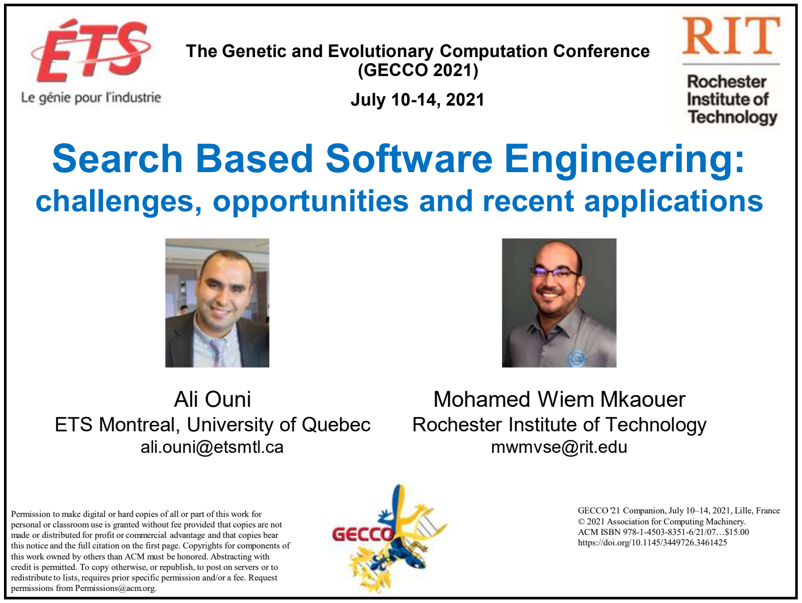

ÉTS
Le génie pour l'industrie
RIT
Rochester Institute of Technology
GECCO

The Genetic and Evolutionary Computation Conference
(GECCO 2021)

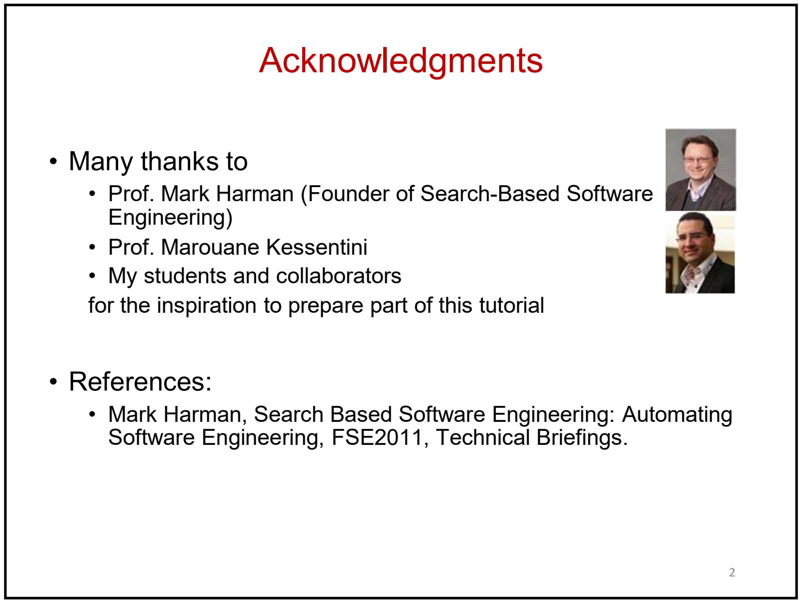
July 10-14, 2021

Search Based Software Engineering:
challenges, opportunities and recent applications

Ali Ouni
ETS Montreal, University of Quebec
ali.ouni@etsmtl.ca

Mohamed Wiem Mkaouer
Rochester Institute of Technology
mwmvse@rit.edu

GECCO '21 Companion, July 10–14, 2021, Lille, France
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3461425

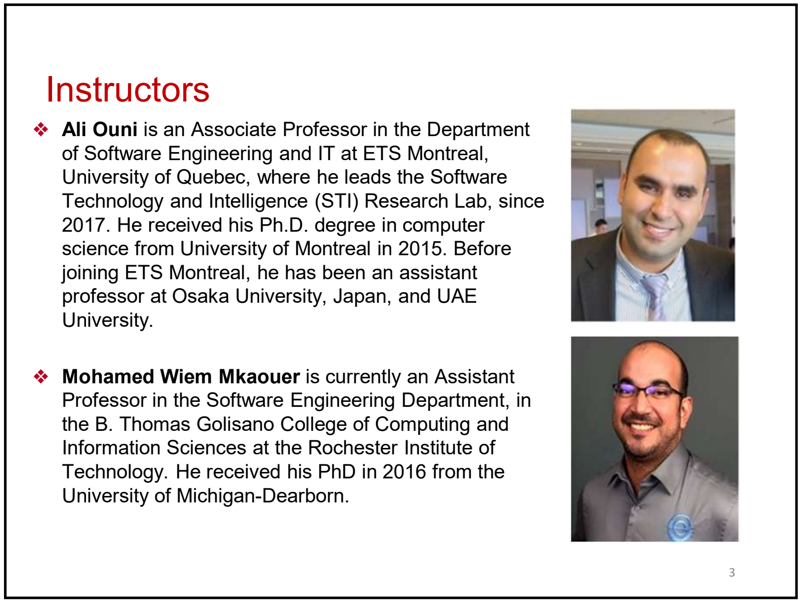


Acknowledgments

- Many thanks to
  - Prof. Mark Harman (Founder of Search-Based Software Engineering)
  - Prof. Marouane Kessentini
  - My students and collaborators

  for the inspiration to prepare part of this tutorial

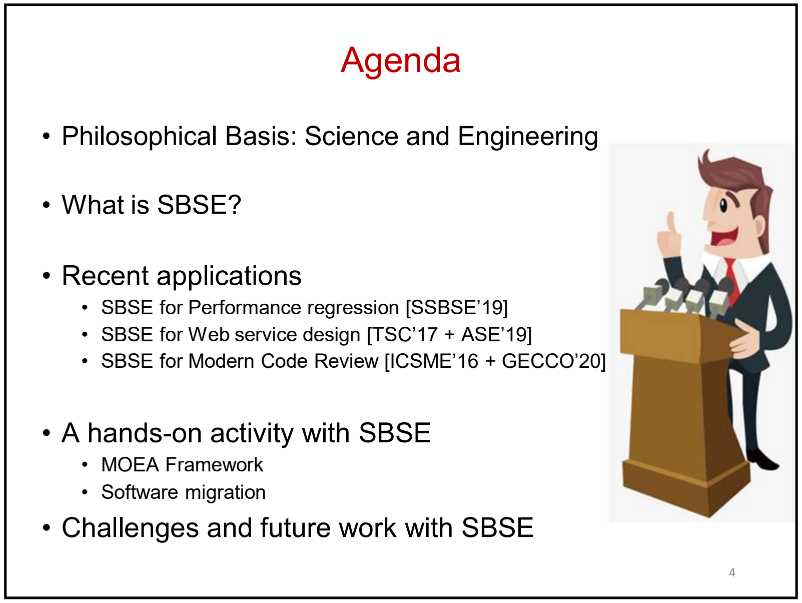
- References:
  - Mark Harman, Search Based Software Engineering: Automating Software Engineering, FSE2011, Technical Briefings.



Instructors

❖ **Ali Ouni** is an Associate Professor in the Department of Software Engineering and IT at ETS Montreal, University of Quebec, where he leads the Software Technology and Intelligence (STI) Research Lab, since 2017. He received his Ph.D. degree in computer science from University of Montreal in 2015. Before joining ETS Montreal, he has been an assistant professor at Osaka University, Japan, and UAE University.

❖ **Mohamed Wiem Mkaouer** is currently an Assistant Professor in the Software Engineering Department, in the B. Thomas Golisano College of Computing and Information Sciences at the Rochester Institute of Technology. He received his PhD in 2016 from the University of Michigan-Dearborn.



Agenda

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent applications
  - SBSE for Performance regression [SSBSE'19]
  - SBSE for Web service design [TSC'17 + ASE'19]
  - SBSE for Modern Code Review [ICSME'16 + GECCO'20]
- A hands-on activity with SBSE
  - MOEA Framework
  - Software migration
- Challenges and future work with SBSE

Scientists' and Engineers' Viewpoints

Scientist:

- What is true
- Correctness
- Model the world to understand

Question:
How things are <u>theoretically</u> done?

Engineer:

- What is possible
- Within tolerance
- Model the world to manipulate it

Question:
How things are <u>practically</u> done?

Scientists' and Engineers' Viewpoints

Computer **Scientist:**

- What is true about computation
- Proof correctness
- Make it perfect

Software **Engineer:**

- What is possible with software
- Test for imperfection
- Find where to improve

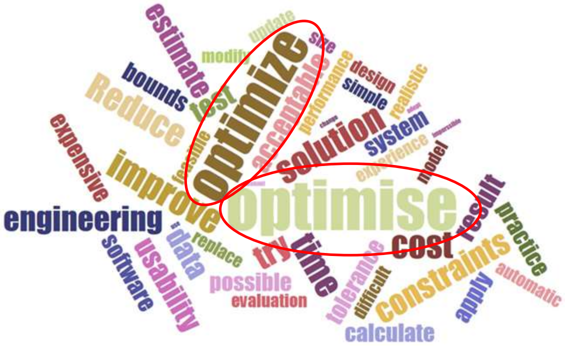
Agenda

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent applications
  - SBSE for Performance regression [SSBSE'19]
  - SBSE for Web service design [TSC'17 + ASE'19]
  - SBSE for Modern Code Review [ICSME'16 + GECCO'20]
- A hands-on activity with SBSE
  - MOEA Framework
- Challenges and future work with SBSE

Engineering Words

estimate
update
modify
size
bounds
Reduce
test
optimize
acceptable
performance
design
simple
realistic
solution
system
experience
model
expensive
improve
feasible
optimise
engineering
result
practice
software
usability
data
replace
try
time
cost
constraints
automatic
possible
evaluation
tolerance
difficult
apply
calculate

Engineering Words



Optimise
Optimize

so good they named it twice! → SBSE

What is SBSE ?

The term “Search-Based Software Engineering” (SBSE) coined in 2001 by Mark Harman.

- SBSE uses intelligent search techniques to explore large search spaces, guided by a fitness function that captures properties of the desirable solutions we seek.

Genetic Programming

Ant Colonies

Harmony Search

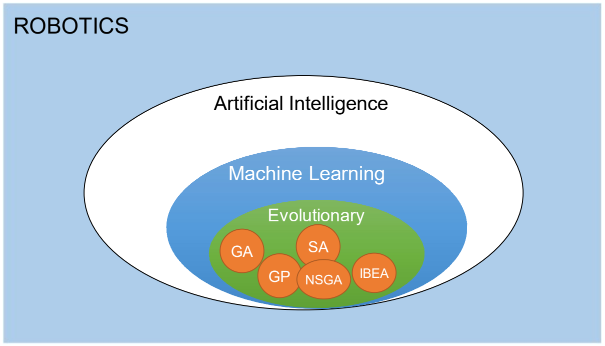
Hill Climbing

Particle Swarm Optimization

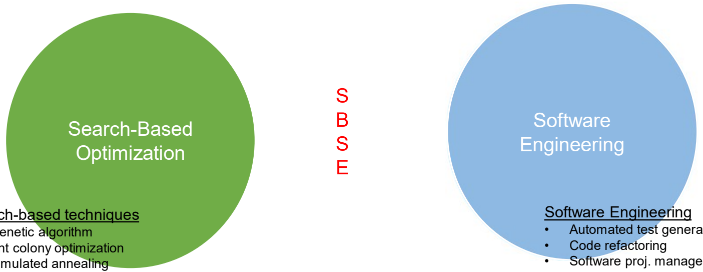
Tabu Search

Simulated Annealing

What are search algorithms?



What is SBSE ?



Search-based techniques

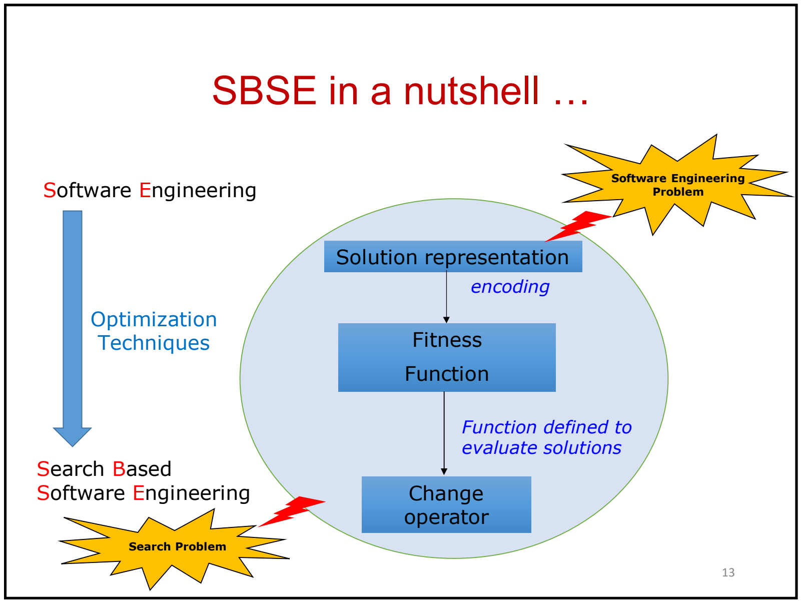
- Genetic algorithm
- Ant colony optimization
- Simulated annealing
- Particle swarm optimization
- NSGA-II
- NSGA-III
- . . .

Software Engineering

- Automated test generation
- Code refactoring
- Software proj. management
- Requirements engineering
- Fault localization
- Program repair
- . . .

SBSE

- Search-based Testing
- Search-based refactoring
- Search-based software management
- Search-based requirements engineering
- Search-based Fault Location
- Search-based program repair
- . . .



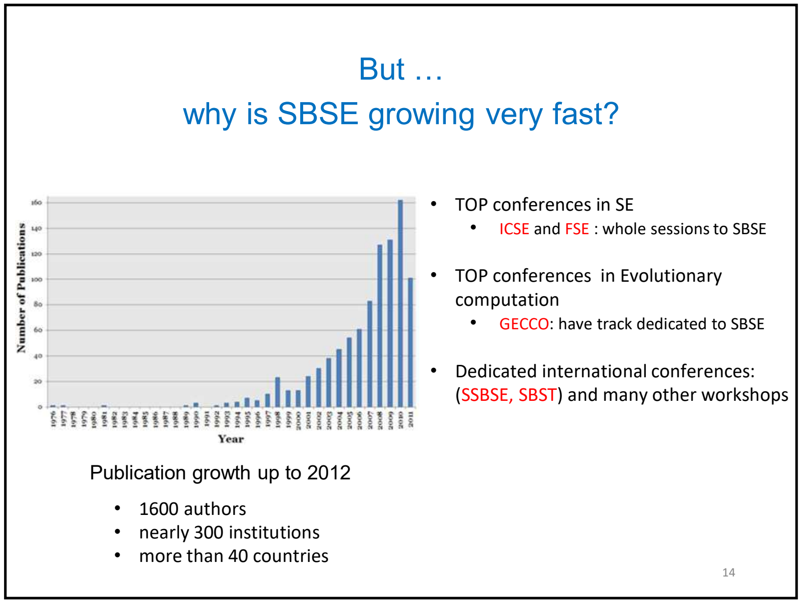
SBSE in a nutshell …

Software Engineering

Optimization Techniques

Search Based Software Engineering

13



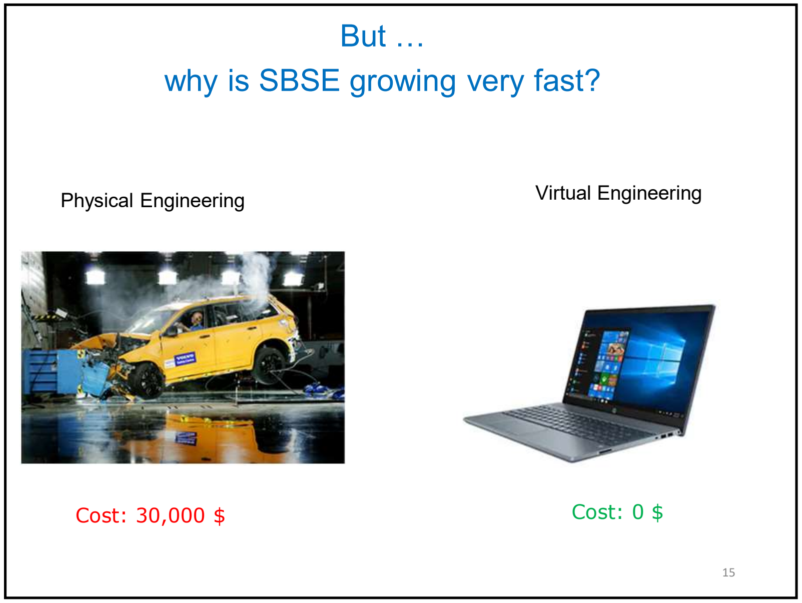
But …
why is SBSE growing very fast?

- TOP conferences in SE
  - ICSE and FSE : whole sessions to SBSE
- TOP conferences in Evolutionary computation
  - GECCO: have track dedicated to SBSE
- Dedicated international conferences: (SSBSE, SBST) and many other workshops

Publication growth up to 2012

- 1600 authors
- nearly 300 institutions
- more than 40 countries

14



But …
why is SBSE growing very fast?

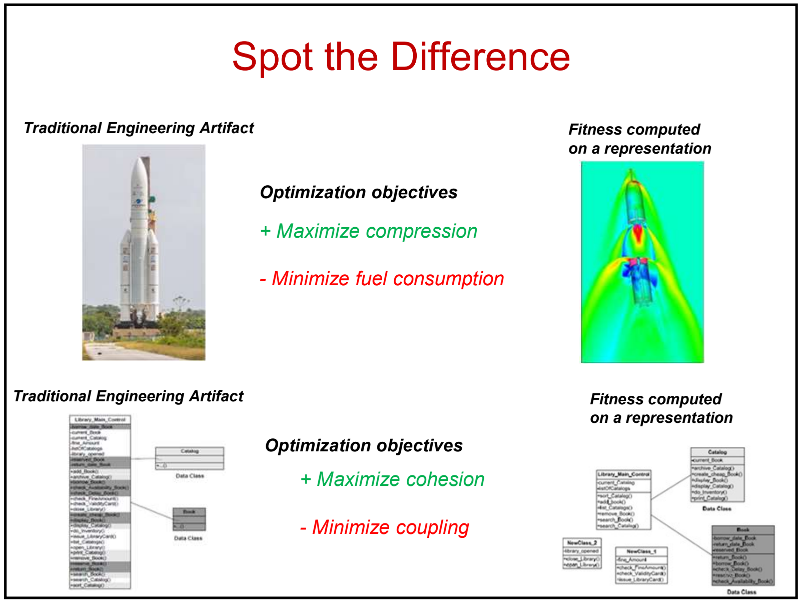
Physical Engineering

Virtual Engineering

Cost: 30,000 $

Cost: 0 $

15



Spot the Difference

Traditional Engineering Artifact

Optimization objectives

+ Maximize compression

- Minimize fuel consumption

Fitness computed on a representation

Traditional Engineering Artifact

Optimization objectives

+ Maximize cohesion

- Minimize coupling

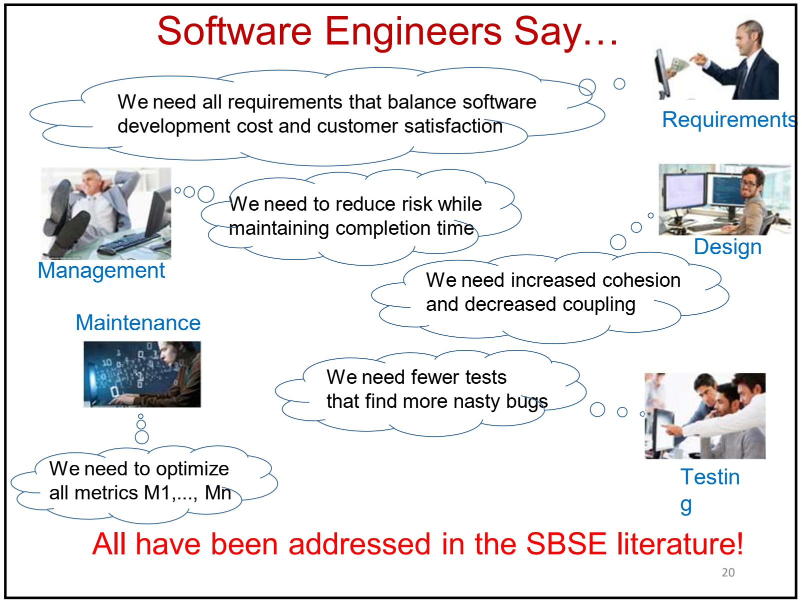
Fitness computed on a representation

Software is eating the world!

Software Engineers …

let’s listen to software engineers ...

... what sort of things do they say?



Software Engineers Say…

We need all requirements that balance software development cost and customer satisfaction

Requirements

We need to reduce risk while maintaining completion time

Management

Design

We need increased cohesion and decreased coupling

Maintenance

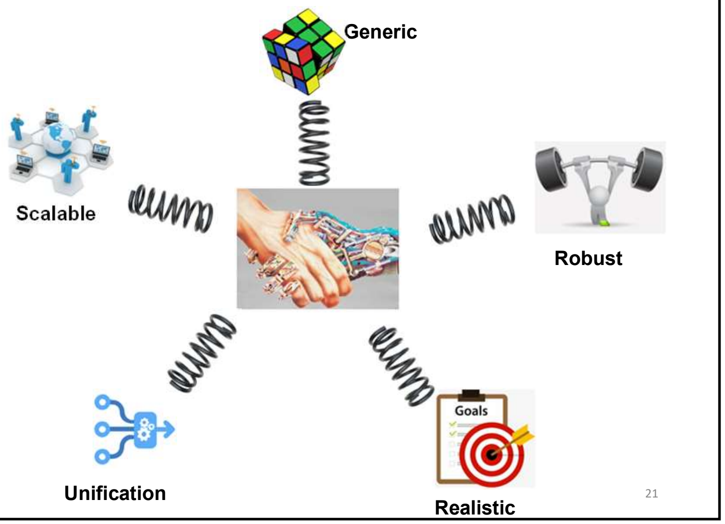
We need fewer tests that find more nasty bugs

We need to optimize all metrics M1,..., Mn

Testin
g

All have been addressed in the SBSE literature!

The Advantages of SBSE



Our Recent Work on SBSE

Search-Based Software Engineering

Automated refactoring recommendation
ASE19, TOSEM16, ASE13, IST16, JSME16, …

Code smells detection
ICWS20, MobileSoft17, TSE15, ASE13, …

Ref… services
…, ICWS16, ICSOC16 …

Social debt in software projects
ICGSE20, …

…eviewers assignment
…CCO20, ICSME16

Software remodularization
ICSOC19, TOSEM14, FSF…

Refactoring detection
EMSE16, …

…oring prioritization
…J14, JSS15, …

Software library reuse
ASOC19, IST17, …

Refactoring prediction
ICSOC16, …

…sion Testing for Refactoring
…16, …

Software Integration
GECCO20, …

Agenda

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent applications
  - SBSE for Performance regression [SSBSE'19]
  - SBSE for Web service design [TSC'17 + ASE'19]
  - SBSE for Modern Code Review [ICSME'16 + GECCO'20]
- A hands-on activity with SBSE
  - MOEA Framework
- Challenges and future work with SBSE

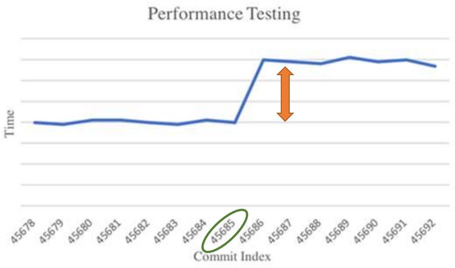
Software Maintenance

- Systems, like people, get old!
  - They increase in complexity and degrade in effectiveness
- Software changes frequently
  - Add new requirements
  - Adapt to environment changes
  - Correct bugs, …

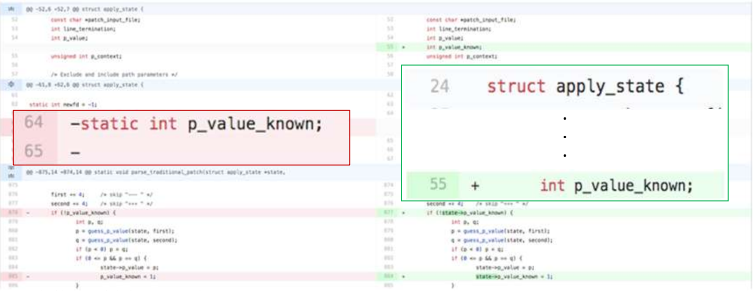
- Challenges
  - These changes may degrade their design and QoS
  - Maintain a high level of quality during the life cycle of a software system

Performance Regression Testing

- Performance regression testing monitors software execution time to avoid degradation during evolution.



Performance Regression Testing



```
64   -static int p_value_known;
65   -
```

```
24        struct apply_state {
                  .
                  .
                  .
55   +        int p_value_known;
```

* This example is from *Git* repository

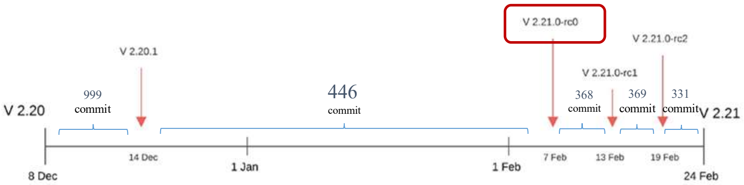
Performance Regression Detection

- Problem: How to find code change introducing performance regression?
- Ideal solution: Test performance of each code change.

Performance Regression Detection

- Problem: How to find code change introducing performance regression?
- Ideal solution: Test performance of each code change.

5 benchmarks ~2 hours ~30 Days for testing



* This example is from *Git* repository

Performance Regression Testing Challenges

- Performance testing is by nature time and resource consuming.
- Growth of committed code.
- Reduction of testing period.
- Bring out the problem of finding which change made the regression.

Software	Avg. Revision per Day	Regular Performance Testing
MySQL	6	every release
Chrome	140	Every 4 revisions
Linux	140	Every week

Table: Estimated commit and performance testing frequency [Huange et al. 2014]

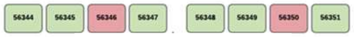
Goal!

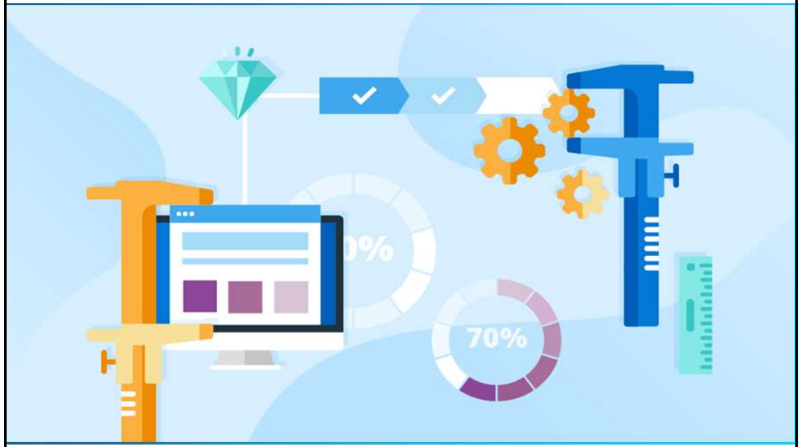
- Apply performance testing only on code change most likely to introduce a performance regression.

How?

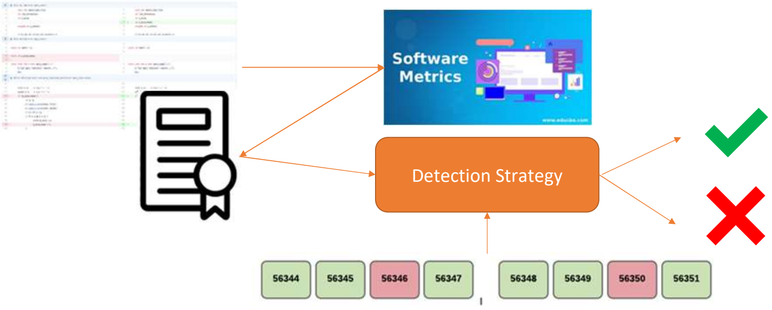
- By **profiling** code changes using data collected from the new code change and compare it with previous code changes data.

Commits Timeline





Profiling a Code Change?



Automated detection of performance regression

PRICE: Detection of Performance Regression Introducing Code Changes Using Static and Dynamic Metrics

Deema Alshoaibi, Kevin Hannigan, Hiten Gupta, and Mohamed Wiem Mkaouer

Rochester Institute of Technology, New York, USA
{da3352,kph1958,hg1938,mwmvse}@rit.edu

Abstract. Performance regression testing is highly expensive as it delays system development when optimally conducted after each code change. Therefore, it is important to prioritize the schedule of performance tests by executing them only when a newly committed change is most likely to introduce performance regression. This paper introduces a novel formulation of the detection of performance regression introducing code changes as an optimization problem. Static and dynamic metrics are combined to generate a detection rule, which is being optimized in terms of its ability to flag problematic code changes, and avoid false positives. We evaluated our approach using performance issues, extracted from the *Git* project. Results show the effectiveness of our approach in accurately detecting performance regression introducing code changes compared with state-of-the-art techniques. Moreover, our suggested detection rules were found to be robust to the software changes over time, which reduces the overhead of updating them frequently.

Keywords: Performance regression, multi-objective optimization, software testing, software quality

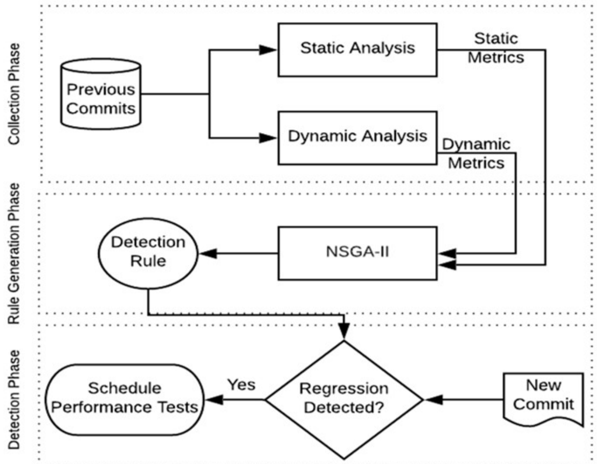
1 Introduction

Performance is critical to software quality. Being one of the practices of quality assurance, performance regression testing monitors the software's overall performance during its evolution to ensure least to negligible degradation of time. It mainly detects whether any committed changes may have introduced performance regressions.

Ideally, in order to prevent any code change from negatively impacting the software performance, performance tests, also known as benchmarks, should be executed along with any committed change, as a sanity check. However, in a real-world setting, performance tests are expensive, and with the growth in the number of committed changes, software testers

International Symposium on Search Based Software Engineering (SSBSE), 2019

Automated detection of performance regression

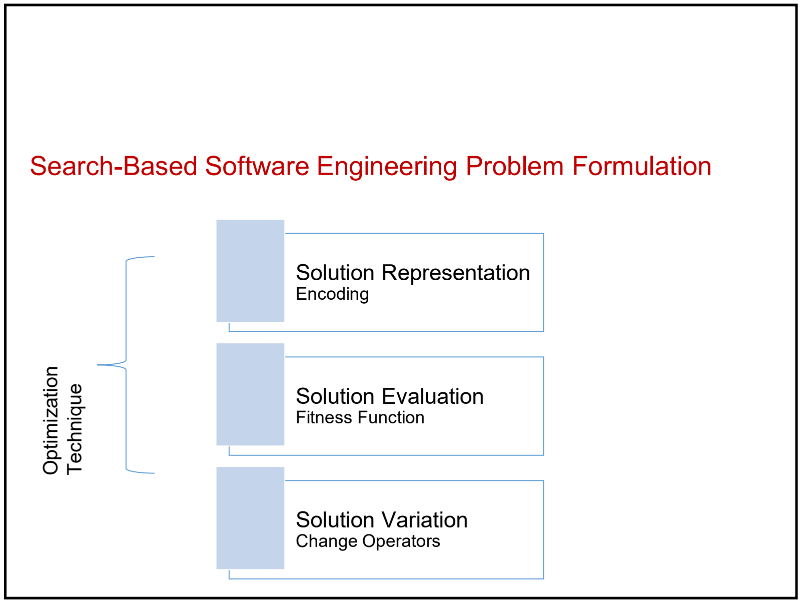


Alshoail
regressi
Softwar

rformance
arch Based

Metrics

Description	Data Source
Number of deleted functions	Static
Number of new functions	Static
Number of deleted Functions reached by the benchmark	Static + Dynamic
The percent overhead of the top most called function that was changed	Static + Dynamic
The percent overhead of the top most called function that was changed by more than 10% of its static instruction length.	Static + Dynamic
The highest percent static function length change	Static
The highest percent static function length change that is called by the benchmark	Static + Dynamic



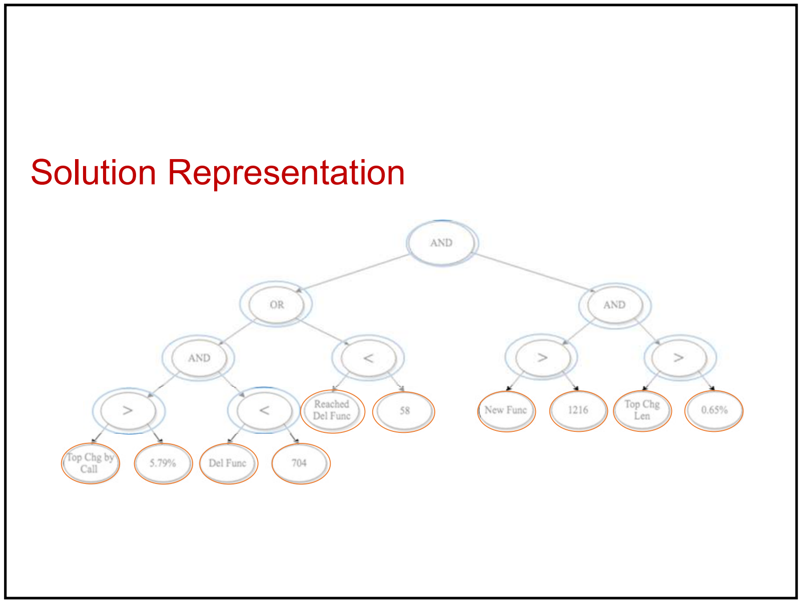
Search-Based Software Engineering Problem Formulation

Optimization Technique

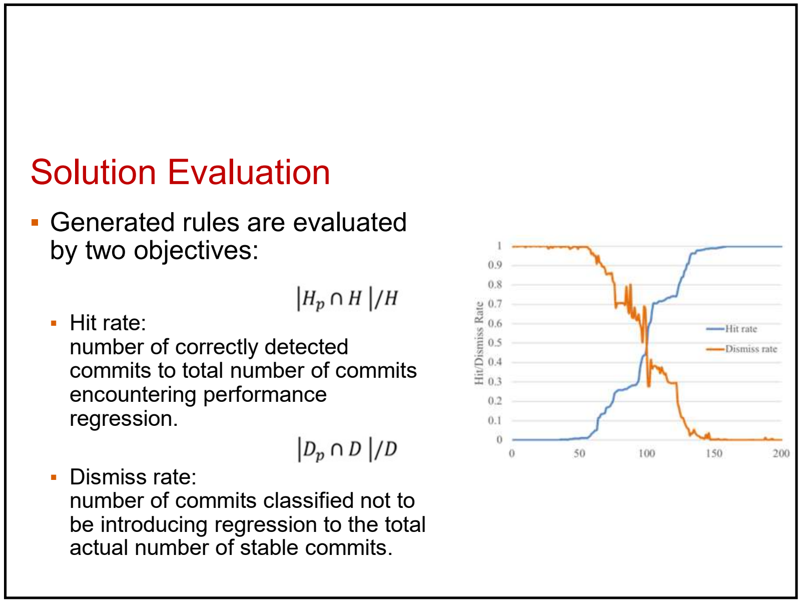
Solution Representation
Encoding

Solution Evaluation
Fitness Function

Solution Variation
Change Operators

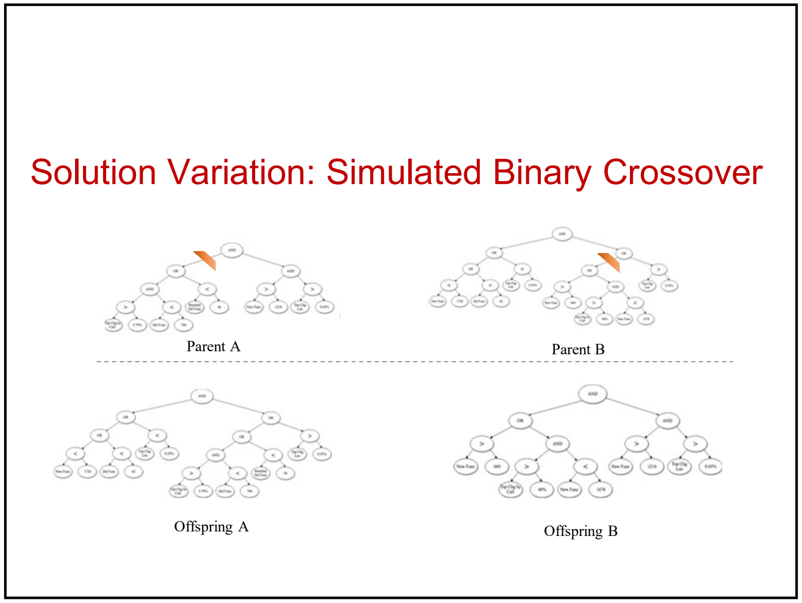


Solution Representation



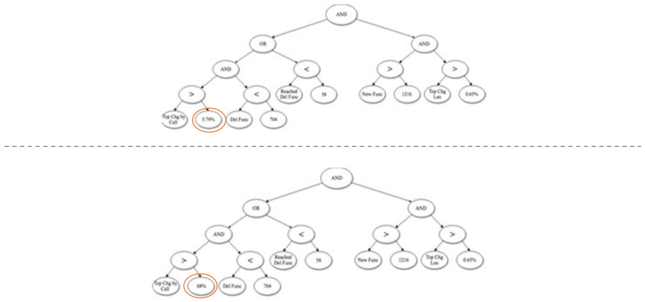
Solution Evaluation

- Generated rules are evaluated by two objectives:
  - Hit rate: $|H_p \cap H|/H$
    number of correctly detected commits to total number of commits encountering performance regression.
  - Dismiss rate: $|D_p \cap D|/D$
    number of commits classified not to be introducing regression to the total actual number of stable commits.

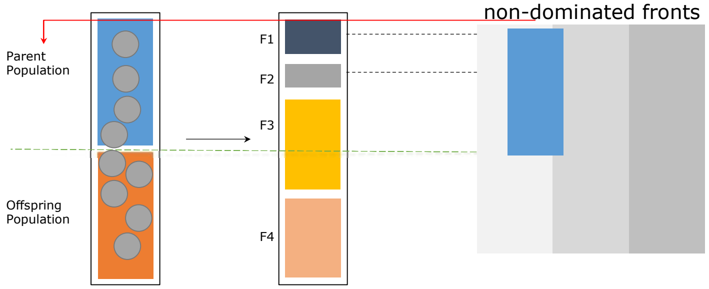


Solution Variation: Simulated Binary Crossover

Solution Variation: Polynomial Mutation



NSGA-II



Experimental Setup

- Used Dataset:

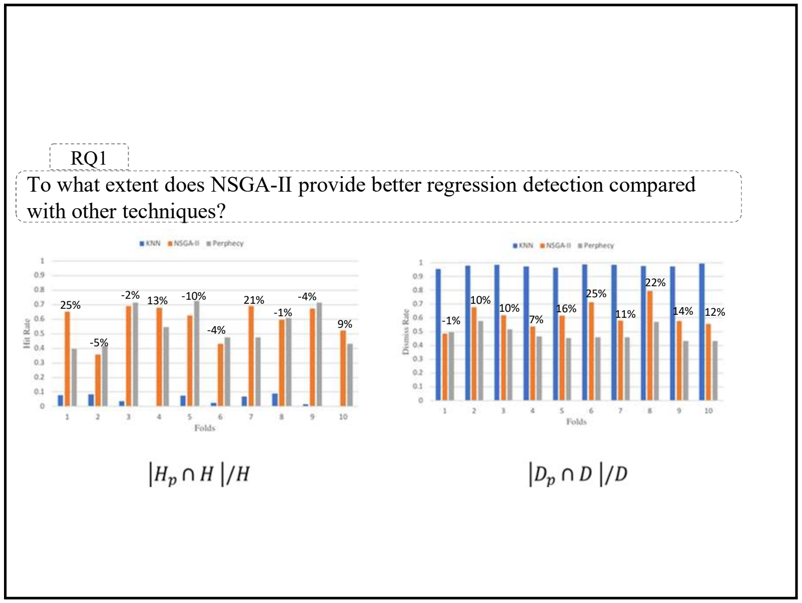
Software	Commits	Benchmarks	Metrics	
			Static	Dynamic
Git	713	5	2	5

- Tuned Parameters:

Population size	Iterations	Simulated binary crossover probability	Polynomial Mutation probability
50	10000	0.8	0.5

Experimental Setup

- **Research questions:**
  - RQ1: To what extent does NSGA-II provide better regression detection compared with other techniques?
    - Compare NSGA-II with a deterministic approach and KNN.
  - RQ2: Does generated rules continue performing well with the evolution of the software?
    - Test performance of rule generated by earlier code changes on latest ones.

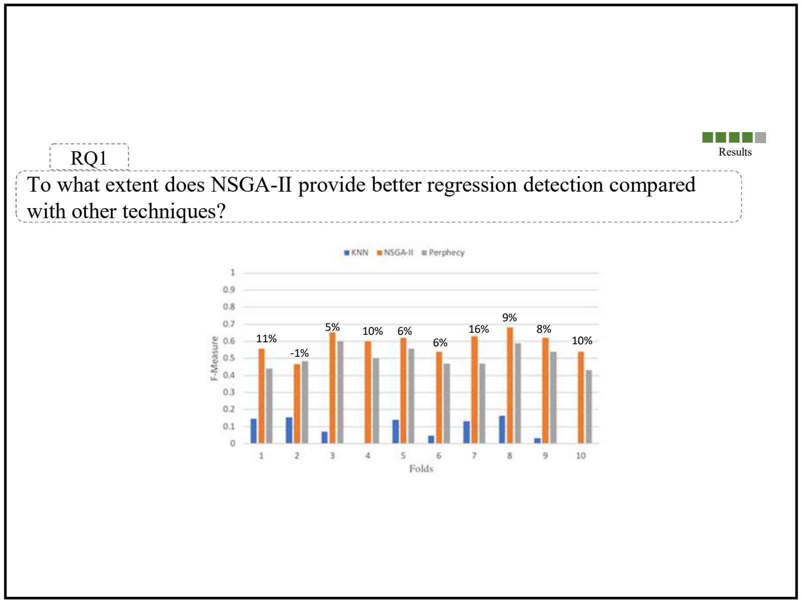


RQ1

To what extent does NSGA-II provide better regression detection compared with other techniques?

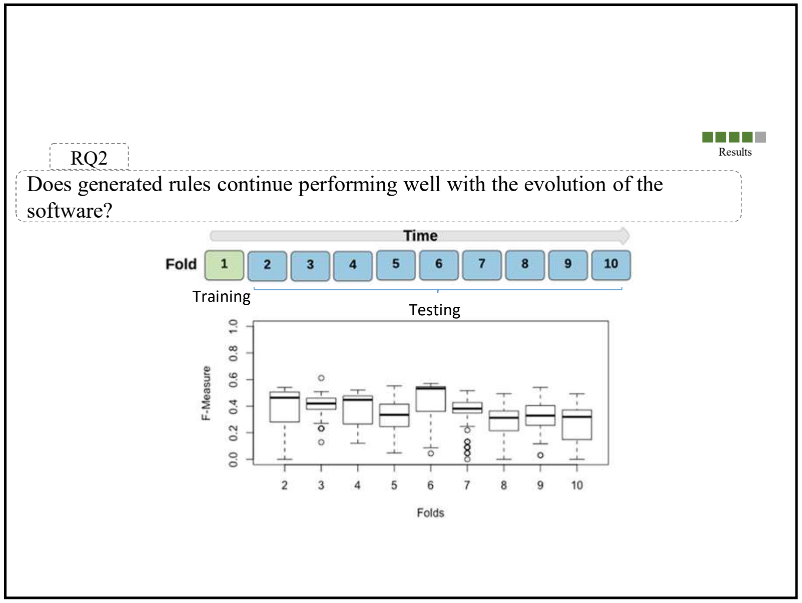
$|H_p \cap H|/H$

$|D_p \cap D|/D$



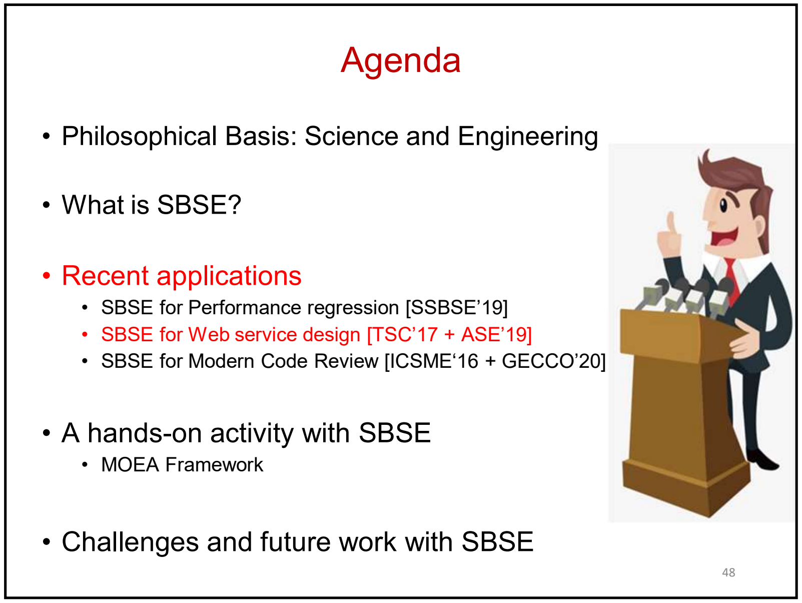
RQ1

To what extent does NSGA-II provide better regression detection compared with other techniques?



RQ2

Does generated rules continue performing well with the evolution of the software?



Agenda

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent applications
  - SBSE for Performance regression [SSBSE’19]
  - SBSE for Web service design [TSC’17 + ASE’19]
  - SBSE for Modern Code Review [ICSME’16 + GECCO’20]
- A hands-on activity with SBSE
  - MOEA Framework
- Challenges and future work with SBSE

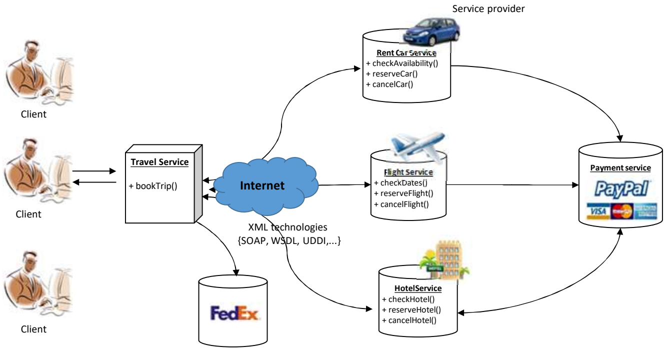
48

What is SOA?

- SOA: Service Oriented Architecture is
  - a way of designing system
  - an approach to system development
  - a design paradigm
- SOA is not an architecture, is not a system
- SOA can be implemented utilizing different technologies
  - OSGi, SCA, REST, **Web services**
- Service-based system = a set of ready-made, composable and reusable services

Service-based system

- Example: Travel system

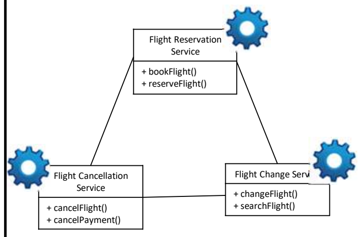


If designed well

- If designed well, Web services reuse can lead to
  - Cost-efficiency
  - interoperability
  - Agility
  - Adaptability
  - Leverage of legacy investments

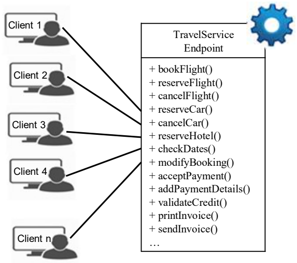
The hard part is the "if designed well".

Web service design antipatterns



Fine-grained Service

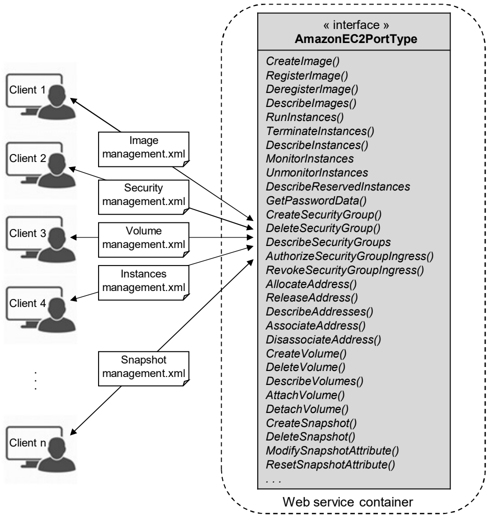
- Few operations
- Low cohesion
- High coupling
- High development complexity
- Reduced usability



God-object Service

- Several operations
- Low cohesion
- High response time
- Low availability
- Not easily reusable

Amazon Elastic Compute Cloud (EC2) Web service



- 87 operation in 1 single interface
- 4,261 lines of WSDL document
- 812 pages of API documentation

Refactor!

State-of-the-art

- Manual approaches/guidelines
  - Service antipatterns definition (Dudney et al., 2003, Král et al, 2009, Rotem-Gal-Oz et al., 2012)

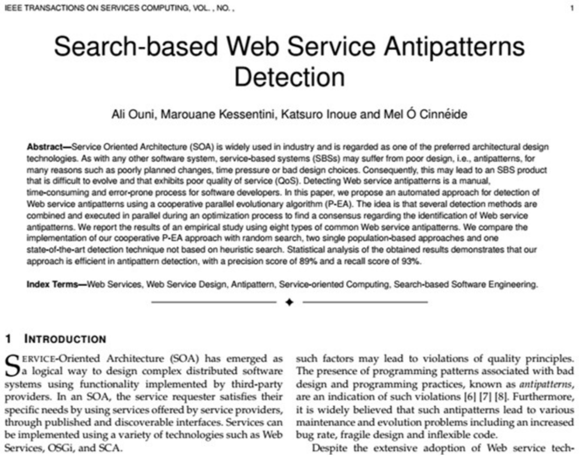
- Symptoms-based approaches
  - Detection rules (Moha et al., 2012, Palma et al., 2014)
    - Translate antipattern symptoms into detection rules
    - Combination metrics/threshold value

Antipatterns detection challenges

- Difficult to define/express detection rules
  - Large list of antipattern types to categorize
  - Large exhaustive list of quality metrics
  - Large number of possible threshold values
  - Huge space to explore: An expert to manually write and validate detection rules
  - No consensual definitions of antipatterns

➡ Idea: Infer detection rules from antipattern examples using combinatorial optimization

Web service anti-patterns detection



IEEE TRANSACTIONS ON SERVICES COMPUTING, VOL. , NO. , 1

Search-based Web Service Antipatterns Detection

Ali Ouni, Marouane Kessentini, Katsuro Inoue and Mel Ó Cinnéide

Abstract—Service Oriented Architecture (SOA) is widely used in industry and is regarded as one of the preferred architectural design technologies. As with any other software system, service-based systems (SBSs) may suffer from poor design, i.e., antipatterns, for many reasons such as poorly planned changes, time pressure or bad design choices. Consequently, this may lead to an SBS product that is difficult to evolve and that exhibits poor quality of service (QoS). Detecting Web service antipatterns is a manual, time-consuming and error-prone process for software developers. In this paper, we propose an automated approach for detection of Web service antipatterns using a cooperative parallel evolutionary algorithm (P-EA). The idea is that several detection methods are combined and executed in parallel during an optimization process to find a consensus regarding the identification of Web service antipatterns. We report the results of an empirical study using eight types of common Web service antipatterns. We compare the implementation of our cooperative P-EA approach with random search, two single population-based approaches and one state-of-the-art detection technique not based on heuristic search. Statistical analysis of the obtained results demonstrates that our approach is efficient in antipattern detection, with a precision score of 89% and a recall score of 93%.

Index Terms—Web Services, Web Service Design, Antipattern, Service-oriented Computing, Search-based Software Engineering.

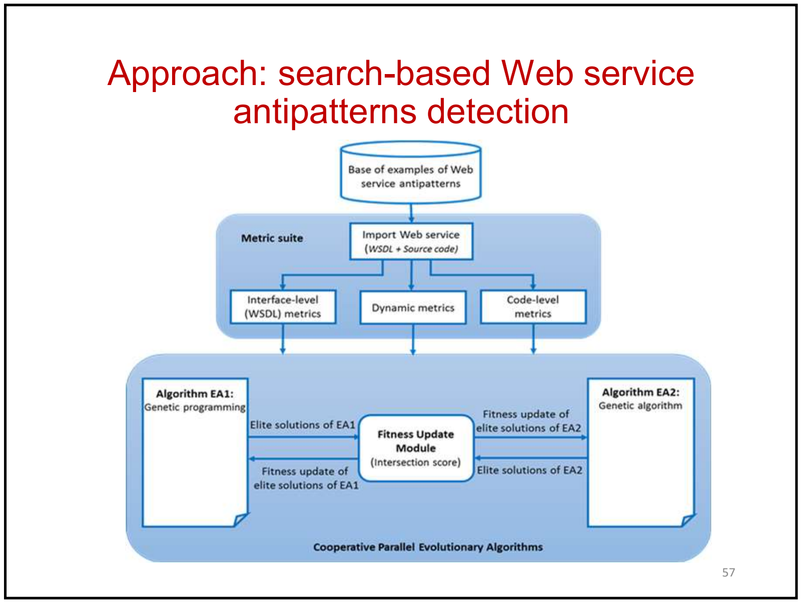
1 Introduction

Service-Oriented Architecture (SOA) has emerged as a logical way to design complex distributed software systems using functionality implemented by third-party providers. In an SOA, the service requester satisfies their specific needs by using services offered by service providers, through published and discoverable interfaces. Services can be implemented using a variety of technologies such as Web Services, OSGi, and SCA.

such factors may lead to violations of quality principles. The presence of programming patterns associated with bad design and programming practices, known as *antipatterns*, are an indication of such violations [6] [7] [8]. Furthermore, it is widely believed that such antipatterns lead to various maintenance and evolution problems including an increased bug rate, fragile design and inflexible code.

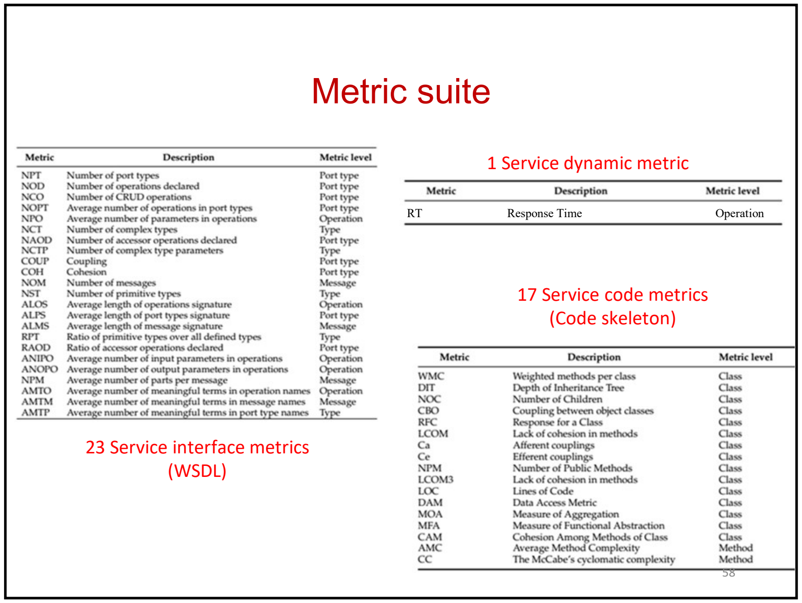
Despite the extensive adoption of Web service tech-

IEEE Transactions on Services Computing (TSC), 2017.



Approach: search-based Web service antipatterns detection

Base of examples of Web service antipatterns
Metric suite
Import Web service (WSDL + Source code)
Interface-level (WSDL) metrics
Dynamic metrics
Code-level metrics
Algorithm EA1: Genetic programming
Elite solutions of EA1
Fitness update of elite solutions of EA1
Fitness Update Module (Intersection score)
Fitness update of elite solutions of EA2
Elite solutions of EA2
Algorithm EA2: Genetic algorithm
Cooperative Parallel Evolutionary Algorithms



Metric suite

Metric	Description	Metric level
NPT	Number of port types	Port type
NOD	Number of operations declared	Port type
NCO	Number of CRUD operations	Port type
NOPT	Average number of operations in port types	Port type
NPO	Average number of parameters in operations	Operation
NCT	Number of complex types	Type
NAOD	Number of accessor operations declared	Port type
NCTP	Number of complex type parameters	Type
COUP	Coupling	Port type
COH	Cohesion	Port type
NOM	Number of messages	Message
NST	Number of primitive types	Type
ALOS	Average length of operations signature	Operation
ALPS	Average length of port types signature	Port type
ALMS	Average length of message signature	Message
RPT	Ratio of primitive types over all defined types	Type
RAOD	Ratio of accessor operations declared	Port type
ANIPO	Average number of input parameters in operations	Operation
ANOPO	Average number of output parameters in operations	Operation
NPM	Average number of parts per message	Message
AMTO	Average number of meaningful terms in operation names	Operation
AMTM	Average number of meaningful terms in message names	Message
AMTP	Average number of meaningful terms in port type names	Type

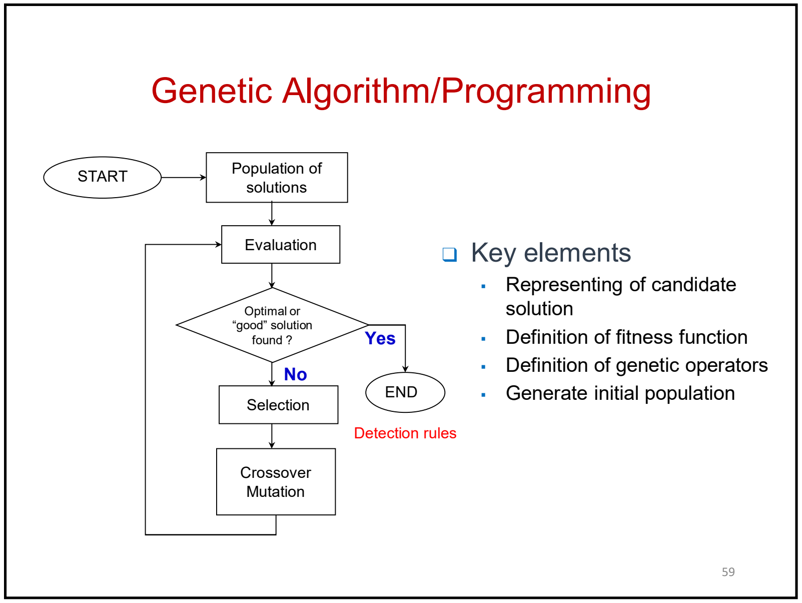
23 Service interface metrics (WSDL)

1 Service dynamic metric

Metric	Description	Metric level
RT	Response Time	Operation

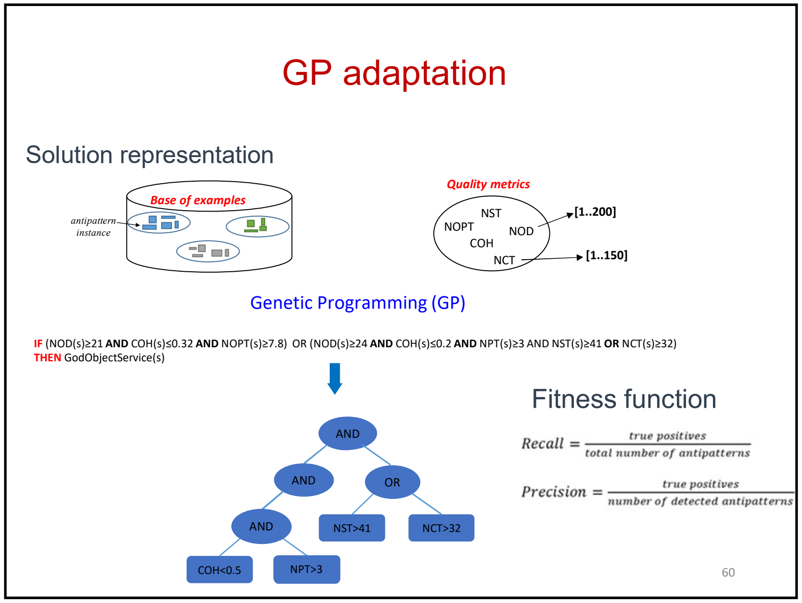
17 Service code metrics (Code skeleton)

Metric	Description	Metric level
WMC	Weighted methods per class	Class
DIT	Depth of Inheritance Tree	Class
NOC	Number of Children	Class
CBO	Coupling between object classes	Class
RFC	Response for a Class	Class
LCOM	Lack of cohesion in methods	Class
Ca	Afferent couplings	Class
Ce	Efferent couplings	Class
NPM	Number of Public Methods	Class
LCOM3	Lack of cohesion in methods	Class
LOC	Lines of Code	Class
DAM	Data Access Metric	Class
MOA	Measure of Aggregation	Class
MFA	Measure of Functional Abstraction	Class
CAM	Cohesion Among Methods of Class	Class
AMC	Average Method Complexity	Method
CC	The McCabe's cyclomatic complexity	Method



Genetic Algorithm/Programming

- ❑ Key elements
  - Representing of candidate solution
  - Definition of fitness function
  - Definition of genetic operators
  - Generate initial population



GP adaptation

Solution representation

Genetic Programming (GP)

IF (NOD(s)≥21 **AND** COH(s)≤0.32 **AND** NOPT(s)≥7.8) OR (NOD(s)≥24 **AND** COH(s)≤0.2 **AND** NPT(s)≥3 AND NST(s)≥41 **OR** NCT(s)≥32)
THEN GodObjectService(s)

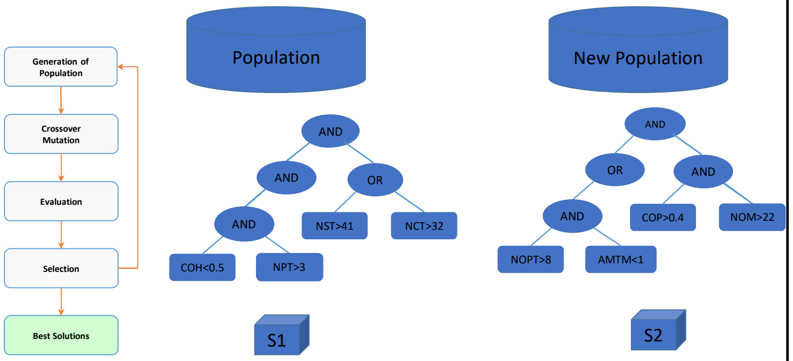
Fitness function

$$Recall = \frac{true\ positives}{total\ number\ of\ antipatterns}$$

$$Precision = \frac{true\ positives}{number\ of\ detected\ antipatterns}$$

GP adaptation

Change operators



Evaluation : research questions

- **RQ1.** How does our P-EA approach compare to GP, GA and random search (RS)?
- **RQ2.** To what extent can the proposed approach efficiently detect Web service antipatterns?
- **RQ3.** What types of Web service antipatterns does it detect correctly?
- **RQ4.** How does P-EA perform compared to existing Web service antipattern detection approaches?

Evaluation

- Studied Web services
  - Benchmark of 425 Web services

Category	# services	# antipatterns	average NOD	average NOM	average NCT
Financial	94	67	29.52	57.31	19.01
Science	34	3	8.47	17.14	96.73
Search	37	13	8.35	18.94	26.13
Shipping	38	10	13.36	27.76	20.21
Travel	65	28	16.09	33.13	121.13
Weather	42	15	8.54	17.16	9.14
Media	19	14	10.9	16.4	28.6
Education	26	15	11.3	15.36	32.46
Messaging	29	20	7.6	11.21	18.25
Location	31	22	5.8	28.32	11.15
All	425	136	17.08	34.2	48.6

  - Dataset: https://github.com/ouniali/WSantipatterns

Evaluation

- Eight common types of antipatterns
  - *God object Web service (GOWS)*
  - *Fine grained Web service (FGWS)*
  - *Chatty Web service (CWS)*
  - *Data Web service (DWS)*
  - *Ambiguous Web service (AWS)*
  - *Redundant PortTypes (RPT)*
  - *CRUDy Interface (CI)*
  - *Maybe It is Not RPC (MNR)*
- 10-fold cross validation
  - Detect antpatterns in one category using the 9 other categories

Evaluation metrics

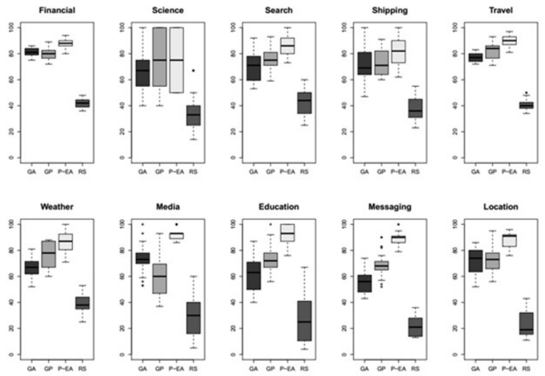
- Detection precision and recall rates

$$Recall = \frac{true\ positives}{total\ number\ of\ antipatterns}$$

$$Precision = \frac{true\ positives}{number\ of\ detected\ antipatterns}$$

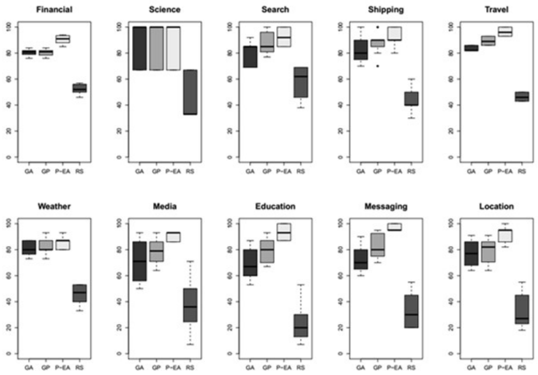
- State of the art comparison
  - SODA-W (Palma et al. 2014)
  - Ouni et al. 2015

RQ1: Comparison of P-EA with GP, GA, and RS



Detection Precision

RQ1: Comparison of P-EA with GP, GA, and RS

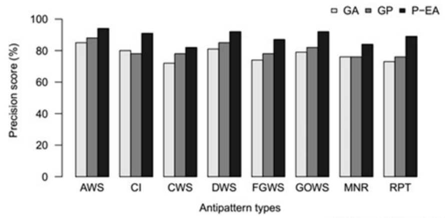


Detection Recall

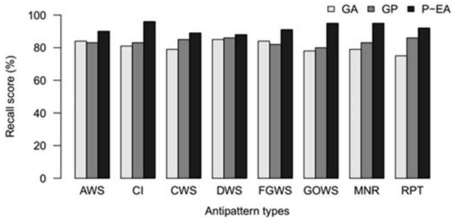
RQ2: 89% precision and 93% recall

Category	Precision (%)	Recall (%)
Financial	88	91
Science	87	92
Search	86	92
Shipping	82	90
Travel	90	96
Weather	87	91
Media	93	93
Education	93	93
Messaging	90	95
Location	91	95
Average	**89**	**93**

RQ3: sensitivity towards antipattern types

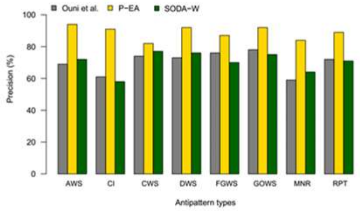


Detection Precision

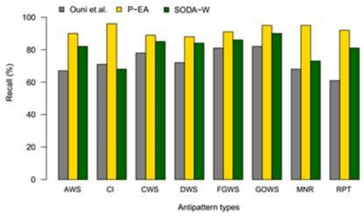


Detection Recall

RQ4: P-EA outperforms SOAD-W and Ouni et al. 2015



Detection Precision



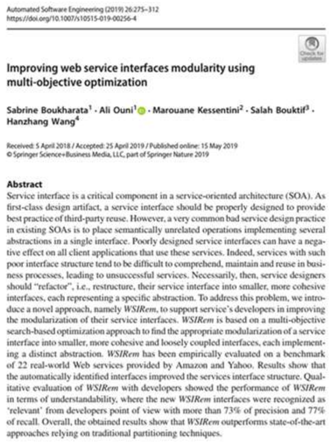
Detection Recall

Well… here are my detected antipatterns! and then … ?

Refactor them!

How to fix these smells?

Automated Software Engineering Journal 2019



Automated Software Engineering (2019) 26:275–312
https://doi.org/10.1007/s10515-019-00256-4

Check for updates

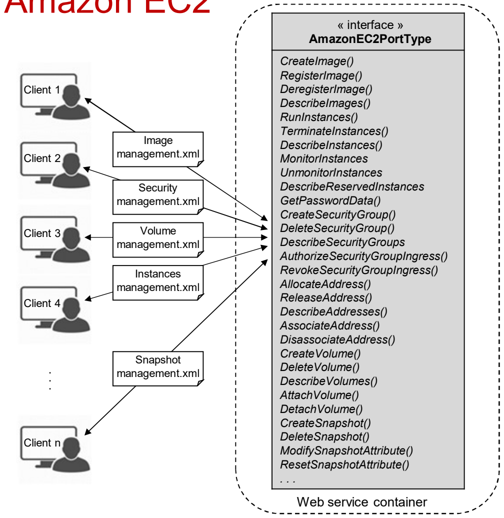
Improving web service interfaces modularity using multi-objective optimization

Sabrine Boukharata[1] · Ali Ouni[1] · Marouane Kessentini[2] · Salah Bouktif[3] · Hanzhang Wang[4]

Received: 5 April 2018 / Accepted: 25 April 2019 / Published online: 15 May 2019

Abstract
Service interface is a critical component in a service-oriented architecture (SOA). As first-class design artifact, a service interface should be properly designed to provide best practice of third-party reuse. However, a very common bad service design practice in existing SOAs is to place semantically unrelated operations implementing several abstractions in a single interface. Poorly designed service interfaces can have a negative effect on all client applications that use these services. Indeed, services with such poor interface structure tend to be difficult to comprehend, maintain and reuse in business processes, leading to unsuccessful services. Necessarily, then, service designers should "refactor", i.e., restructure, their service interface into smaller, more cohesive interfaces, each representing a specific abstraction. To address this problem, we introduce a novel approach, namely *WSIRem*, to support service's developers in improving the modularization of their service interfaces. *WSIRem* is based on a multi-objective search-based optimization approach to find the appropriate modularization of a service interface into smaller, more cohesive and loosely coupled interfaces, each implementing a distinct abstraction. *WSIRem* has been empirically evaluated on a benchmark of 22 real-world Web services provided by Amazon and Yahoo. Results show that the automatically identified interfaces improved the services interface structure. Qualitative evaluation of *WSIRem* with developers showed the performance of *WSIRem* in terms of understandability, where the new *WSIRem* interfaces were recognized as 'relevant' from developers point of view with more than 73% of precision and 77% of recall. Overall, the obtained results show that *WSIRem* outperforms state-of-the-art approaches relying on traditional partitioning techniques.

Amazon EC2

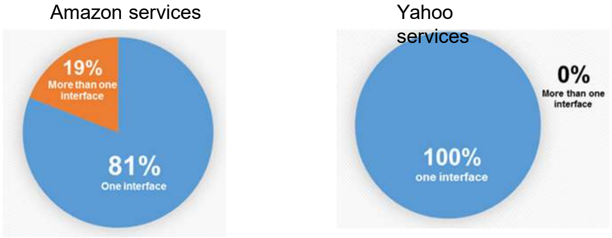


- 87 operations in one single interface
- 4,261 lines of WSDL document
- 812 pages of API documentation

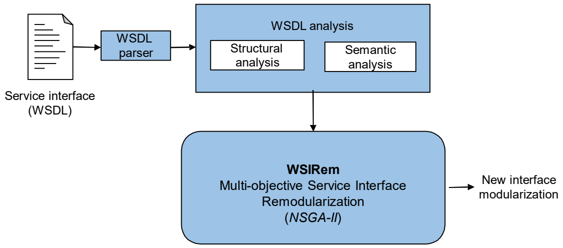
Re-design!

Service interface design

- Amazon: 81% of services provides only one interface
- Yahoo: 100% of services provides only one interface



WSIRem : multi-objective search to improve service interfaces modularity

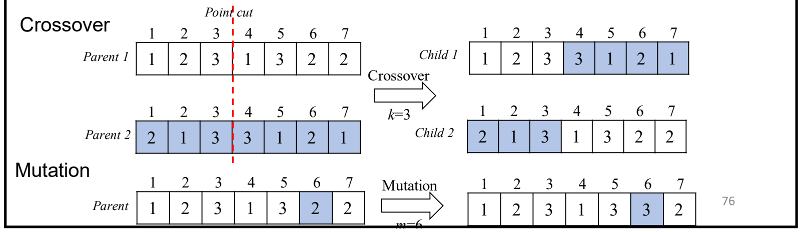


WSIRem : multi-objective search to improve service interfaces modularity

- Search algorithm : NSGA-II
- Solution representation

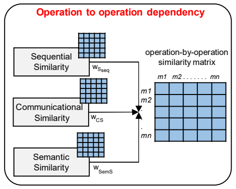
op_	1	2	3	4	5	6	7
si_	1	2	3	1	3	2	2

- Objective functions
  1. Maximize cohesion
  2. Minimize coupling
  3. Minimize the interfaces modifications

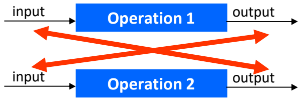


	1	2	3	4	5	6	7
Parent 1	1	2	3	1	3	2	2
Parent 2	2	1	3	3	1	2	1
Child 1	1	2	3	3	1	2	1
Child 2	2	1	3	1	3	2	2
Parent	1	2	3	1	3	2	2
	1	2	3	1	3	3	2

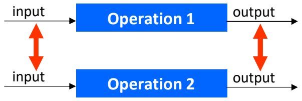
Operations Cohesion


Operation to operation dependency
Sequential Similarity
Communicational Similarity
Semantic Similarity
wSseq
wCS
wSemS
operation-by-operation similarity matrix
m1 m2 . . . . mn
m1
m2
mn

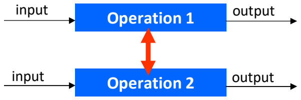
- Sequential cohesion



- Communicational cohesion



- Semantic cohesion

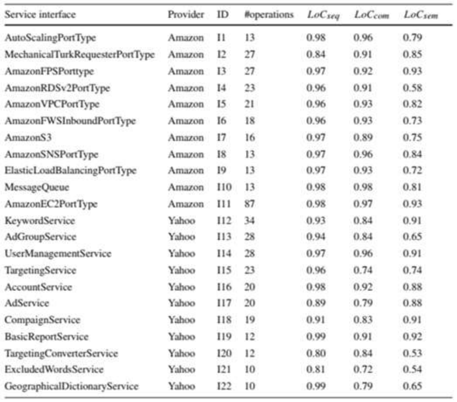


Evaluation

- Research questions
  - **RQ1.** What is the impact of the suggested remodularizations by our approach on service interface design quality?
  - **RQ2.** Do the suggested remodularizations provide a better partitioning of abstractions from a developer's point of view?

Evaluation

- 22 Web services



Service interface	Provider	ID	#operations	LoC_{seq}	LoC_{com}	LoC_{sem}
AutoScalingPortType	Amazon	I1	13	0.98	0.96	0.79
MechanicalTurkRequesterPortType	Amazon	I2	27	0.84	0.91	0.85
AmazonFPSPorttype	Amazon	I3	27	0.97	0.92	0.93
AmazonRDSv2PortType	Amazon	I4	23	0.96	0.91	0.58
AmazonVPCPortType	Amazon	I5	21	0.96	0.93	0.82
AmazonFWSInboundPortType	Amazon	I6	18	0.96	0.93	0.73
AmazonS3	Amazon	I7	16	0.97	0.89	0.75
AmazonSNSPortType	Amazon	I8	13	0.97	0.96	0.84
ElasticLoadBalancingPortType	Amazon	I9	13	0.97	0.93	0.72
MessageQueue	Amazon	I10	13	0.98	0.98	0.81
AmazonEC2PortType	Amazon	I11	87	0.98	0.97	0.93
KeywordService	Yahoo	I12	34	0.93	0.84	0.91
AdGroupService	Yahoo	I13	28	0.94	0.84	0.65
UserManagementService	Yahoo	I14	28	0.97	0.96	0.91
TargetingService	Yahoo	I15	23	0.96	0.74	0.74
AccountService	Yahoo	I16	20	0.98	0.92	0.88
AdService	Yahoo	I17	20	0.89	0.79	0.88
CompaignService	Yahoo	I18	19	0.91	0.83	0.91
BasicReportService	Yahoo	I19	12	0.99	0.91	0.92
TargetingConverterService	Yahoo	I20	12	0.80	0.84	0.53
ExcludedWordsService	Yahoo	I21	10	0.81	0.72	0.54
GeographicalDictionaryService	Yahoo	I22	10	0.99	0.79	0.65

- https://github.com/ouniali/AmazonYahooBenchmark

Evaluation Method

- Quantitative measures
  - Cohesion
  - Coupling
  - Modularity
- Qualitative measures
  - Precision
  - Recall

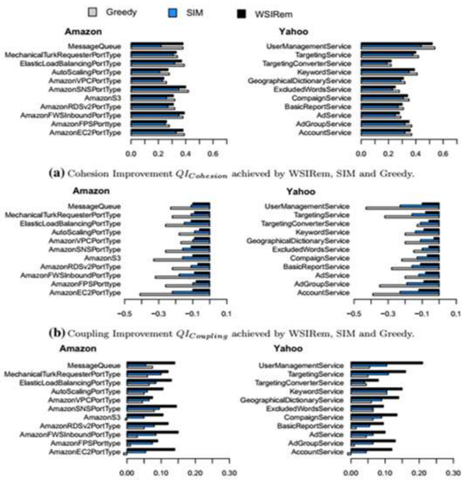
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

- Baseline approaches
  - SIM (Ouni et al. 2016) : graph-based partitioning
  - Greedy algorithm (Athanasopoulos et al. 2015)

Quantitative evaluation results

-



(a) Cohesion Improvement $QI_{Cohesion}$ achieved by WSIRem, SIM and Greedy.

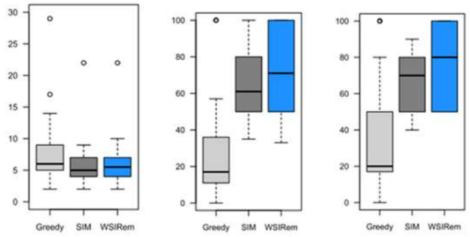
(b) Coupling Improvement $QI_{Coupling}$ achieved by WSIRem, SIM and Greedy.

(c) Modularity Improvement $QI_{Modularity}$ achieved by WSIRem, SIM and Greedy.

Qualititative evaluation results

Provider	Interface	WSIRem			Greedy			SIM		
		# interfaces	Precision (%)	Recall (%)	# interfaces	Precision (%)	Recall (%)	# interfaces	Precision (%)	Recall (%)
Amazon	AmazonEC2PortType	22	82	90	29	14	18	22	82	90
	MechanicalTurkRequesterPortType	7	100	100	17	0	0	7	80	85
	AmazonFPSPorttype	10	80	89	11	27	30	9	60	70
	AmazonRDSv2PortType	6	67	80	5	20	20	5	66	70
	AmazonVPCPortType	6	100	100	6	0	0	6	85	80
	AmazonFWSInboundPortType	6	67	67	5	40	33	6	60	58
	AmazonS3	4	75	60	5	17	20	5	65	50
	AmazonSNSPortType	5	40	50	6	17	20	4	35	40
	ElasticLoadBalancingPortType	4	50	67	4	0	0	3	40	55
	MessageQueue	4	50	50	6	50	60	4	45	50
	AutoScalingPortType	4	50	67	6	17	33	3	45	55
Yahoo	KeywordService	8	78	88	9	11	14	7	68	70
	AdGroupService	9	100	100	9	100	80	9	100	100
	UserManagementService	7	100	100	14	36	71	7	80	90
	TargetingService	6	67	80	8	13	20	5	62	70
	AccountService	6	100	100	14	7	17	6	90	90
	AdService	5	60	50	3	25	17	6	55	45
	CompaignService	3	67	50	7	14	25	4	60	40
	BasicReportService	5	100	100	7	57	80	5	80	85
	TargetingConverterService	2	100	100	2	50	60	2	100	100
	ExcludedWordsService	3	33	50	4	33	50	2	50	70
	GeographicalDictionaryService	3	33	50	4	0	0	2	50	50
Average		6.14	72.68	76.73	8.23	27.18	33.09	5.86	64.00	66.05

Qualititative evaluation results

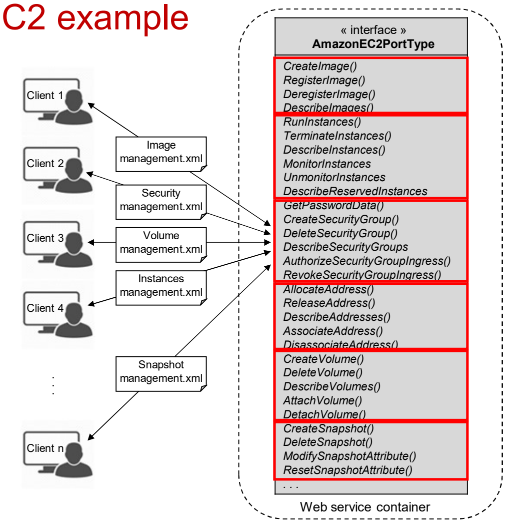


(a) # of interfaces.

(b) Precision.

(c) Recall.

The EC2 example



« interface »
AmazonEC2PortType

CreateImage()
RegisterImage()
DeregisterImage()
DescribeImages()

RunInstances()
TerminateInstances()
DescribeInstances()
MonitorInstances
UnmonitorInstances
DescribeReservedInstances

GetPasswordData()
CreateSecurityGroup()
DeleteSecurityGroup()
DescribeSecurityGroups
AuthorizeSecurityGroupIngress()
RevokeSecurityGroupIngress()

AllocateAddress()
ReleaseAddress()
DescribeAddresses()
AssociateAddress()
DisassociateAddress()

CreateVolume()
DeleteVolume()
DescribeVolumes()
AttachVolume()
DetachVolume()

CreateSnapshot()
DeleteSnapshot()
ModifySnapshotAttribute()
ResetSnapshotAttribute()

. . .

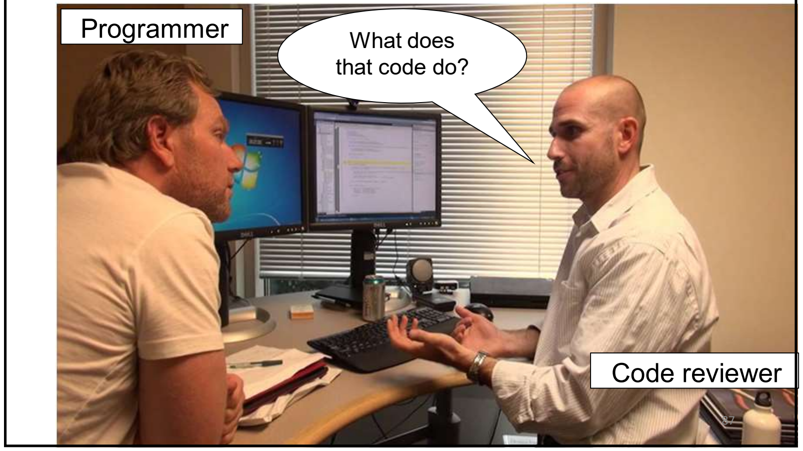
Web service container

Well… here are my refacroring changes! and then … Who should review/approve them?

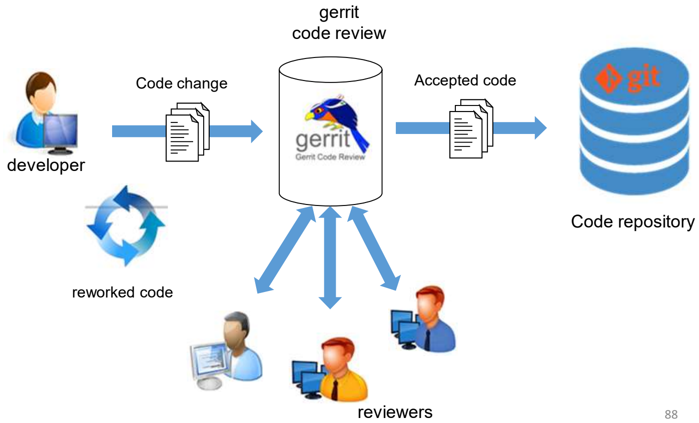
Agenda

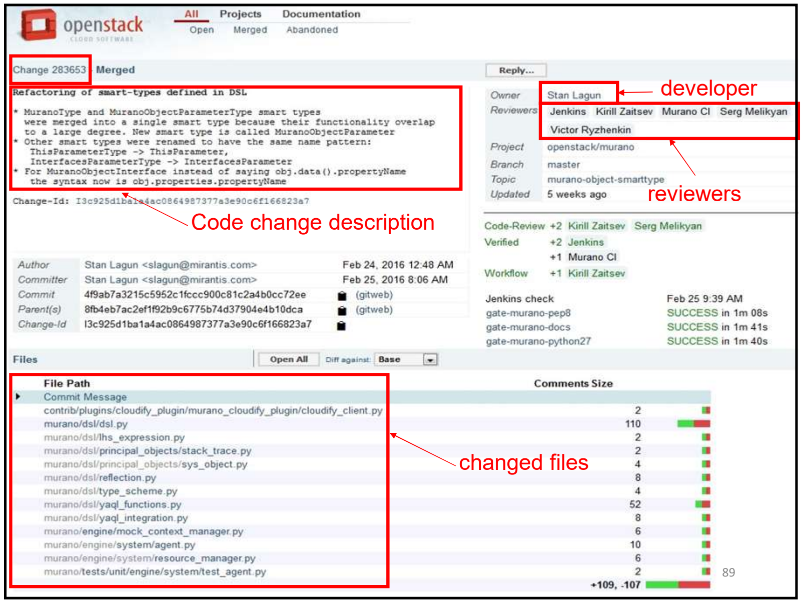
- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent applications
  - SBSE for Performance regression [SSBSE'19]
  - SBSE for Web service design [TSC'17 + ASE'19]
  - SBSE for Modern Code Review [ICSME'16 + GECCO'20]
- A hands-on activity with SBSE
  - MOEA Framework
- Challenges and future work with SBSE

Code review is a key part of the software development process



The "modern", lightweight, tool-supported code review





Change 283653 Merged

```
Refactoring of smart-types defined in DSL

* MuranoType and MuranoObjectParameterType smart types
  were merged into a single smart type because their functionality overlap
  to a large degree. New smart type is called MuranoObjectParameter
* Other smart types were renamed to have the same name pattern:
   ThisParameterType -> ThisParameter,
   InterfacesParameterType -> InterfacesParameter
* For MuranoObjectInterface instead of saying obj.data().propertyName
   the syntax now is obj.properties.propertyName

Change-Id: I3c925d1ba1a4ac0864987377a3e90c6f166823a7
```

Code change description

Author	Stan Lagun <slagun@mirantis.com>	Feb 24, 2016 12:48 AM
Committer	Stan Lagun <slagun@mirantis.com>	Feb 25, 2016 8:06 AM
Commit	4f9ab7a3215c5952c1fccc900c81c2a4b0cc72ee	(gitweb)
Parent(s)	8fb4eb7ac2ef1f92b9c6775b74d37904e4b10dca	(gitweb)
Change-Id	I3c925d1ba1a4ac0864987377a3e90c6f166823a7	

Owner	Stan Lagun
Reviewers	Jenkins Kirill Zaitsev Murano CI Serg Melikyan Victor Ryzhenkin
Project	openstack/murano
Branch	master
Topic	murano-object-smarttype
Updated	5 weeks ago

developer

reviewers

Code-Review	+2	Kirill Zaitsev Serg Melikyan
Verified	+2	Jenkins
	+1	Murano CI
Workflow	+1	Kirill Zaitsev

Jenkins check	Feb 25 9:39 AM
gate-murano-pep8	SUCCESS in 1m 08s
gate-murano-docs	SUCCESS in 1m 41s
gate-murano-python27	SUCCESS in 1m 40s

Files

File Path	Comments	Size
Commit Message		
contrib/plugins/cloudify_plugin/murano_cloudify_plugin/cloudify_client.py		2
murano/dsl/dsl.py		110
murano/dsl/lhs_expression.py		2
murano/dsl/principal_objects/stack_trace.py		2
murano/dsl/principal_objects/sys_object.py		4
murano/dsl/reflection.py		8
murano/dsl/type_scheme.py		4
murano/dsl/yaql_functions.py		52
murano/dsl/yaql_integration.py		8
murano/engine/mock_context_manager.py		6
murano/engine/system/agent.py		10
murano/engine/system/resource_manager.py		6
murano/tests/unit/engine/system/test_agent.py		2
		+109, -107

changed files

89

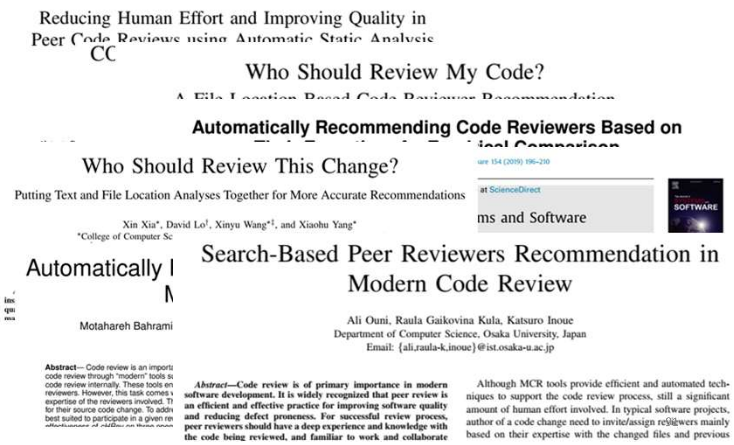
Code reviewers assignment problem

"Who should review my code?"

- Identifying appropriate reviewers is a hard task
  - A code change involve multiple files
  - A file is edited by multiple developers and reviewed my multiple reviewers
- Reviewer assignment problem [Patanamon et al., 2015]
  - delays acceptance : 12 days
  - sometimes patches are completely forgotten

90

State of the art



Reducing Human Effort and Improving Quality in Peer Code Reviews using Automatic Static Analysis

Who Should Review My Code?

Automatically Recommending Code Reviewers Based on

Who Should Review This Change?

Putting Text and File Location Analyses Together for More Accurate Recommendations

Xin Xia*, David Lo†, Xinyu Wang*‡, and Xiaohu Yang*
*College of Computer Sc

ure 154 (2019) 196–210

at ScienceDirect

ms and Software

SOFTWARE

Automatically

Motahareh Bahrami

Abstract— Code review is an import code review through "modern" tools s code review internally. These tools en reviewers. However, this task comes expertise of the reviewers involved. T for their source code change. To addr best suited to participate in a given re

Search-Based Peer Reviewers Recommendation in Modern Code Review

Ali Ouni, Raula Gaikovina Kula, Katsuro Inoue
Department of Computer Science, Osaka University, Japan
Email: {ali,raula-k,inoue}@ist.osaka-u.ac.jp

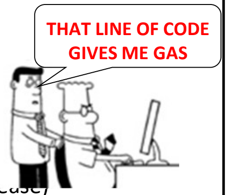
Abstract—Code review is of primary importance in modern software development. It is widely recognized that peer review is an efficient and effective practice for improving software quality and reducing defect proneness. For successful review process, peer reviewers should have a deep experience and knowledge with the code being reviewed, and familiar to work and collaborate

Although MCR tools provide efficient and automated techniques to support the code review process, still a significant amount of human effort involved. In typical software projects, author of a code change need to invite/assign reviewers mainly based on their expertise with the changed files and previous

91

Problem Statement

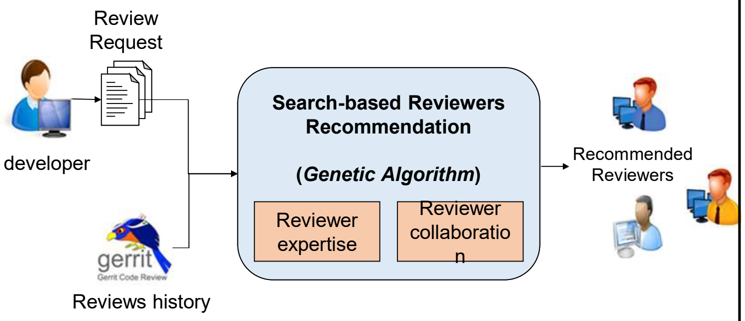
- Single/independent reviewers
  - A change might require many reviewers
- Focus only on reviewer expertise
  - Expertise change over time (increase or decrease)
- No consideration of the socio-technical aspect
  - "Peer" code review
  - human process = personal + social aspects



Heuristic search to optimize both expertise and social aspects

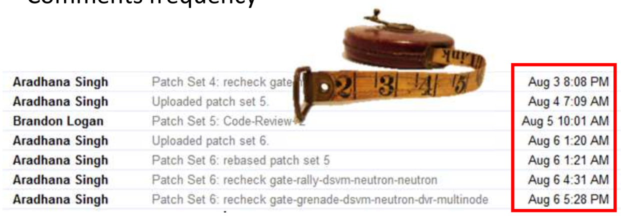
92

Approach overview: RevRec



Reviewer Expertise (RE) model

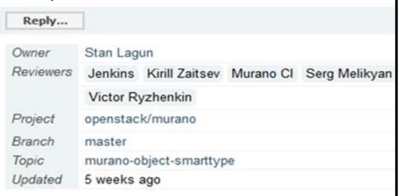
- For each modified file we consider
  - Comments frequency



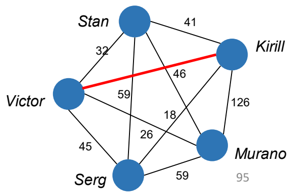
Aradhana Singh	Patch Set 4: recheck gate	Aug 3 8:08 PM
Aradhana Singh	Uploaded patch set 5.	Aug 4 7:09 AM
Brandon Logan	Patch Set 5: Code-Review	Aug 5 10:01 AM
Aradhana Singh	Uploaded patch set 6.	Aug 6 1:20 AM
Aradhana Singh	Patch Set 6: rebased patch set 5	Aug 6 1:21 AM
Aradhana Singh	Patch Set 6: recheck gate-rally-dsvm-neutron-neutron	Aug 6 4:31 AM
Aradhana Singh	Patch Set 6: recheck gate-grenade-dsvm-neutron-dvr-multinode	Aug 6 5:28 PM

Reviewer Collaboration (RC) model

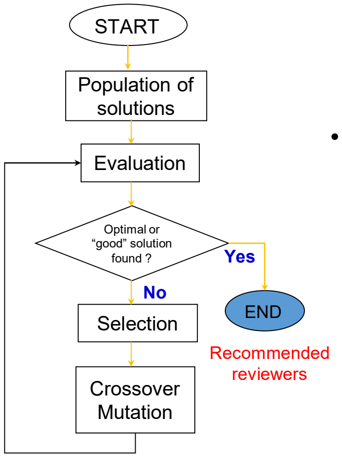
- Social network: each reviewer may have a review collaboration with
  - developer
  - other reviewers
- Graph representation
  - sub-graph connectivity
  - weights count on the edges (comments count)



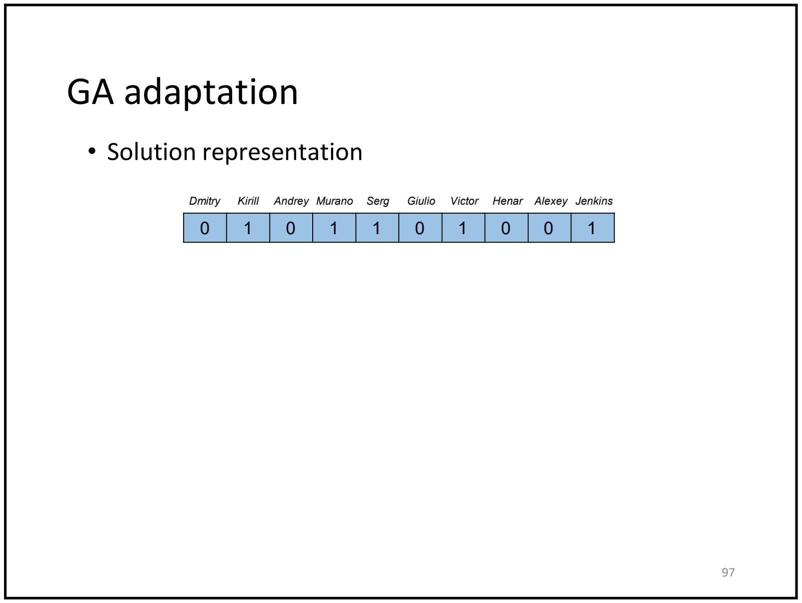
Owner	Stan Lagun
Reviewers	Jenkins Kirill Zaitsev Murano CI Serg Melikyan Victor Ryzhenkin
Project	openstack/murano
Branch	master
Topic	murano-object-smarttype
Updated	5 weeks ago



Genetic algorithm (GA)



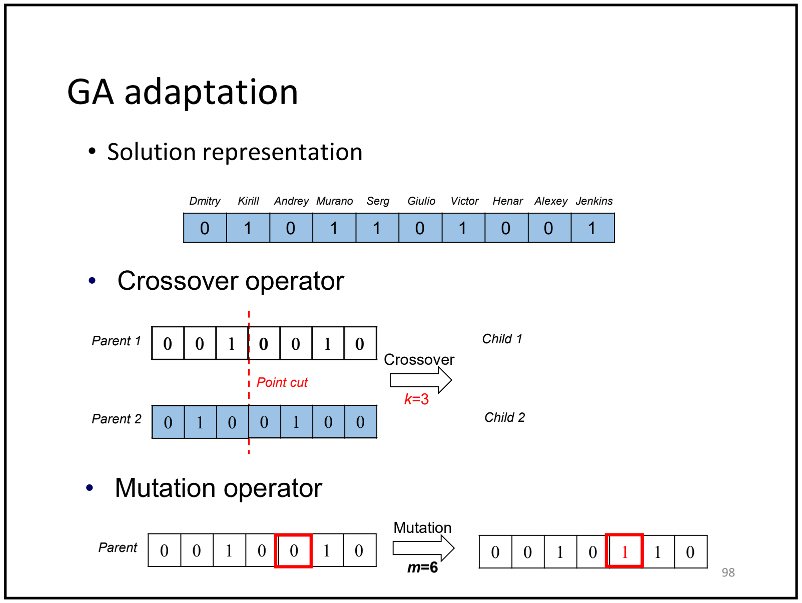
- Key elements
  - Solution representation
  - Change operators
  - Fitness function
  - Selection
  - Creation of the initial population



GA adaptation

- Solution representation

Dmitry	Kirill	Andrey	Murano	Serg	Giulio	Victor	Henar	Alexey	Jenkins
0	1	0	1	1	0	1	0	0	1



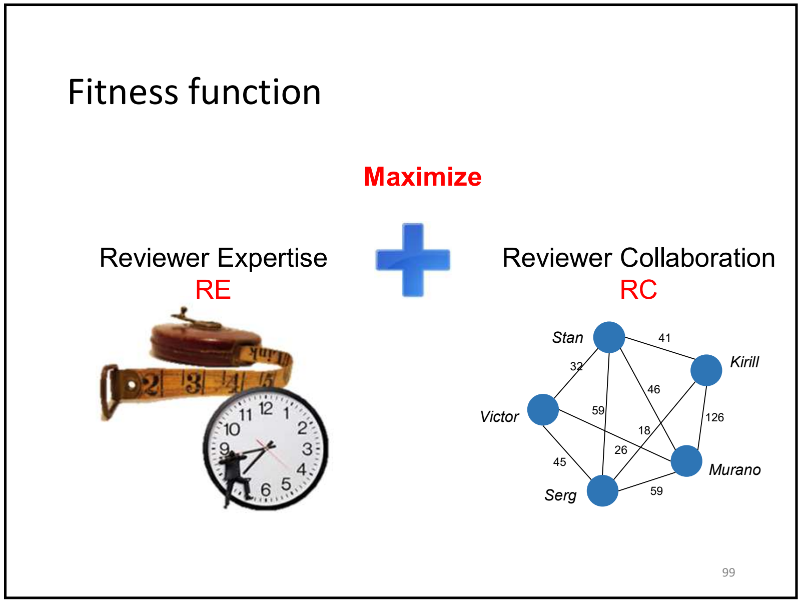
GA adaptation

- Solution representation

Dmitry	Kirill	Andrey	Murano	Serg	Giulio	Victor	Henar	Alexey	Jenkins
0	1	0	1	1	0	1	0	0	1

- Crossover operator

- Mutation operator

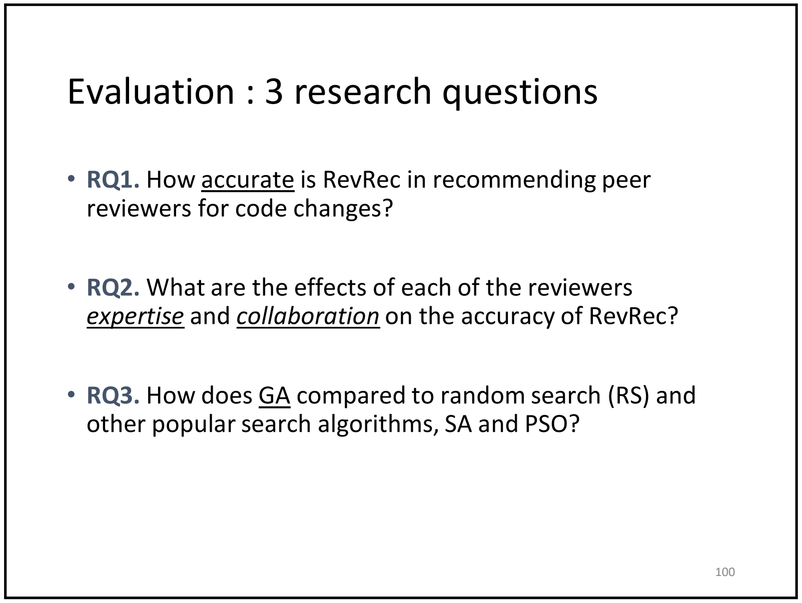


Fitness function

Maximize

Reviewer Expertise
RE

Reviewer Collaboration
RC



Evaluation : 3 research questions

- **RQ1.** How accurate is RevRec in recommending peer reviewers for code changes?
- **RQ2.** What are the effects of each of the reviewers *expertise* and *collaboration* on the accuracy of RevRec?
- **RQ3.** How does GA compared to random search (RS) and other popular search algorithms, SA and PSO?

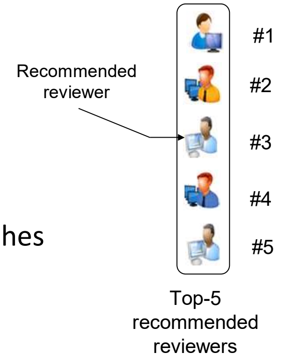
Studied systems

System	Period studied	#Reviews	#Reviewers	#Files
Android	2008-10 ~ 2010-01	5,126	94	26,840
OpenStack	2011-04 ~ 2012-05	6,586	82	16,953
Qt	2011-05 ~ 2012-05	23,810	202	78,401

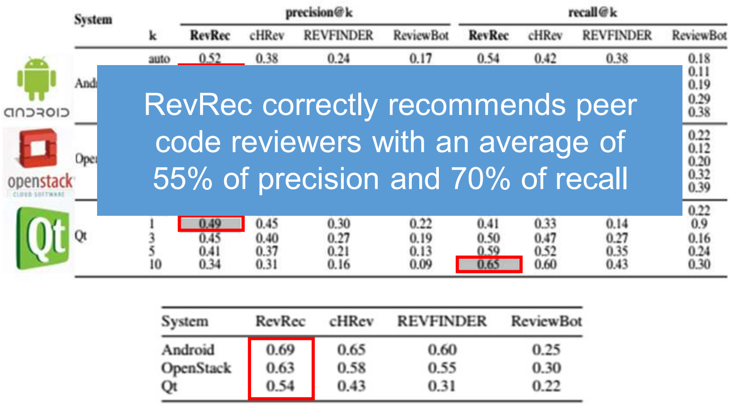
Dataset: http://kin-y.github.io/miningReviewRepo/

Analysis method

- Accuracy
  - Precision@k
  - Recall@k
- Ranking performance
  - Mean Reciprocal Rank (MRR)
- Compare with 3 existing approaches
  - RevFinder [[Patanamon et al., 2015]
  - cHRec [Zanjani et al., 2015]
  - ReviewBot [Balachandran et al. 2013]



RQ1. Accuracy results

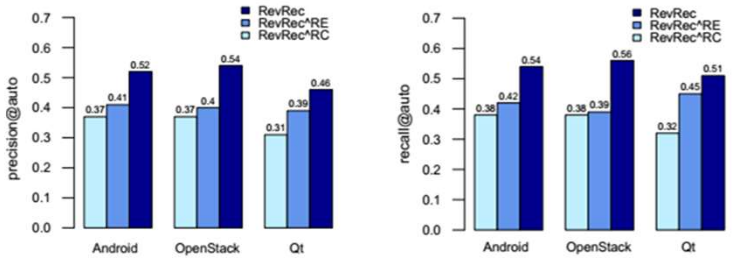


System	k	precision@k				recall@k			
		RevRec	cHRev	REVFINDER	ReviewBot	RevRec	cHRev	REVFINDER	ReviewBot
Android	auto	0.52	0.38	0.24	0.17	0.54	0.42	0.38	0.18
									0.11
									0.19
									0.29
									0.38
OpenStack									0.22
									0.12
									0.20
									0.32
									0.39
Qt									0.22
	1	0.49	0.45	0.30	0.22	0.41	0.33	0.14	0.9
	3	0.45	0.40	0.27	0.19	0.50	0.47	0.27	0.16
	5	0.41	0.37	0.21	0.13	0.59	0.52	0.35	0.24
	10	0.34	0.31	0.16	0.09	0.65	0.60	0.43	0.30

RevRec correctly recommends peer code reviewers with an average of 55% of precision and 70% of recall

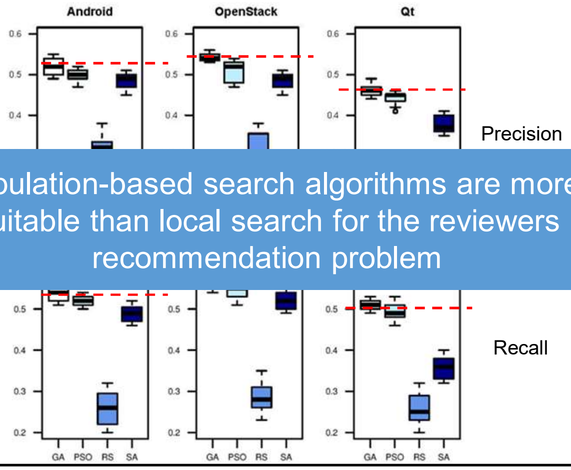
System	RevRec	cHRev	REVFINDER	ReviewBot
Android	0.69	0.65	0.60	0.25
OpenStack	0.63	0.58	0.55	0.30
Qt	0.54	0.43	0.31	0.22

RQ2. Expertise vs. Collaboration



The social aspect plays an important role in modern code review

RQ3. Performance of Genetic Algorithm



Population-based search algorithms are more suitable than local search for the reviewers recommendation problem

RevRec : Pros and Cons

- Empirical evaluation on 3 open-source projects
  - Promising accuracy results: **55%** of precision and **70%** of recall
  - Social aspect plays an important role in modern code review
  - Golbal search achieves better performance than local search
- Limitations
  - Reviewers workload is not considered
  - 80% of reviews are assigned to 20% of reviewers!

Solution : Optimize *Expertise* and *Workload*

Recommending Peer Reviewers in Modern Code Review : A Multi-Objective Search-based Approach

Motaz Chouchen
ETS Montreal, University of Quebec
Montreal, QC, Canada
moataz.chouchen.1@ens.etsmtl.ca

Ali Ouni
ETS Montreal, University of Quebec
Montreal, QC, Canada
ali.ouni@etsmtl.ca

Mohamed Wiem Mkaouer
Rochester Institute of Technology
Rochester, NY, USA
mwmvse@rit.edu

Raula Gaikovina Kula
Nara Institute of Science and
Technology, Nara, Japan
raula-k@is.naist.jp

Katsuro Inoue
Osaka University
Osaka, Japan
inoue@ist.osaka-u.ac.jp

ABSTRACT

Modern code review is a common practice used by software developers to ensure high software quality in open source and industrial projects. During code review, developers submit their code changes which should be reviewed, via tool-based code review platforms, before being integrated into the codebase. Then, reviewers provide their feedback to developers, and may request further modifications before finally accepting or rejecting the submitted code changes. However, the identification of appropriate reviewers is still a tedious task as the number of code reviews to be performed is inflated with the increasing number of code changes and the increasing size of software development teams in today's large and active software projects. To help developers with the review process, we introduce a multi-objective search-based approach to find the appropriate set of reviewers. We use the Non-dominated Sorting Genetic Algorithm (NSGA-II) to optimize two conflicting objectives (*i*) maximize reviewers expertise with the changed files, and (*ii*) minimize reviewers workload in terms of their current open code reviews. We conduct a preliminary evaluation on two open source projects to evaluate our approach. Results indicate that our approach is efficient as compared to state-of-the-art approaches.

CCS CONCEPTS

• **Software and its engineering** → Software maintenance tools; • **Human-centered computing** → Empirical studies in collaborative and social computing;

in-person meetings are required to discuss about potential problem with the code changes before they are merged with the code repository. MCR fosters less formal, lightweight and tool-based code review where developers can submit, discuss and rework code changes prior merging them.

Finding appropriate code reviewers is a non-trivial decision-making task in software engineering involving several considerations. As modern software projects code-base and development teams size are in constant increase, reviewers identification is becoming more challenging. In typical software projects, source files could be edited by several contributors, and reviewed by several reviewers. Contributors and reviewers could perform different software engineering tasks with different workloads. It is thus more difficult to identify suitable peer reviewers especially when the number of changed modules/files is large and reviewers are overloaded with different submitted code changes. Various approaches have been proposed to recommend code reviewers relying mostly on the expertise with the code files being submitted for review [5, 6, 8]. However, most of these approaches deal with the peer reviewers recommendation problem from a single perspective, *i.e.*, reviewers expertise and/or past collaborations, ignoring an important aspect which is the *reviewers workload*. Hence, reviewers are recommended regardless of their number of outstanding reviews.

To address this issue, we extend our previous work [5] by introducing a new reviewers recommendation formulation as a multi-objective search based problem. Our approach aims at finding the set of suitable developers that provide an optimal trade-off between

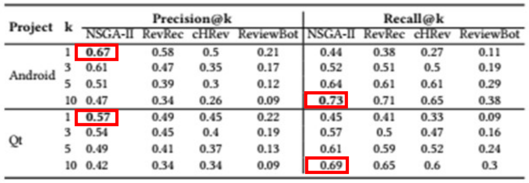
GECCO 2020

Multi-objective Code Reviewers Recommendation

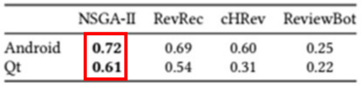
- Search algorithm
  - NSGA-II
- Objective function
  - Maximize reviewers expertise
  - Minimize reviewers workload
- Solution representation
  - Vector based representation

Evaluation

- Two long-lived systems
  - Android
  - Qt



Project	k	Precision@k				Recall@k			
		NSGA-II	RevRec	cHRev	ReviewBot	NSGA-II	RevRec	cHRev	ReviewBot
Android	1	**0.67**	0.58	0.5	0.21	0.44	0.38	0.27	0.11
	3	0.61	0.47	0.35	0.17	0.52	0.51	0.5	0.19
	5	0.51	0.39	0.3	0.12	0.64	0.61	0.61	0.29
	10	0.47	0.34	0.26	0.09	**0.73**	0.71	0.65	0.38
Qt	1	**0.57**	0.49	0.45	0.22	0.45	0.41	0.33	0.09
	3	0.54	0.45	0.4	0.19	0.57	0.5	0.47	0.16
	5	0.49	0.41	0.37	0.13	0.61	0.59	0.52	0.24
	10	0.42	0.34	0.34	0.09	0.69	0.65	0.6	0.3



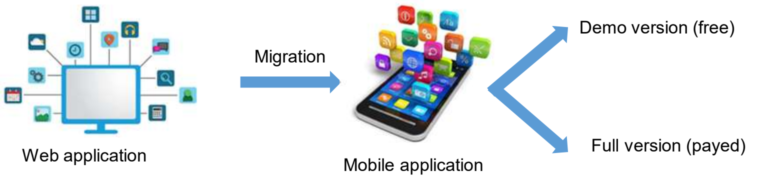
	NSGA-II	RevRec	cHRev	ReviewBot
Android	**0.72**	0.69	0.60	0.25
Qt	**0.61**	0.54	0.31	0.22

Agenda

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent applications
  - SBSE for Performance regression [SSBSE'19]
  - SBSE for Web service design [TSC'17 + ASE'19]
  - SBSE for Modern Code Review [ICSME'16 + GECCO'20]
- A hands-on activity with SBSE
  - MOEA Framework
- Challenges and future work with SBSE

SBSE with MOEA Framework

- MOEA Framework
  - http://moeaframework.org/
  - https://github.com/MOEAFramework/MOEAFramework
- Case study
  - Software Migration : components selection for mobile app migration



Web application

Migration

Mobile application

Demo version (free)

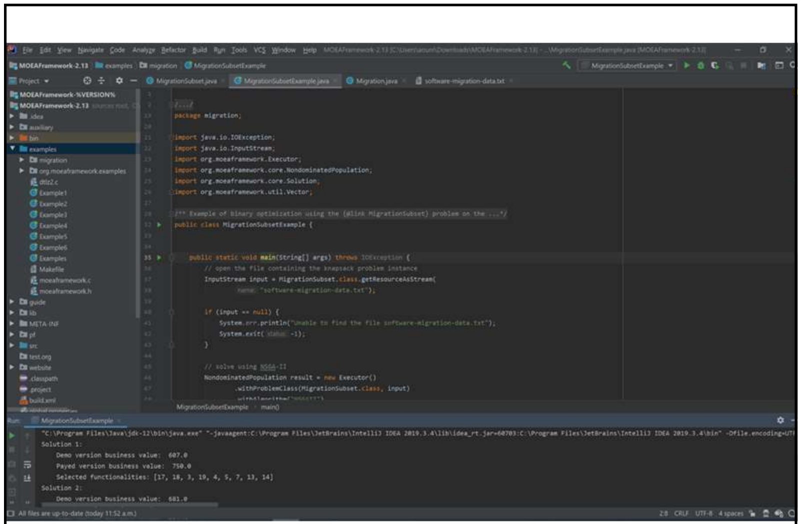
Full version (payed)

SBSE with MOEA Framework

- Desktop application
  - A number of functionalities
  - Not possible to implement all functionalities in the mobile version

- Each functionality has
  - A cost
  - A business value

- Cost and business value depend on the target mobile app (Demo/Full)
- Each mobile app version (Demo/Full) has
  - An allocated budget



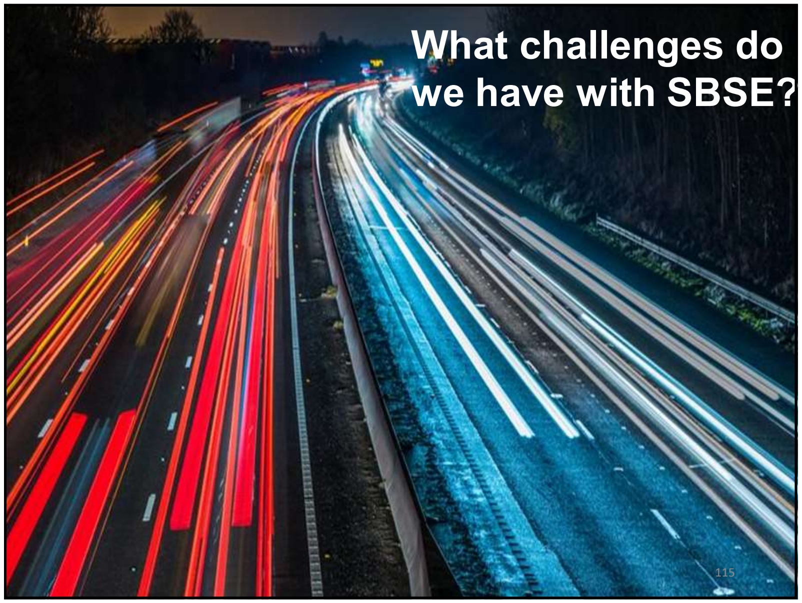
```java
package migration;

import java.io.IOException;
import java.io.InputStream;
import org.moeaframework.Executor;
import org.moeaframework.core.NondominatedPopulation;
import org.moeaframework.core.Solution;
import org.moeaframework.util.Vector;

/** Example of binary optimization using the {@link MigrationSubset} problem on the ...*/
public class MigrationSubsetExample {


    public static void main(String[] args) throws IOException {
        // open the file containing the knapsack problem instance
        InputStream input = MigrationSubset.class.getResourceAsStream(
                name: "software-migration-data.txt");

        if (input == null) {
            System.err.println("Unable to find the file software-migration-data.txt");
            System.exit( status: -1);
        }

        // solve using NSGA-II
        NondominatedPopulation result = new Executor()
                .withProblemClass(MigrationSubset.class, input)
```

```
"C:\Program Files\Java\jdk-12\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\lib\idea_rt.jar=60703:C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\bin" -Dfile.encoding=UT
Solution 1:
    Demo version business value:  607.0
    Payed version business value:  750.0
    Selected functionalities: [17, 18, 3, 19, 4, 5, 7, 13, 14]
Solution 2:
    Demo version business value:  681.0
```

https://github.com/ouniali/MOEAFramework

Agenda

- Philosophical Basis: Science and Engineering
- What is SBSE?
- Recent applications
  - SBSE for Performance regression [SSBSE'19]
  - SBSE for Web service design [TSC'17 + ASE'19]
  - SBSE for Modern Code Review [ICSME'16 + GECCO'20]
- A hands-on activity with SBSE
  - MOEA Framework
- Challenges and future work with SBSE



What challenges do we have with SBSE?

Challenges in the adoption of SBSE

- The selection of the solution representation and the right fitness function
- Changing the optimization algorithm may not necessarily change the output to better
- It is the coverage criteria and the finest function that lead to better results
- Parameters tuning is challenging!

SBSE challenges with industry

- The non-deterministic output due to the randomness
- Expensive computation
- Modeling the system or the problem space?
- This is crazy!

Probably we need to change our way of communication with industry

Reproducibility of SBSE solutions

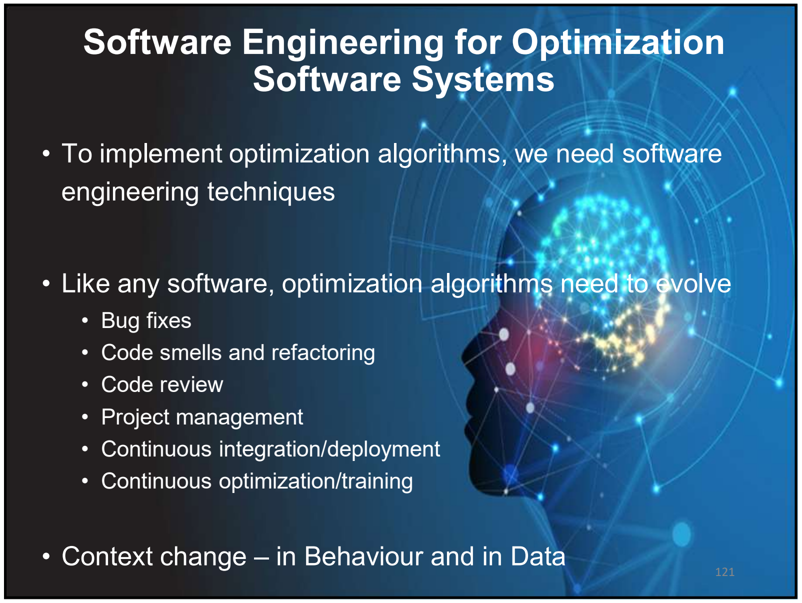
- Reproducibility
  - Indeterministic nature is a barrier
  - Hyperparamters tuning
  - Clear mathematical function of the objective functions
  - Descriptions of the settings
  - Datasets used (sometimes data confidentiality is a concern with several companies)
  - Training: how the dataset is split training-testing
  - Code : readme, specification of dependencies, etc.
- Take a paper : and write a reproductivity report
  - In a course project could be interesting for students
- Replication

Challenges and Open Research Directions

- Why do we currently need to **design special algorithms for each software engineering problem** instance?
  - This is unrealistic: Science is about generality. Several software engineering activities have a lot of common patterns and similarities
- Why do we currently **address silos of software engineering activity**?
  - This is unrealistic: engineering decision making needs to take account of requirements, designs, test cases and implementation details *simultaneously*.

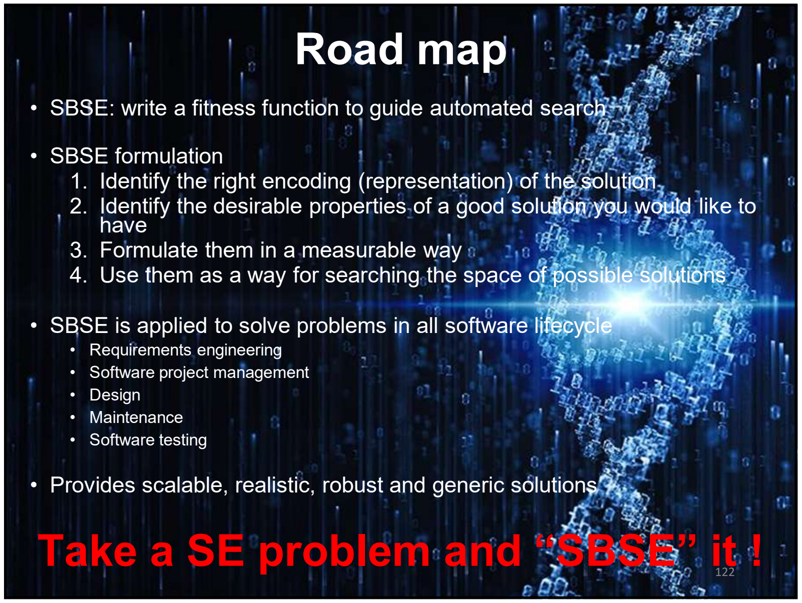
Challenges and Open Research Directions

- **Automation level**
  - How best do we draw the dividing line between adaptive automation for small changes and human intervention to invoke more fundamental adaption and to provide oversight and decision making?
- **Surrogate metrics**
  - Any approach that seeks dynamic adaptivity must necessarily compute many fitness evaluations between adaptations surrogate fitness computation will need to be fast.
- **Dynamic Adaptativity**



Software Engineering for Optimization Software Systems

- To implement optimization algorithms, we need software engineering techniques
- Like any software, optimization algorithms need to evolve
  - Bug fixes
  - Code smells and refactoring
  - Code review
  - Project management
  - Continuous integration/deployment
  - Continuous optimization/training
- Context change – in Behaviour and in Data

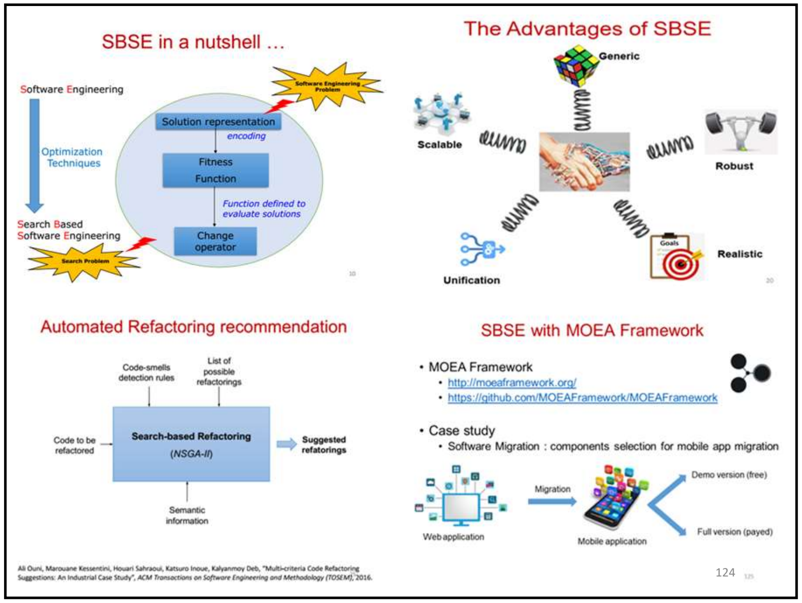


Road map

- SBSE: write a fitness function to guide automated search
- SBSE formulation
  1. Identify the right encoding (representation) of the solution
  2. Identify the desirable properties of a good solution you would like to have
  3. Formulate them in a measurable way
  4. Use them as a way for searching the space of possible solutions
- SBSE is applied to solve problems in all software lifecycle
  - Requirements engineering
  - Software project management
  - Design
  - Maintenance
  - Software testing
- Provides scalable, realistic, robust and generic solutions

Take a SE problem and "SBSE" it !

Search-based Software Engineering



SBSE in a nutshell …

The Advantages of SBSE

Automated Refactoring recommendation

Ali Ouni, Marouane Kessentini, Houari Sahraoui, Katsuro Inoue, Kalyanmoy Deb, "Multi-criteria Code Refactoring Suggestions: An Industrial Case Study", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2016.

SBSE with MOEA Framework

- MOEA Framework
  - http://moeaframework.org/
  - https://github.com/MOEAFramework/MOEAFramework
- Case study
  - Software Migration : components selection for mobile app migration

ÉTS
Le génie pour l'industrie

RIT
Rochester Institute of Technology

Thank You!

Questions?