Dependency Frontier Exploration for Many-Criteria Test Case Generation*(R)

Allen Kanapala and Gregory Gay*Mapping Class Dependencies for Fun and Profit*(H)

Using Search-Based Test Generation to Discover Real Faults in Guava

Hussein Almulla, Alireza Salahirad, Gregory Gay

The Guava project—a collection of Java libraries from Google
Guava is a set of core libraries that includes new collection types (such as multimap and multiset), immutable collections, a graph library, functional types, an in-memory cache, and APIs/utilities for concurrency, I/O, hashing, primitives, reflection, string processing, etc

EvoSuite (http://www.evosuite.org) is a tool that automatically generates test cases with assertions for classes written in Java code. To achieve this, EvoSuite applies a novel hybrid approach that generates and optimizes whole test suites towards satisfying a coverage criterion.

Identified 11 faults in the Guava project, added them to Defects4J, and assessed the ability of the EvoSuite framework to detect these faults

Just, R., Jalali, D., Ernst, M.D.: 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. ISSTA 2014, ACM, New York, NY, USA (2014)

Defects4J (https://github.com/rjust/defects4j), a database and extensible framework providing real bugs to enable reproducible studies in software testing research. The initial version of Defects4J contains 357 real bugs from 5 real-world open source programs. Each real bug is accompanied by a comprehensive test suite that can expose (demonstrate) that bug.

Contents of Defects4J

Defects4J contains 395 bugs from the following open-source projects:

Identifier	Project name	Number of bugs
Chart	JfreeChart	26
Closure	Closure compiler	133
Lang	Apache commons-lang	65
Math	Apache commons-math	106
Mockito	Mockito	38
Time	Joda-Time	27

Each bug has the following properties:

- Issue filed in the corresponding issue tracker, and issue tracker identifier mentioned in the fixing commit message.
- Fixed in a single commit
- Minimized: the Defects4J maintainers manually pruned out irrelevant changes in the commit (e.g., refactorings or feature additions).
- Fixed by modifying the source code (as opposed to configuration files, documentation, or test files).
- A triggering test exists that failed before the fix and passes after the fix -- the test failure is not random or dependent on test execution order.

Deploying Search Based Software Engineering with Sapienz at Facebook

Nadia Alshahwan, Xinbo Gao, Mark Harman(B) , Yue Jia, Ke Mao, Alexander Mols, Taijin Tei, and Ilya Zorin

describe the deployment of the Sapienz Search Based Software Engineering (SBSE) testing system. Sapienz has been deployed in production at Facebook since September 2017 to design test cases, localise and triage crashes to developers and to monitor their fixes.

Sapienz, an approach to Android testing that uses multi-objective search-based testing to automatically explore and optimise test sequences, minimising length, while simultaneously maximising coverage and fault revelation

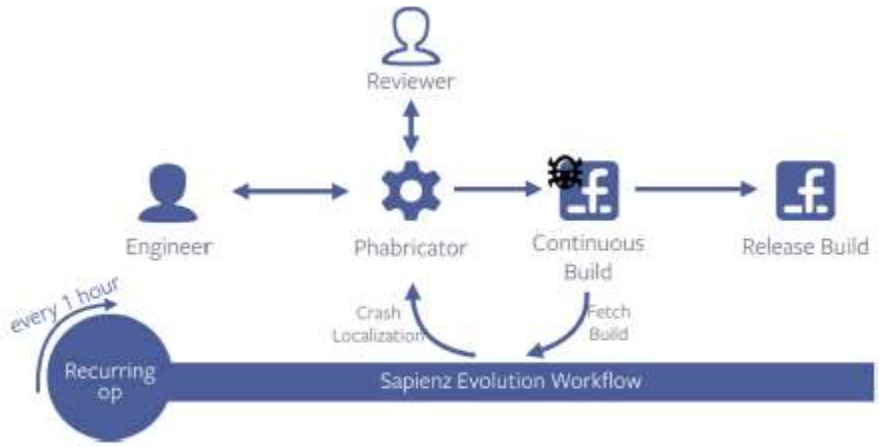


Fig. 1. Overall deployment mode for Sapienz at Facebook

Component reuse

Component retrieval from a software repository is an important issue in supporting software reuse.

Formulated into an instance-based learning problem:
1. Components in a software repository are represented as points in the n-dimensional Euclidean space (or cases in a case base).

2. Information in a component can be divided into *indexed* and *unindexed* information (attributes). Indexed information is used for retrieval purpose and unindexed information is used for contextual purpose.

3. Queries to the repository for desirable components can be represented as constraints on indexable attributes.

4. Similarity measures for the nearest neighbors of the desirable component can be based on the standard Euclidean distance, distance-weighted measure, or symbolic measure.

5. The possible retrieval methods include: *K-Nearest Neighbor*, *inductive retrieval*, *Locally Weighted Regression*.

6. The adaptation of the retrieved component for the task at hand can be *structural* (applying adaptation rules directly to the retrieved component), or *derivational* (reusing adaptation rules that generated the original solution to produce a new solution).

Rapid prototyping/Generating programs

Rapid prototyping - tool for understanding and validating software requirements.

In genetic programming (GP), a computer program is often represented as a tree:

- nodes are functions
- leaf nodes are input to functions

Start with a random generated tree (program), GP generates the final computer program.

The framework of a GP-based rapid prototyping process can be described as follows:

1. Define sets of functions and terminals to be used in the developed (prototype) systems.

2. Define a *fitness* function to be used in evaluating the worthiness of a generated program. Test data (input values and expected output) may be needed in assisting the evaluation.

3. Generate the initial program population.

4. Determine selection strategies for programs in the current generation to be included in the next generation population.

5. Decide how the genetic operations (*crossover* and *mutation*) are carried out during each generation and how often these operations are performed.

6. Specify the terminating criteria for the evolution process and the way of checking for termination.

7. Translate the returned program into a desired programming language format.

Requirement engineering

Requirement engineering refers to the process of establishing the services a system should provide and the constraints under which it must operate.

A requirement may be functional or non-functional. A functional requirement describes a system service or function, whereas a non-functional requirement represents a constraint imposed on the system.

Functional requirements can be “learned” from the data if there are empirical data from the problem domain that describe how the system should react to certain inputs .

1. Let X and C be the domain and the co-domain of a system function f to be learned. The data set D is defined as: D = {<xi, ck>| xi in X and ck in C}.

2. The target functions f to be learned is such that any xi in X and any ck in C, f(xi) = ck .

3. The learning methods applicable here have to be of supervised type. Depending on the nature of the data set D, different learning algorithms (in AL, BBN, CL, DT, ILP) can be utilized to capture (learn) a system’s functional requirements.

Reverse engineering (program comprehension and understanding)

Recover the design or specification of a legacy system from its source or executable code

Legacy systems are old systems that are critical to the operation of an organization which uses them and that must still be maintained. They may be poorly structured and their documentation may be either out-of-date or non-existent.

Deriving functional specification of a legacy software system from its executable code.

1. Given the executable code p and its input data set X, and output set C, the training data set D is defined as: D = {< xi, p(xi)>| xi in X and p(xi) in C}.

2. The process of deriving the functional specification f for p can be described as a learning problem in which f is learned through some ML algorithm such that any xi in X [f(xi) = p(xi)].

3. Many supervised learning methods can be used here (e.g., CL).

Validation

Check a software implementation against its specification

If the specification and the executable code is available validation can be performed as an analytic learning task as follows:

1. Let X and C be the domain and co-domain of the implementation (executable code) p, which is defined as: p: X -> C.

2. The training set D is defined as: D = {<xi, p(xi)>| xi in X }.

3. The specification for p is denoted as B, which corresponds to the domain theory in the analytic learning.

4. The validation checking is defined to be: p is valid if any <xi, p(xi)> in D [B and xi imply p(xi)].

5. Explanation-based learning algorithms can be utilized to carry out the checking process.

Software defect prediction

predict the likely delivered quality and maintenance effort before software system is deployed

size, complexity, testing metics, proces quality must be taken into consideration for the defect prediction to be successful.

compute the probability distribution for any subset of variables given the values or distributions for any subset of the remaining variables - Bayesian Belief Networks (BBN)

specifies the probability that the variable will take on each of its possible values (e.g., “very low”, “low”, “medium”, “high”, or “very high” for the variable “Defects Detected”) given the observed values of the other variables (complexity, metrics, etc)

When using a BBN for a decision support system such as software defect prediction, the steps below should be followed.

1. Identify variables in the BBN. Variables can be:
   - *hypothesis* variables for which the user would like to find out their probability distributions (hypothesis variable are either unobservable or too costly to observe),
   - *information* variables that can be observed
   - *mediating* variables that are introduced for certain purpose (help reflect independence properties, facilitate acquisition of conditional probabilities, and so forth).

2. Define the proper causal relationships among variables. These relationships also should capture and reflect the causality exhibited in the software life-cycle processes.

3. Acquire a probability distribution for each variable in the BBN.

Software defect prediction

use any supervised learning technique

- Collect data (measurement, metric, resource allocation) about defective systems, components, classes, modules. The measurements will be used as attribute values
- train the model using well know defective/un-defective system/component/ class
- The model can be used to predict defect rate for new systems (other than the ones used in the training phase)

Project effort (cost) prediction

approach to the project effort prediction using instance-based learning.

1. Introduce a set of features or attributes (e.g., number of interfaces, size of functional requirements, development tools and methods, and so forth) to characterize projects.

2. Collect data on completed projects and store them as instances in the case base.

3. Define similarity or distance between instances in the case base according to the symbolic representations of instances (e.g., Euclidean distance in an n-dimensional space where n is the number of features used). To overcome the potential curse of dimensionality problem, features may be weighed differently when calculating the distance (or similarity) between two instances.

4. Given a query for predicting the effort of a new project, use an algorithm such as Knearest Neighbor, or, Locally Weighted Regression to retrieve similar projects and use them as the basis for returning the prediction result.

Software restructuring/refactoring/transformation

Model the system as a set of objects.

Object can be a class, a module, a function/method, a package.

Use a clustering algorithm to group the objects.

Using the obtained partition one can:

- identify the new/better structure of the system
- derive transformations for the analyzed system
- identify components, concepts or other higher-level abstraction in the source code (i.e design patterns)

Object in the software system can be described by a set of features (i.e. software metric) -> we can use Euclidian or other distances

We can define custom distances between the objects. We can encapsulate domain knowledge in the definition of the distance used in the clustering algorithm.

Some Existing Work

Scenario-based requirement engineering

Formal method for supporting the process of inferring specifications of system goals and requirements inductively from interaction scenarios provided by stakeholders.

The method is based on a learning algorithm that takes scenarios as examples and counterexamples (positive and negative scenarios) and generates goal specifications as temporal rules.

Software project effort estimation

Instance-based learning techniques are used in for predicting the software project effort for new projects.
The empirical results obtained (from nine different industrial data sets totaling 275 projects) indicate that case-based reasoning offers a viable complement to the existing prediction and estimations techniques.

Decision trees (DT) and artificial neural networks (ANN) are used to help predict software development effort. The results were competitive with conventional methods such as COCOMO and function points. The main advantage of DT and ANN based estimation systems is that they are adaptable and nonparametric.

Software defect prediction

Bayesian belief networks are used to predict software defects.
incorporating multiple perspectives on defect prediction into a single, unified model.

Variables are chosen to represent the life-cycle processes of

- specification
- design
- implementation
- testing

(Problem-Complexity, Design-Effort, Design-Size, Defects-Introduced, Testing-Effort, Defects-Detected, Defects-Density-At-Testing, Residual-Defect-Count, and Residual-Defect-Density).

Evaluation criteria for search-based software engineering

Criteria that capture essential aspects/goals for any search-based approach

Base line validity

The solution should be able to find better solutions or find the in less computational effort than random search

In some case maybe we just must change the choice of search technique

Discovery of known solutions

There are situations where there is no known general algorithm for solving a problem.

We can validate the search-based approach by showing that the metaheuristic search is able to find solutions that compare well with known individual solutions.

Discovery of desirable solutions

Gather empirical data to provide evidence of the kind of solution which may be obtained.

Evaluation criteria for search-based software engineering

Efficiency

In many cases a search-based approach is slower than existing analytical approaches (search involve repeated trails)

If the search-based approach produces better solutions then it can be a valuable tool where quality overrides speed.

Validation with respect to existing analytical techniques (no search-based)

Even if there are non-search-based approaches the search-based variant may still be useful if existent approaches:

only work on subset of the problem space

solution is not consistent, we can use the search-based variant as a "second guess"

only produce sub-optimal solutions, we can use search-based variant to improve on the known solution or to produce better solutions

Psychological consideration

aesthetic application of metaheuristic search

(semi) automatization of common software engineering task

We can use search-based to better understand solutions