Overview of SBSE

CS454 AI-Based Software Engineering

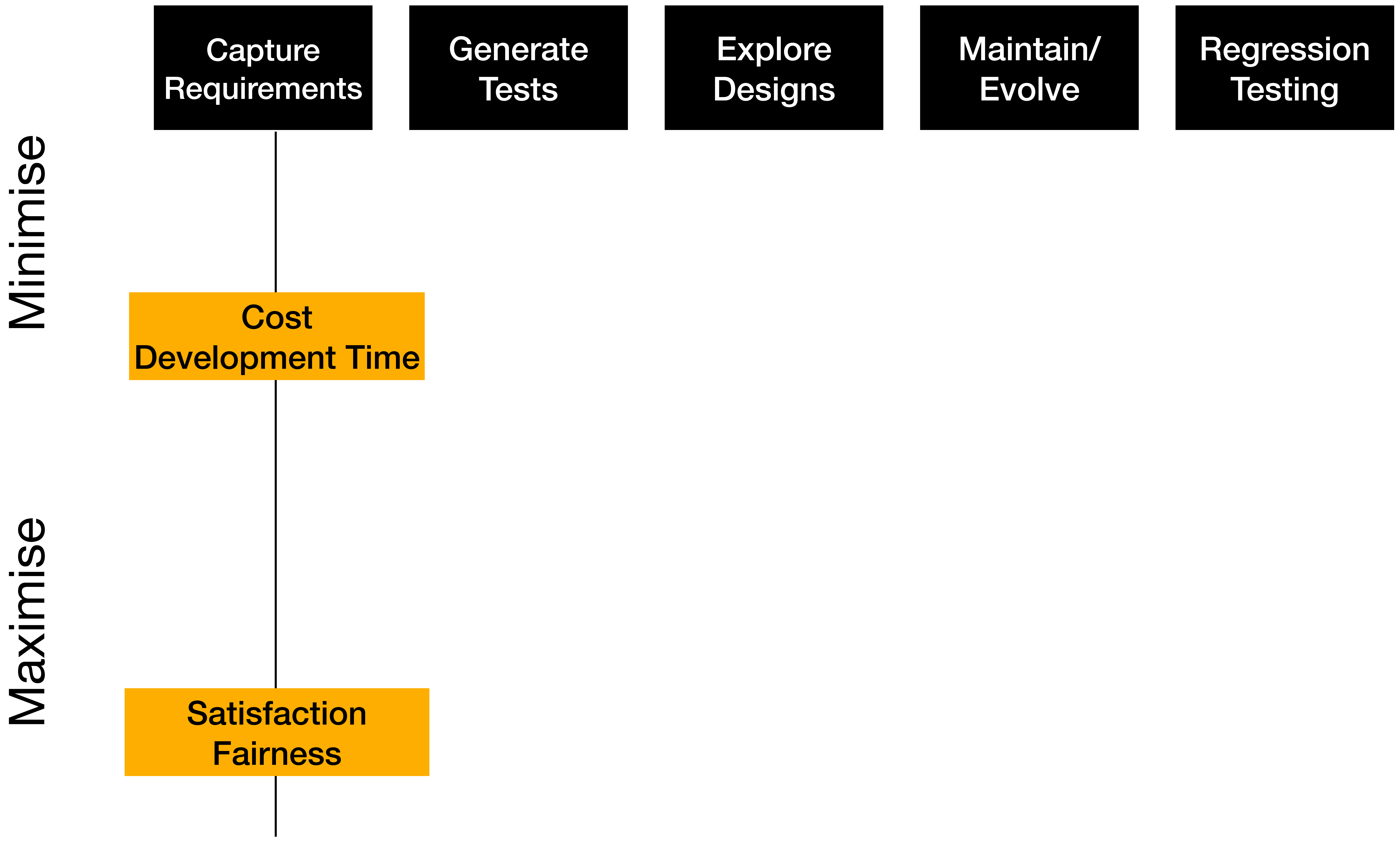
Shin Yoo

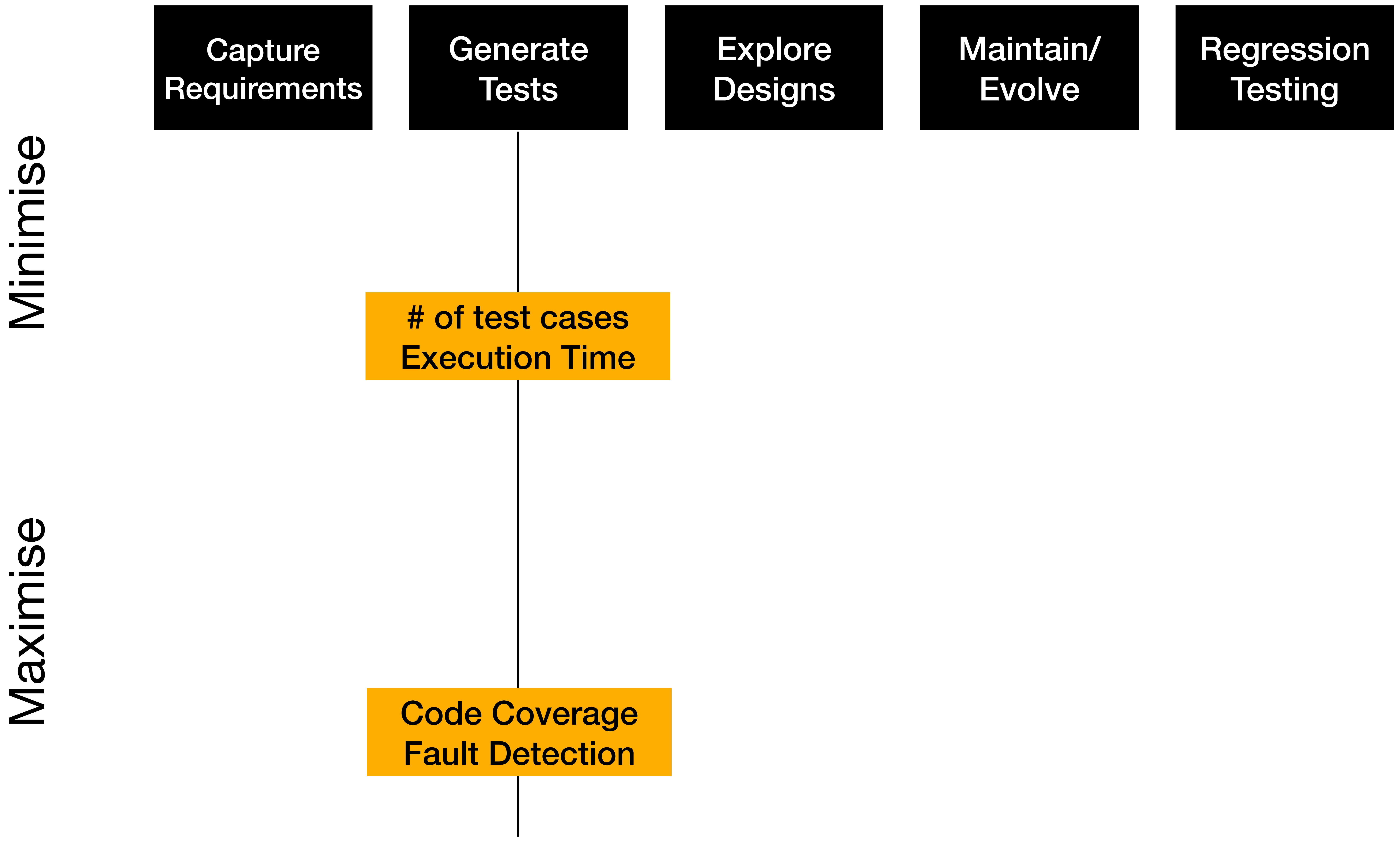
Search-Based Software Engineering

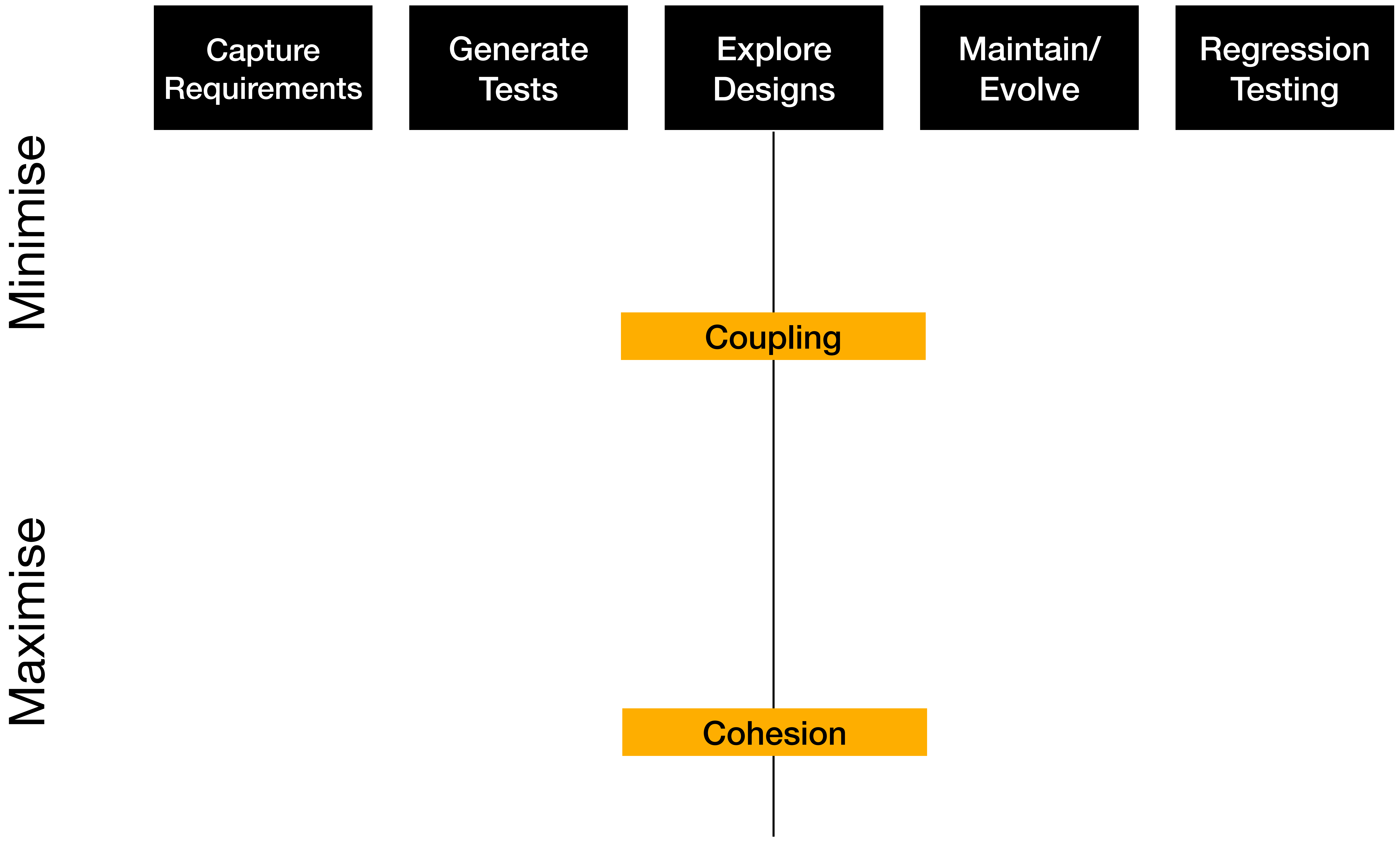
- Application of all the optimisation techniques we have seen so far, to the various problems in software engineering.
- Not web search engines :(
- Not code search :(

SBSE is a methodology

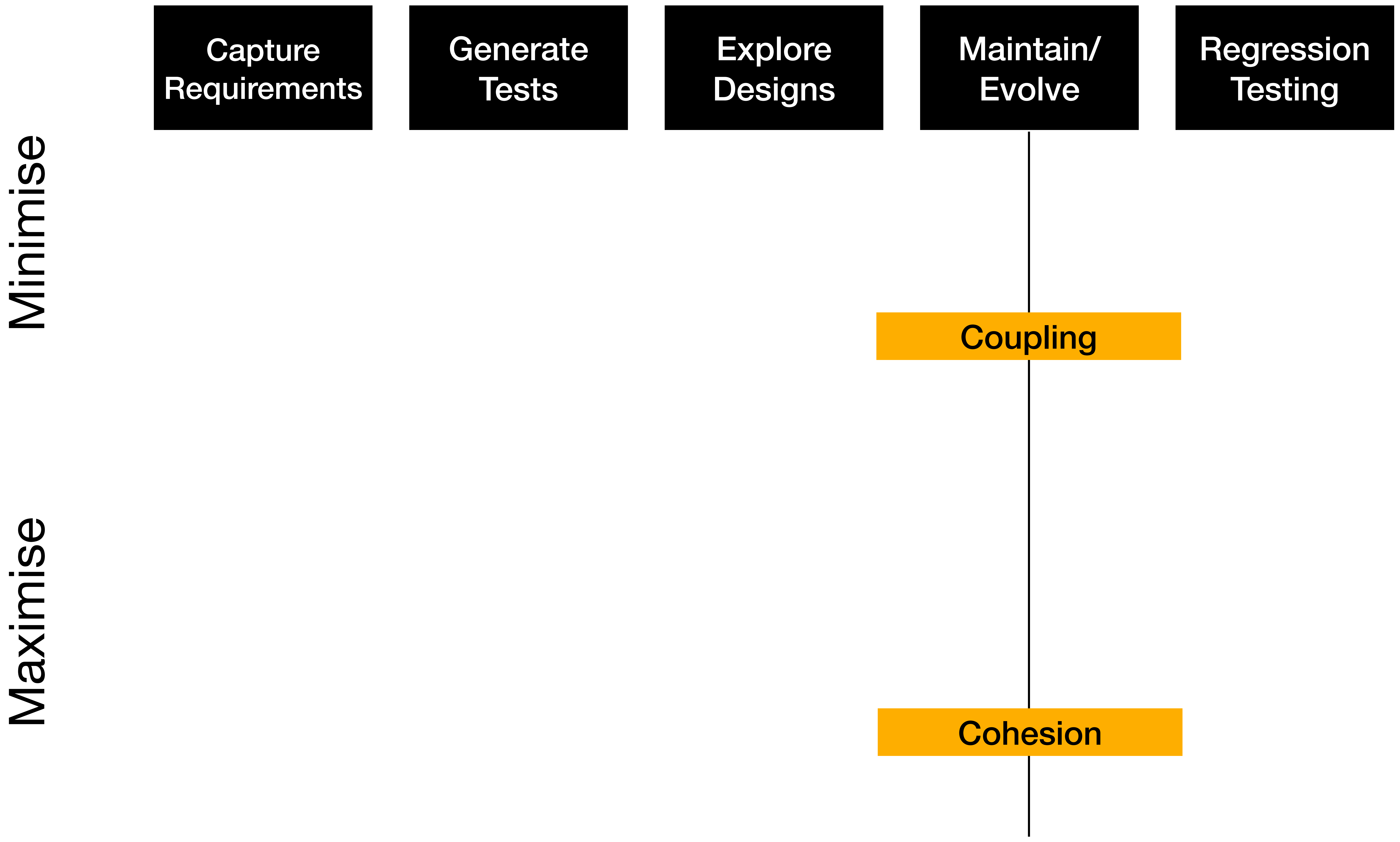
- Looking at everything from a methodology point of view has its own pros and cons
  - Cons: when you have a hammer, everything looks like a nail.
  - Pros
    - You find yourself trying to justify why two very different things are both nails, i.e., sometimes you see a cross-cutting theme in different domains :)
    - A better hammer makes everything easier (as long as they are nails)


Capture Requirements
Generate Tests
Explore Designs
Maintain/ Evolve
Regression Testing
Minimise
Cost
Development Time
Maximise
Satisfaction
Fairness


Capture Requirements
Generate Tests
Explore Designs
Maintain/ Evolve
Regression Testing
Minimise
Maximise
# of test cases
Execution Time
Code Coverage
Fault Detection



Capture Requirements
Generate Tests
Explore Designs
Maintain/ Evolve
Regression Testing
Minimise
Maximise
Coupling
Cohesion


Capture Requirements
Generate Tests
Explore Designs
Maintain/ Evolve
Regression Testing
Minimise
Maximise
Coupling
Cohesion


Capture Requirements
Generate Tests
Explore Designs
Maintain/ Evolve
Regression Testing
Minimise
Maximise
# of Test Cases
Execution Time
Coverage
Fault Coverage

Good Starting Points

- M. Harman. The current state and future of search based software engineering. In FOSE ’07: 2007 Future of Software Engineering, pages 342–357,2007.
- M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 45(1):11:1–11:61, December 2012.

Cost Estimation

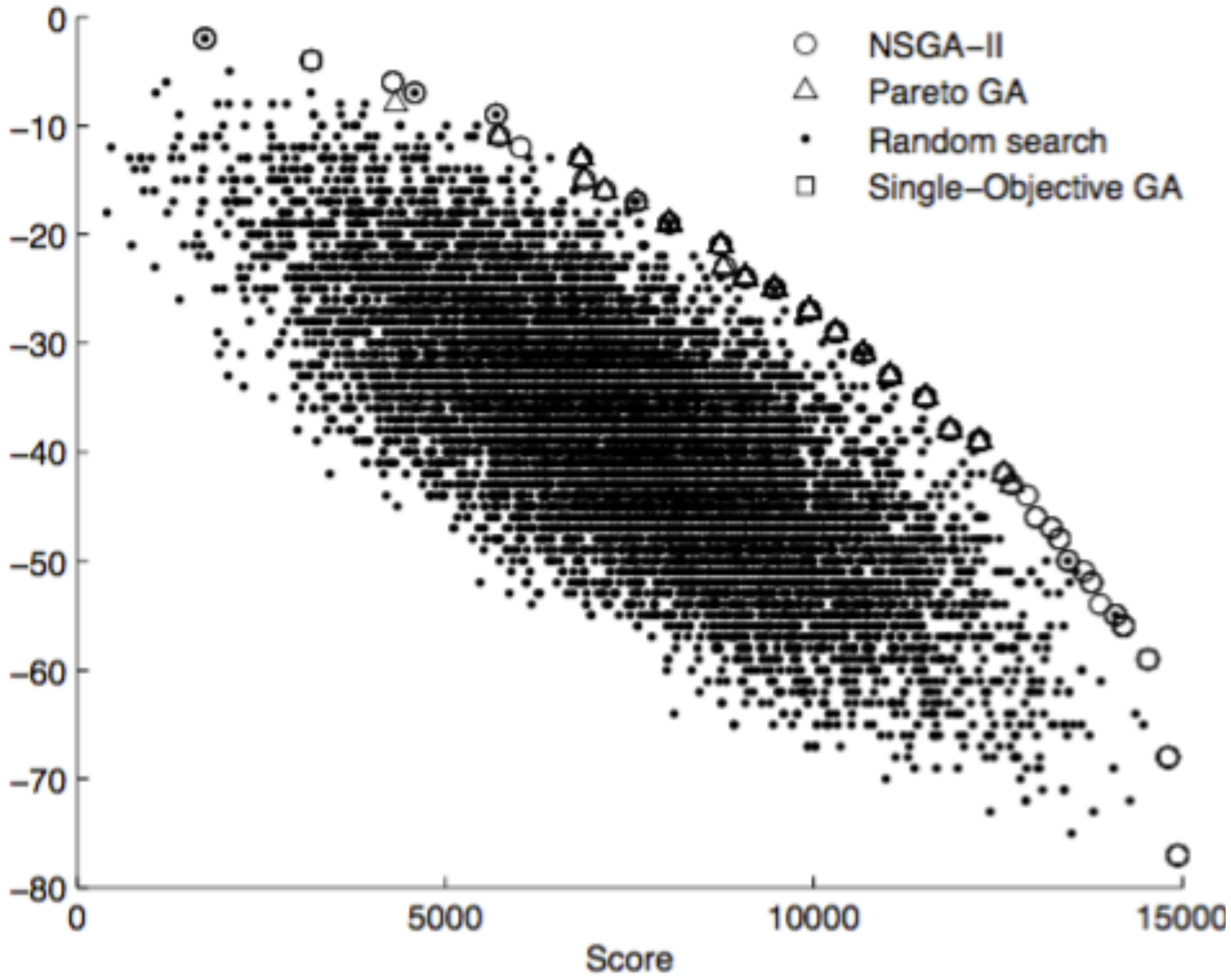
- Evolve mathematical functions (symbolic regression) that would predict the project development effort based on various input variables.
  - J. J. Dolado. A validation of the component-based method for software size estimation. IEEE Transactions on Software Engineering, 26(10):1006–1021, 2000.

Project Planning

- Team allocation to project work packages, including the possibility of abandonment (i.e. work no longer needed/practical) and rework (i.e. additional work needed).
    - G. Antoniol, M. Di Penta, and M. Harman. A Robust Search-based Approach to Project Management in the Presence of Abandonment, Rework, Error and Uncertainty. In Proceedings of the 10th International Symposium on the Software Metrics (METRICS ’04), pages 172–183, Chicago, USA, 11-17 September 2004. IEEE Computer Society.

Next Release Problem

- Find the ideal set of requirements that balances customer requests, resource constraints, and interdependencies between requirements.
  - A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem. Information and Software Technology, 43(14):883–890, Dec. 2001.
  - Y. Zhang, M. Harman, and S. A. Mansouri. The Multi-Objective Next Release Problem. In GECCO '07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference, pages 1129–1136. ACM Press, 2007.



(d) 100 customers; 20 requirements

Optimising Source Code

- Random sampling of code transformation to find compiler optimisation
  - K. D. Cooper, P. J. Schielke, and D. Subramanian. Optimizing for reduced code space using genetic algorithms. In Proceedings of the ACM SIGPLAN 1999 Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'99), volume 34.7 of ACM SIGPLAN Notices, pages 1–9, NY, May 5 1999. ACM Press.
- Automated Parallelisation
  - K. P. Williams. Evolutionary Algorithms for Automatic Parallelization. PhD thesis, University of Reading, UK, Department of Computer Science, Sept. 1998.

Test Data Generation

- Many, many different approaches and ideas; too many to list all:
    - P. McMinn. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 14(2):105–156, June 2004.

Regression Testing

- Pareto-efficient Test Suite Minimisation:
    - S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In Proceedings of International Symposium on Software Testing and Analysis, pages 140–150. ACM Press, July 2007.
- Test Case Prioritisation:
    - Z. Li, M. Harman, and R. M. Hierons. Search Algorithms for Regression Test Case Prioritization. IEEE Transactions on Software Engineering, 33(4):225–237, 2007.
- Multi-objective Prioritisation:
    - M. G. Epitropakis, S. Yoo, M. Harman, and E. K. Burke. Empirical evaluation of pareto efficient multi- objective regression test case prioritisation. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, pages 234–245, New York, NY, USA, 2015. ACM.

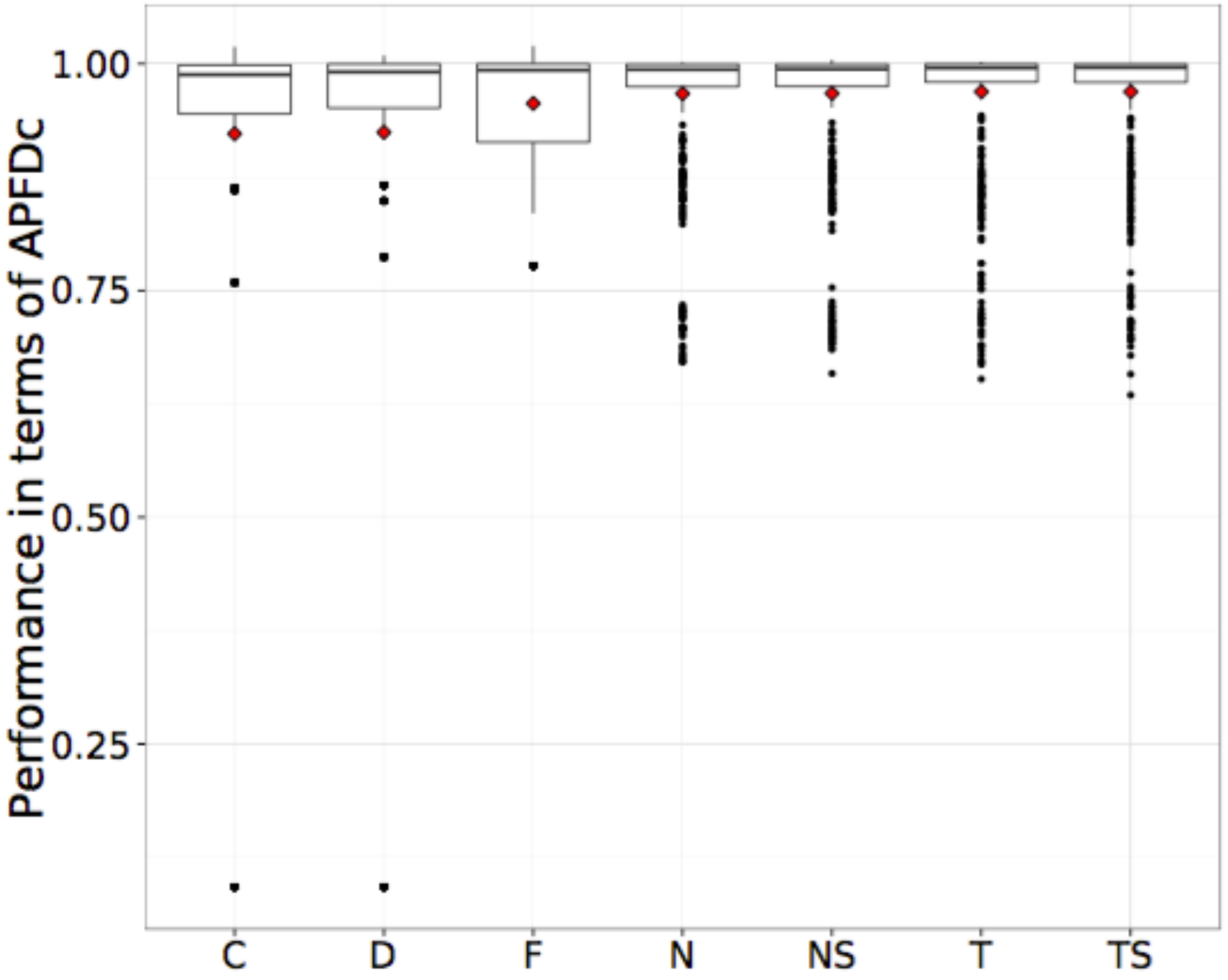


Figure 3: Boxplots of the APFD$_c$ metric across all studied subjects. MOEAs and their variants show higher median values and smaller variances.

Maintenance & Reverse Engineering

- Module Clustering: assign modules to clusters based on their relationships
  - B. S. Mitchell and S. Mancoridis. On the automatic modularization of software systems using the bunch tool. IEEE Transactions on Software Engineering, 32(3):193–208, 2006.
  - K. Praditwong, M. Harman, and X. Yao. Software module clustering as a multi-objective search problem. IEEE Transactions on Software Engineering, 37(2):264–282, March-April 2010.

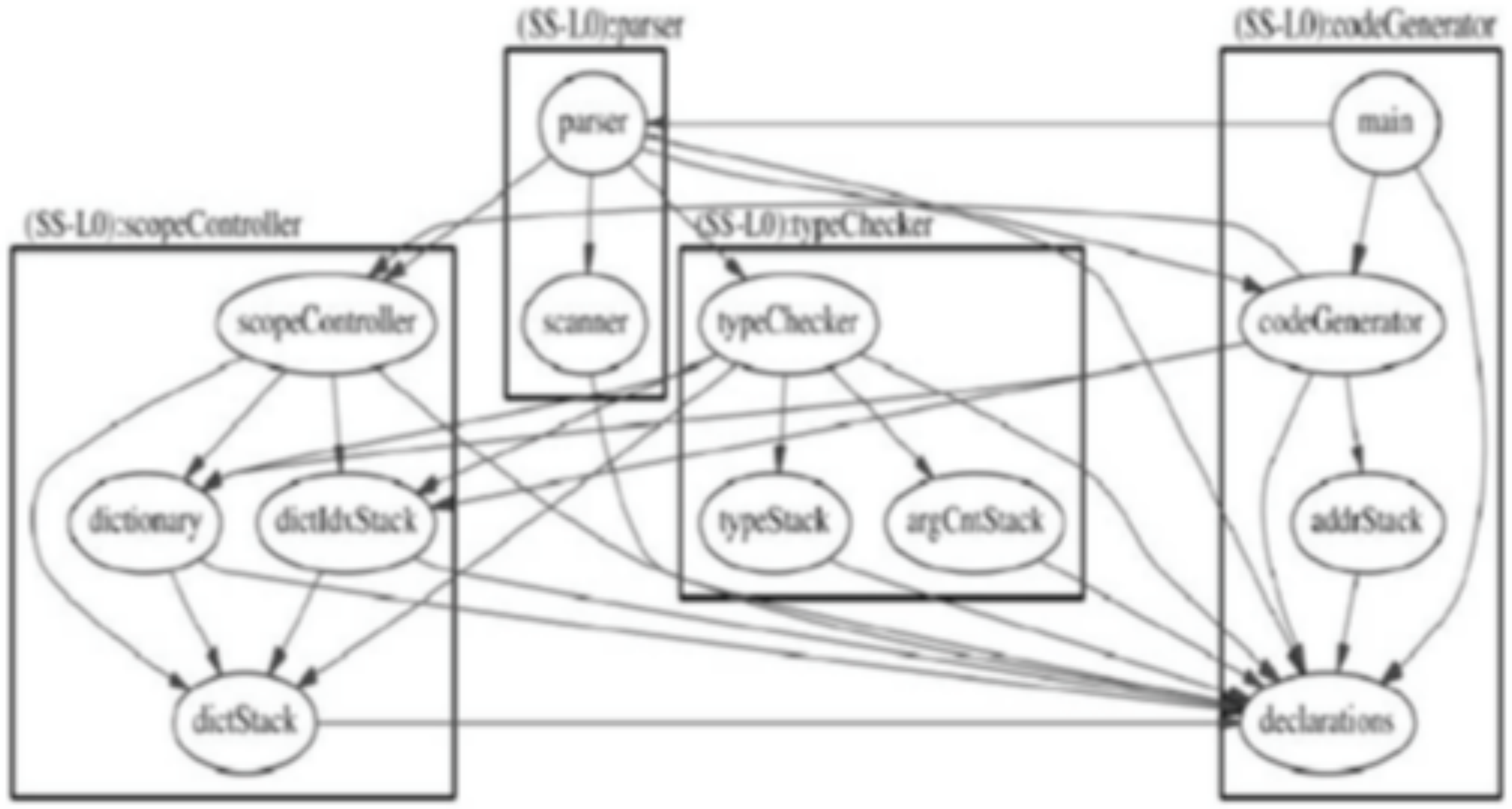
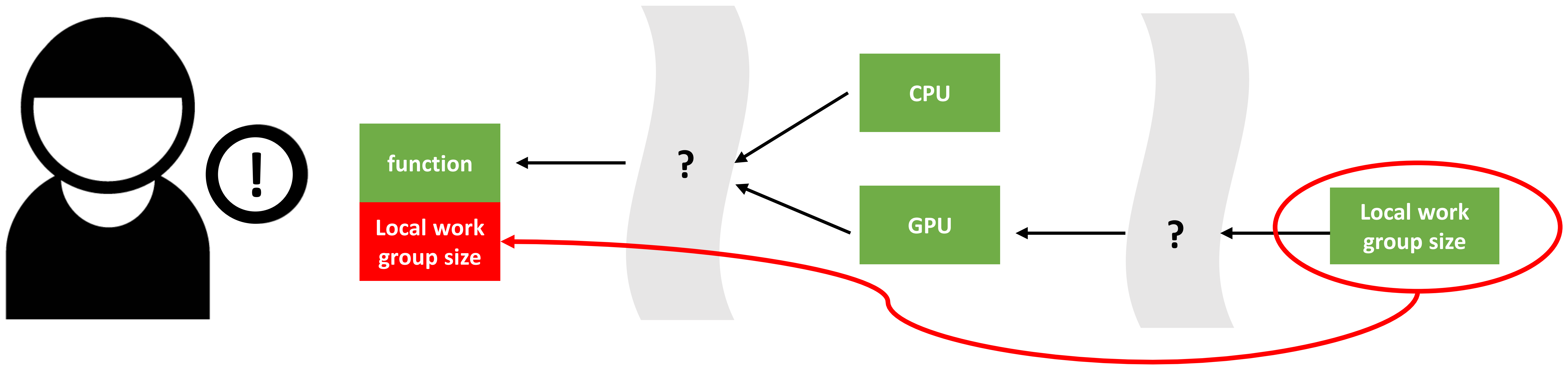


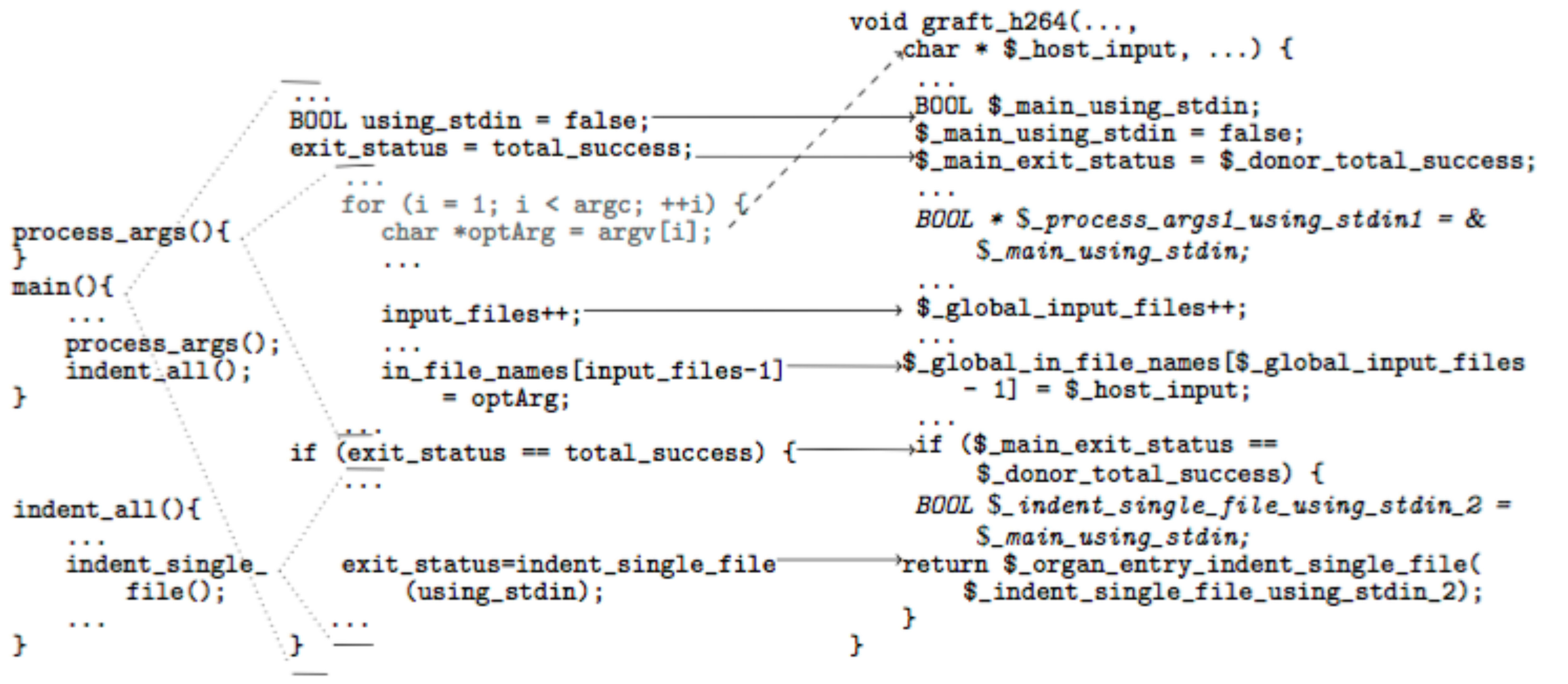
Figure 3. A Module Dependency Graph and its Modularisation using Bunch, taken from [65]

Deep Parameter Optimisation

- Reveal a property hidden in software as a parameter for tuning.
  - F. Wu, W. Weimer, M. Harman, Y. Jia, and J. Krinke. Deep parameter optimisation. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO 2015, pages 1375–1382, 2015.
  - J. Sohn, S. Lee, and S. Yoo. Amortised deep parameter optimisation of GPGPU work group size for OpenCV. In F. Sarro and K. Deb, editors, Proceedings of the 8th International Symposium on Search Based Software Engineering, volume 9962 of Lecture Notes in Computer Science, pages 211–217. Springer International Publishing, 2016.



Code Transplantation



(a) Indent Code (b) Indent Inlined Source Code (c) Transplanted Source Code

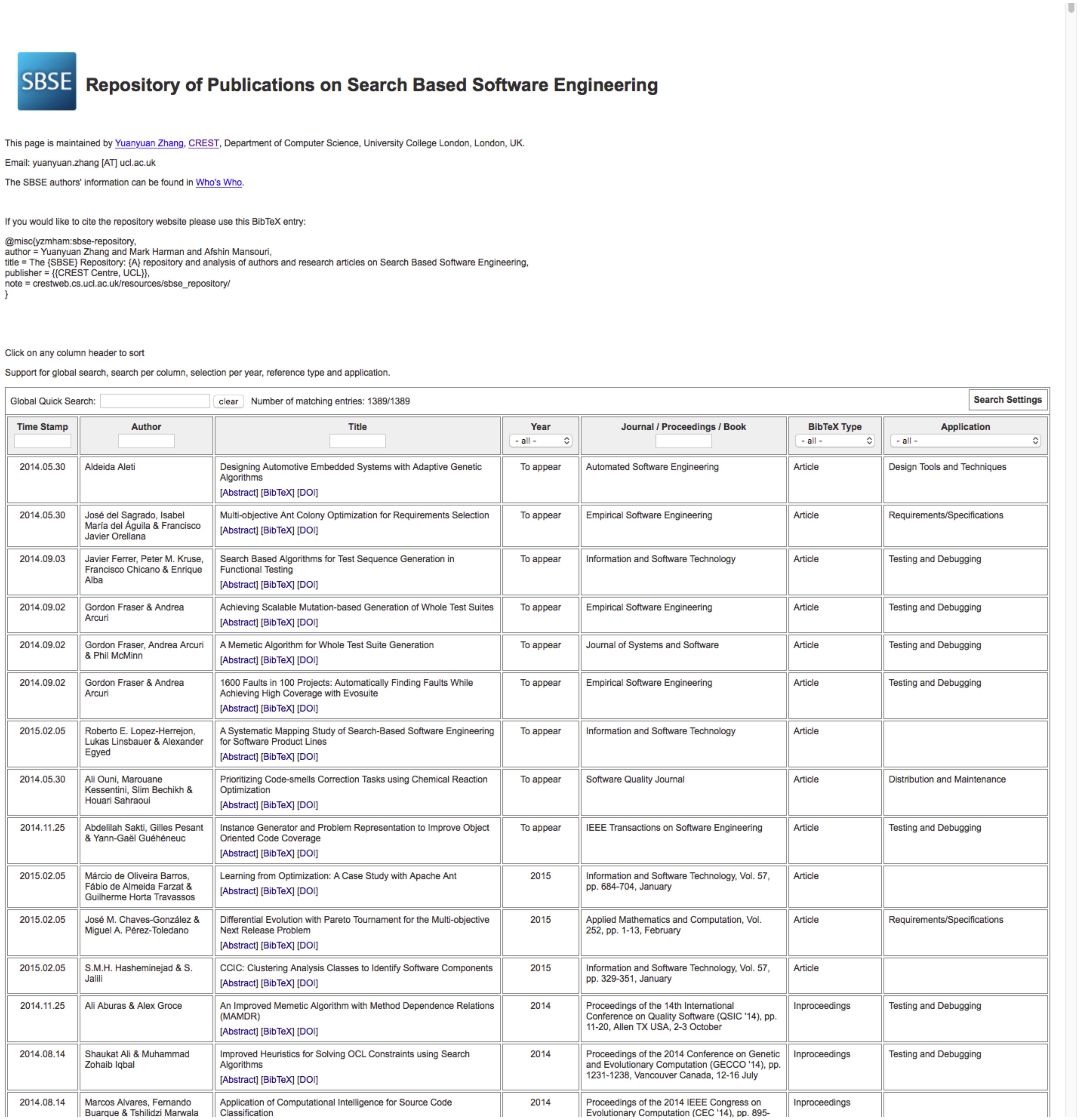
Fig. 1: Transplant operation in Cflow donor transplant. Code snippet from the beginning of the graft. ⋯⊣ means function inlining; `optArg` is mapped to `$_host_input`; → means original statement replacement under α — renaming; grayed statements are deleted.

- E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke. Automated software transplantation. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, pages 257–269, New York, NY, USA, 2015. ACM.

- A. Marginean, E. Barr, M. Harman, and Y. Jia. Automated transplantation of call graph and layout features into kate. In M. Barros and Y. Labiche, editors, Search-Based Software Engineering, volume 9275 of Lecture Notes in Computer Science, pages 262–268. Springer International Publishing, 2015.

SBSE Repository

- Most of the papers published on SBSE, stored and categorised online:
- http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/



SBSE Repository of Publications on Search Based Software Engineering

This page is maintained by Yuanyuan Zhang, CREST, Department of Computer Science, University College London, London, UK.

Email: yuanyuan.zhang [AT] ucl.ac.uk

The SBSE authors' information can be found in Who's Who.

If you would like to cite the repository website please use this BibTeX entry:

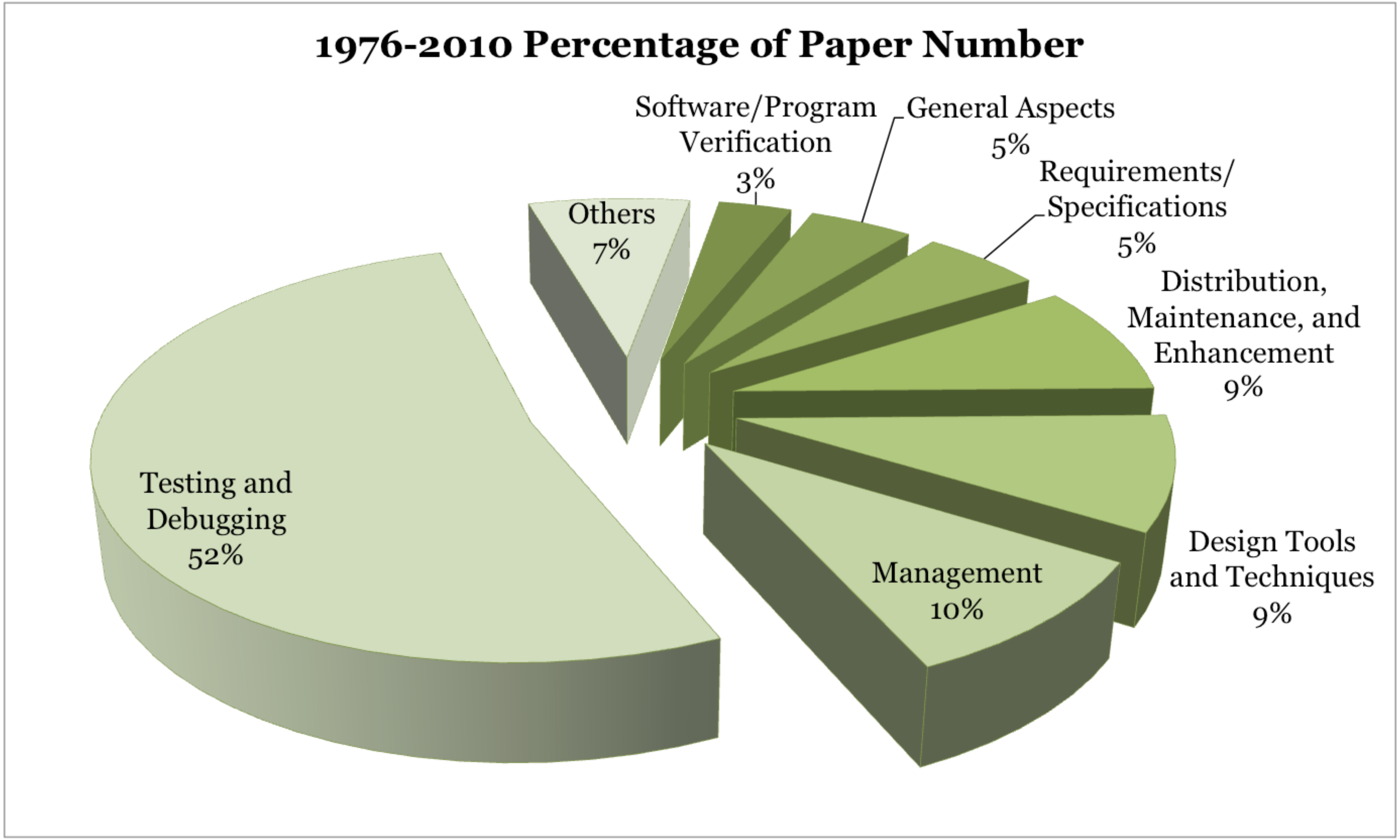
```
@misc{yzmham:sbse-repository,
author = Yuanyuan Zhang and Mark Harman and Afshin Mansouri,
title = The {SBSE} Repository: {A} repository and analysis of authors and research articles on Search Based Software Engineering,
publisher = {{CREST Centre, UCL}},
note = crestweb.cs.ucl.ac.uk/resources/sbse_repository/
}
```

Click on any column header to sort

Support for global search, search per column, selection per year, reference type and application.

Global Quick Search: clear Number of matching entries: 1389/1389 Search Settings

Time Stamp	Author	Title	Year - all -	Journal / Proceedings / Book	BibTeX Type - all -	Application - all -
2014.05.30	Aldeida Aleti	Designing Automotive Embedded Systems with Adaptive Genetic Algorithms [Abstract] [BibTeX] [DOI]	To appear	Automated Software Engineering	Article	Design Tools and Techniques
2014.05.30	José del Sagrado, Isabel María del Águila & Francisco Javier Orellana	Multi-objective Ant Colony Optimization for Requirements Selection [Abstract] [BibTeX] [DOI]	To appear	Empirical Software Engineering	Article	Requirements/Specifications
2014.09.03	Javier Ferrer, Peter M. Kruse, Francisco Chicano & Enrique Alba	Search Based Algorithms for Test Sequence Generation in Functional Testing [Abstract] [BibTeX] [DOI]	To appear	Information and Software Technology	Article	Testing and Debugging
2014.09.02	Gordon Fraser & Andrea Arcuri	Achieving Scalable Mutation-based Generation of Whole Test Suites [Abstract] [BibTeX] [DOI]	To appear	Empirical Software Engineering	Article	Testing and Debugging
2014.09.02	Gordon Fraser, Andrea Arcuri & Phil McMinn	A Memetic Algorithm for Whole Test Suite Generation [Abstract] [BibTeX] [DOI]	To appear	Journal of Systems and Software	Article	Testing and Debugging
2014.09.02	Gordon Fraser & Andrea Arcuri	1600 Faults in 100 Projects: Automatically Finding Faults While Achieving High Coverage with Evosuite [Abstract] [BibTeX] [DOI]	To appear	Empirical Software Engineering	Article	Testing and Debugging
2015.02.05	Roberto E. Lopez-Herrejon, Lukas Linsbauer & Alexander Egyed	A Systematic Mapping Study of Search-Based Software Engineering for Software Product Lines [Abstract] [BibTeX] [DOI]	To appear	Information and Software Technology	Article	
2014.05.30	Ali Ouni, Marouane Kessentini, Slim Bechikh & Houari Sahraoui	Prioritizing Code-smells Correction Tasks using Chemical Reaction Optimization [Abstract] [BibTeX] [DOI]	To appear	Software Quality Journal	Article	Distribution and Maintenance
2014.11.25	Abdelilah Sakti, Gilles Pesant & Yann-Gaël Guéhéneuc	Instance Generator and Problem Representation to Improve Object Oriented Code Coverage [Abstract] [BibTeX] [DOI]	To appear	IEEE Transactions on Software Engineering	Article	Testing and Debugging
2015.02.05	Márcio de Oliveira Barros, Fábio de Almeida Farzat & Guilherme Horta Travassos	Learning from Optimization: A Case Study with Apache Ant [Abstract] [BibTeX] [DOI]	2015	Information and Software Technology, Vol. 57, pp. 684-704, January	Article	
2015.02.05	José M. Chaves-González & Miguel A. Pérez-Toledano	Differential Evolution with Pareto Tournament for the Multi-objective Next Release Problem [Abstract] [BibTeX] [DOI]	2015	Applied Mathematics and Computation, Vol. 252, pp. 1-13, February	Article	Requirements/Specifications
2015.02.05	S.M.H. Hasheminejad & S. Jalili	CCIC: Clustering Analysis Classes to Identify Software Components [Abstract] [BibTeX] [DOI]	2015	Information and Software Technology, Vol. 57, pp. 329-351, January	Article	
2014.11.25	Ali Aburas & Alex Groce	An Improved Memetic Algorithm with Method Dependence Relations (MAMDR) [Abstract] [BibTeX] [DOI]	2014	Proceedings of the 14th International Conference on Quality Software (QSIC '14), pp. 11-20, Allen TX USA, 2-3 October	Inproceedings	Testing and Debugging
2014.08.14	Shaukat Ali & Muhammad Zohaib Iqbal	Improved Heuristics for Solving OCL Constraints using Search Algorithms [Abstract] [BibTeX] [DOI]	2014	Proceedings of the 2014 Conference on Genetic and Evolutionary Computation (GECCO '14), pp. 1231-1238, Vancouver Canada, 12-16 July	Inproceedings	Testing and Debugging
2014.08.14	Marcos Alvares, Fernando Buarque & Tshilidzi Marwala	Application of Computational Intelligence for Source Code Classification	2014	Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC '14), pp. 895-	Inproceedings	



The ratio of SE research fields that involved in SBSE.

CLOSE ✕

How about other (classical) ML techniques?

- Many SE techniques are about automation, so ML is not new to SE.
- SE techniques naturally benefit from any advances in ML (or SE is a good application area for any serious ML technique):
  - Clustering, predictive modelling, recommendataion system…

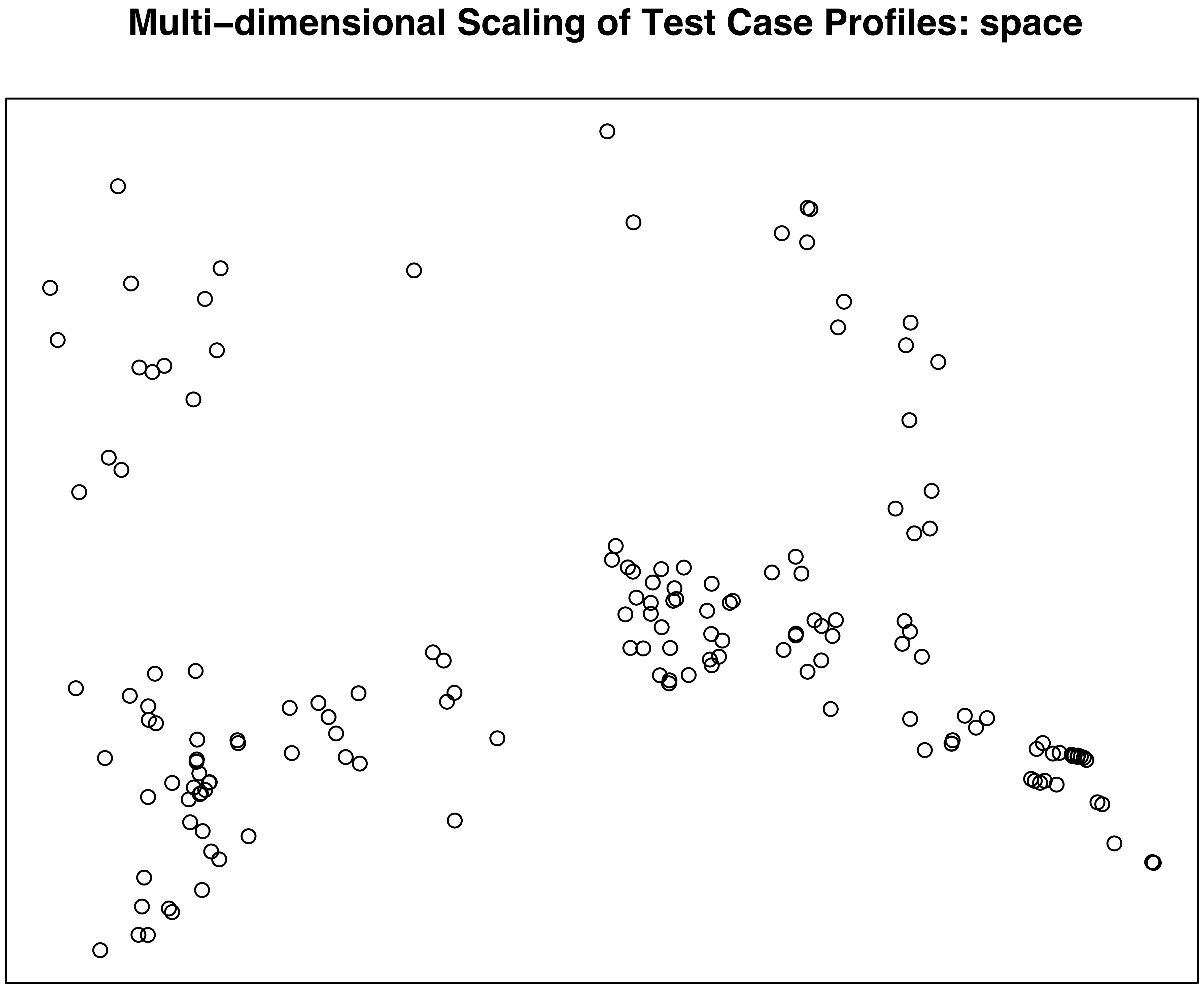
Clustering

- Clustering is one of the representative form of unsupervised learning
- Whenever you suspect there are internal patterns in a problem, you can attempt clustering to reveal and exploit the pattern

Maintenance & Reverse Engineering

- Module Clustering: assign modules to clusters based on their relationships
  - B. S. Mitchell and S. Mancoridis. On the automatic modularization of software systems using the bunch tool. IEEE Transactions on Software Engineering, 32(3):193–208, 2006.
  - K. Praditwong, M. Harman, and X. Yao. Software module clustering as a multi-objective search problem. IEEE Transactions on Software Engineering, 37(2):264–282, March-April 2010.

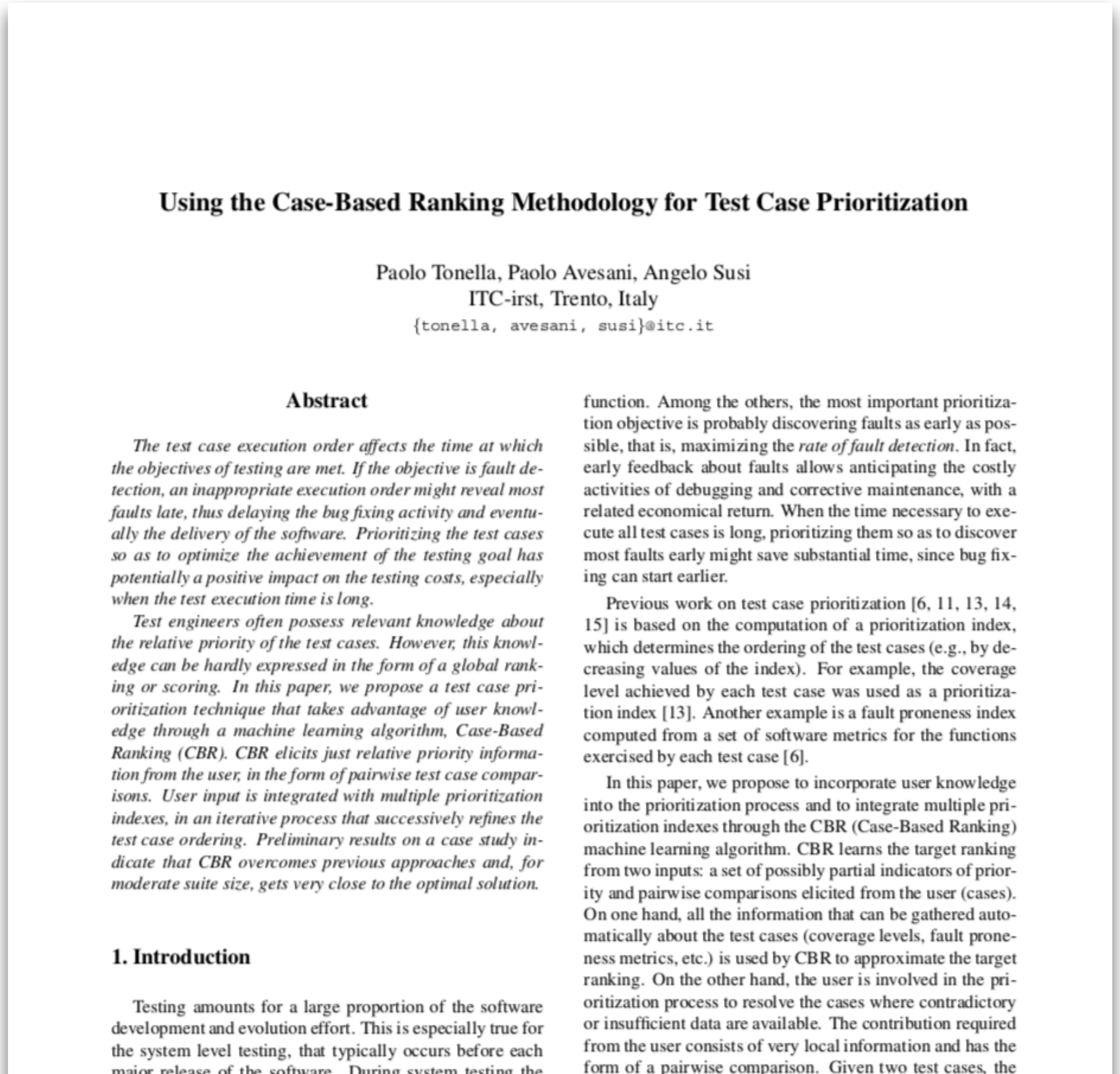
Test Case Prioritisation



D. Leon and A. Podgurski. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE 2003), pages 442–456. IEEE Computer Society Press, November 2003.

Case-Based Reasoning

- P. Tonella, P. Avesani, and A. Susi. Using the case-based ranking methodology for test case prioritization, ICSME 2006
- Human testers make pairwise comparison between test cases
- CBR learns to put priority scores to test cases, based on human examples
- Effective, but human comparison is extremely expensive



Using the Case-Based Ranking Methodology for Test Case Prioritization

Paolo Tonella, Paolo Avesani, Angelo Susi
ITC-irst, Trento, Italy
{tonella, avesani, susi}@itc.it

Abstract

The test case execution order affects the time at which the objectives of testing are met. If the objective is fault detection, an inappropriate execution order might reveal most faults late, thus delaying the bug fixing activity and eventually the delivery of the software. Prioritizing the test cases so as to optimize the achievement of the testing goal has potentially a positive impact on the testing costs, especially when the test execution time is long.

Test engineers often possess relevant knowledge about the relative priority of the test cases. However, this knowledge can be hardly expressed in the form of a global ranking or scoring. In this paper, we propose a test case prioritization technique that takes advantage of user knowledge through a machine learning algorithm, Case-Based Ranking (CBR). CBR elicits just relative priority information from the user, in the form of pairwise test case comparisons. User input is integrated with multiple prioritization indexes, in an iterative process that successively refines the test case ordering. Preliminary results on a case study indicate that CBR overcomes previous approaches and, for moderate suite size, gets very close to the optimal solution.

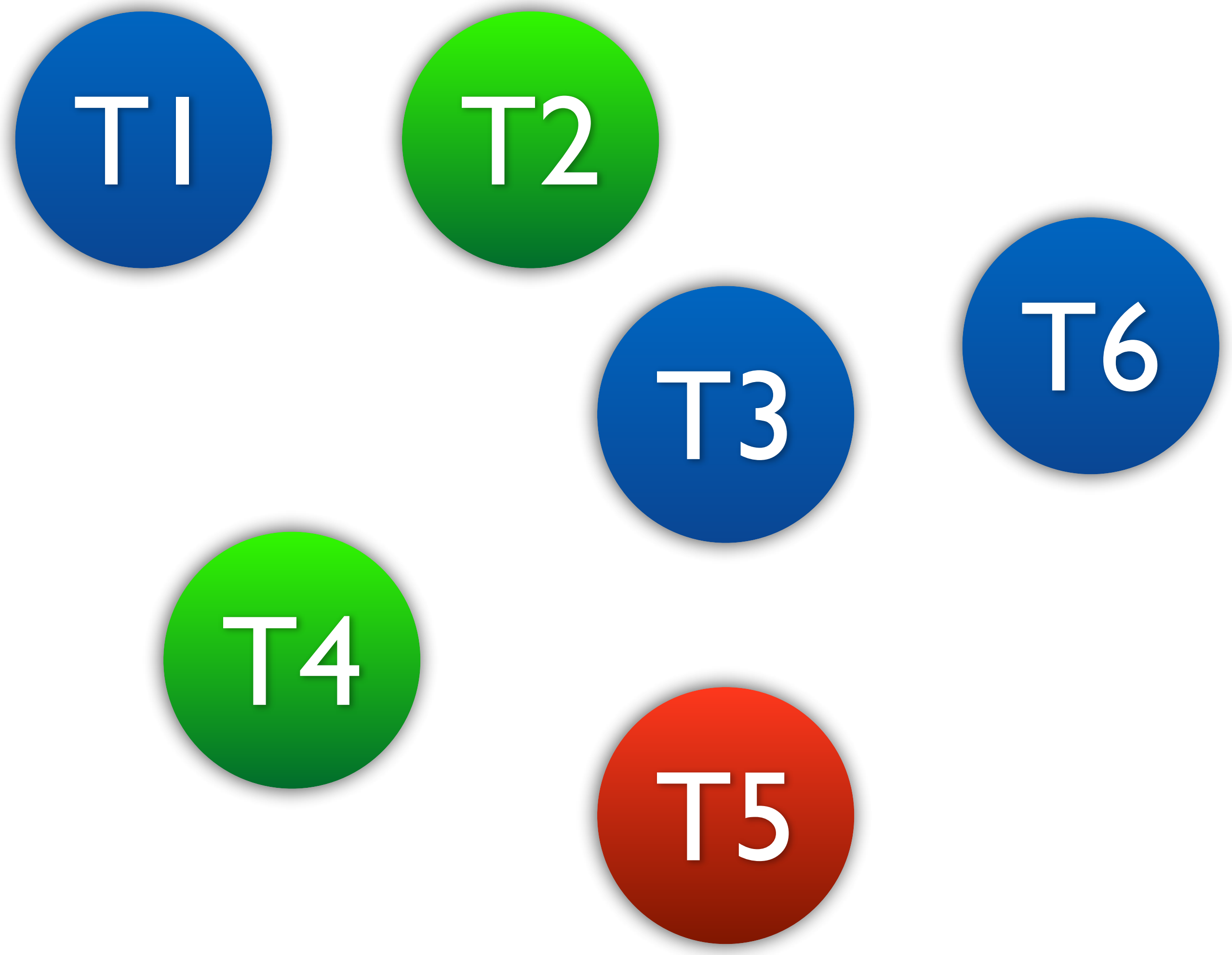
1. Introduction

Testing amounts for a large proportion of the software development and evolution effort. This is especially true for the system level testing, that typically occurs before each major release of the software. During system testing the function. Among the others, the most important prioritization objective is probably discovering faults as early as possible, that is, maximizing the *rate of fault detection*. In fact, early feedback about faults allows anticipating the costly activities of debugging and corrective maintenance, with a related economical return. When the time necessary to execute all test cases is long, prioritizing them so as to discover most faults early might save substantial time, since bug fixing can start earlier.

Previous work on test case prioritization [6, 11, 13, 14, 15] is based on the computation of a prioritization index, which determines the ordering of the test cases (e.g., by decreasing values of the index). For example, the coverage level achieved by each test case was used as a prioritization index [13]. Another example is a fault proneness index computed from a set of software metrics for the functions exercised by each test case [6].

In this paper, we propose to incorporate user knowledge into the prioritization process and to integrate multiple prioritization indexes through the CBR (Case-Based Ranking) machine learning algorithm. CBR learns the target ranking from two inputs: a set of possibly partial indicators of priority and pairwise comparisons elicited from the user (cases). On one hand, all the information that can be gathered automatically about the test cases (coverage levels, fault proneness metrics, etc.) is used by CBR to approximate the target ranking. On the other hand, the user is involved in the prioritization process to resolve the cases where contradictory or insufficient data are available. The contribution required from the user consists of very local information and has the form of a pairwise comparison. Given two test cases, the

Interleaved Clusters Prioritisation



Cluster

Intra-cluster Prioritisation

Inter-cluster Prioritisation

Interleaving Clusters

S. Yoo, M. Harman, P. Tonella, and A. Susi. Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge. In Proceedings of International Symposium on Software Testing and Analysis, ISSTA 2009, pages 201–211. ACM Press, July 2009.

Classification/Prediction

- To identify to which of a set of categories a new example belongs
  - Defect Prediction / Fault Localisation: Is this statement/method/file (likely to be) faulty or not?
  - Hypotheses
    - “If a file goes through an unusually high number of changes, it is more likely to be faulty”
    - “If a file is modified by an unusually high number of developers, it is more likely to be faulty”

Defect Prediction

- Collect past test history as well as various features leading up to the test results
- Train a classification model
- Before a large project moves into the testing stage, feed the collected data to see which file is more likely to be faulty

Table 4. List of Change metrics used in the study.

Metric name	Definition
REVISIONS	Number of revisions of a file
REFACTORINGS	Number of times a file has been refactored[1]
BUGFIXES	Number of times a file was involved in bug-fixing[2]
AUTHORS	Number of distinct authors that checked a file into the repository
LOC_ADDED	Sum over all revisions of the lines of code added to a file
MAX_ LOC_ADDED	Maximum number of lines of code added for all revisions
AVE_ LOC_ADDED	Average lines of code added per revision
LOC_DELETED	Sum over all revisions of the lines of code deleted from a file
MAX_ LOC_DELETED	Maximum number of lines of code deleted for all revisions
AVE_ LOC_DELETED	Average lines of code deleted per revision
CODECHURN	Sum of (added lines of code – deleted lines of code) over all revisions
MAX_ CODECHURN	Maximum CODECHURN for all revisions
AVE_ CODECHURN	Average CODECHURN per revision
MAX_CHANGESET	Maximum number of files committed together to the repository
AVE_CHANGESET	Average number of files committed together to the repository
AGE	Age of a file in weeks (counting backwards from a specific release)
WEIGHTED_AGE	See equation (1)

R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In 2008 ACM/IEEE 30th International Conference on Software Engineering, pages 181–190, May 2008.

Information Retrieval

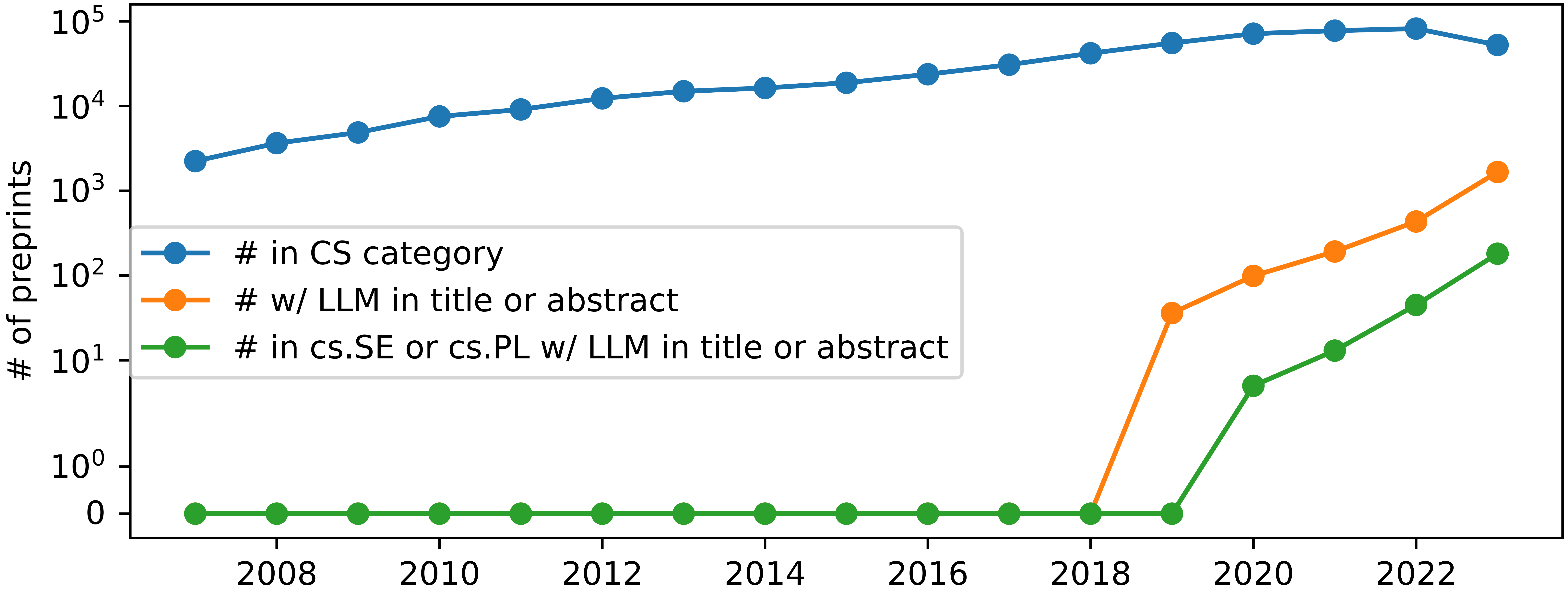
- IR is also used to perform fault localisation
- Given a bug report, the program element responsible for the observed failure is the part of the source code that is lexically the most similar to the bug report
- See for example: R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry. Improving bug localization using structured information retrieval. In Automated Software Engineering (ASE), 2013.

Recommendation System

- You buy X from Amazon, and give it five star. Amazon gives you “people who bought (and liked) this also bought…”
- Similarly, we can think of bug-triaging (i.e., the question of who should handle the new bug report) as:
  - You fix bug X from Project Z, and does the job well. The project gives you “people who successfully fixed bugs like this may also do well on…”

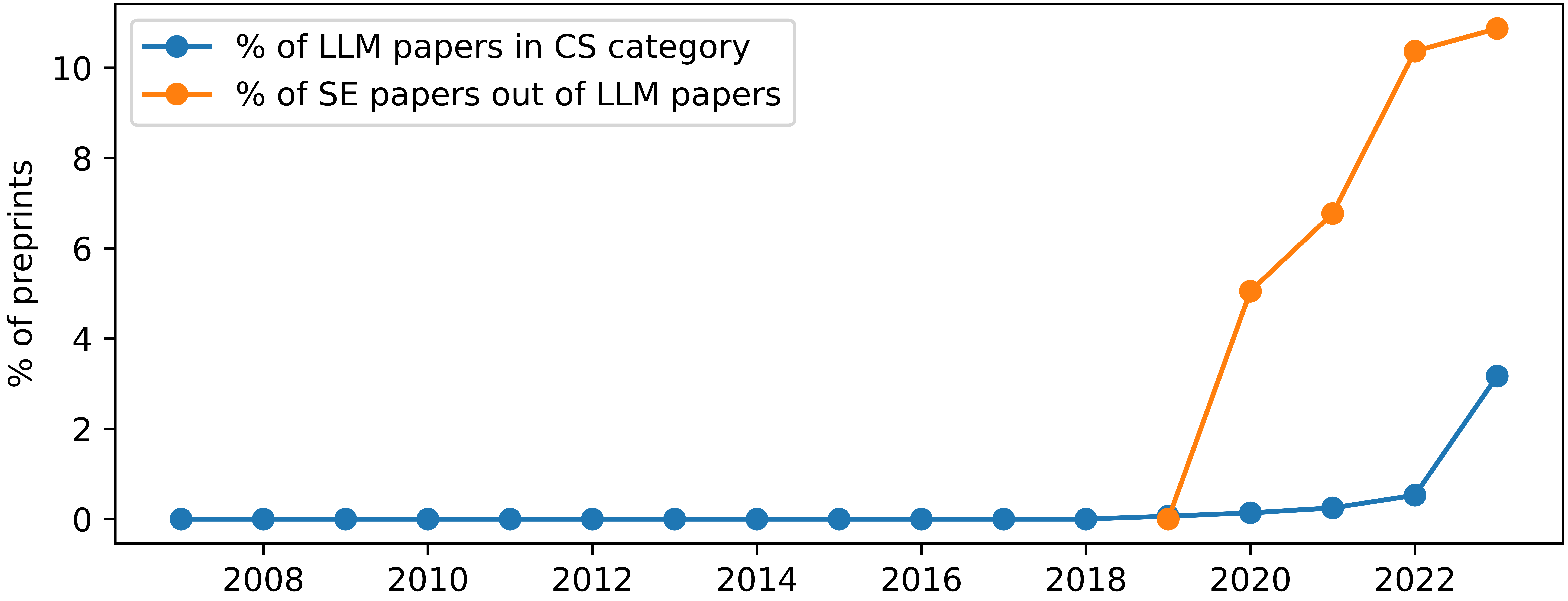
There is a new bandwagon in town: LLMs

(and we are expanding into this, to live up to the course name :p)



There is a new bandwagon in town: LLMs

(and we are expanding into this, to live up to the course name :p)



Initial Surveys

- Large Language Models for Software Engineering: Survey and Open Problemsm, Fan et al., (https://arxiv.org/abs/2310.03533) <— yours truly :)
- Large Language Models for Software Engineering: A Systematic Literature Review, Hou et al., (https://arxiv.org/abs/2308.10620)
- Software Testing with Large Language Model: Survey, Landscape, and Vision, Wang et al., (https://arxiv.org/abs/2307.07221)

Emerging Topics

- Code synthesis is a primary application, as LLMs can generate code
- Automating testing using LLMs is also big, as writing test code is often perceived as boring and repetitive
- There are other sub-areas of software engineering where natural language processing power appears to be critically important, but remain relatively unexplored
  - Requirement Engineering

A Million Dollar Question

- How do we validate the LLM outputs?
  - So, it is testing again, after all.
  - Bright side: we have been doing automated testing so long, we should know how to do this…
  - Down side: oracle still depends on humans…
- More details when we reach the LLM lecture.