School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Automated Bug Localization and Repair

David Lo
School of Information Systems
Singapore Management University
davidlo@smu.edu.sg

Invited Talk, ISHCS 2016, China

A Brief Self-Introduction

Singapore Management University

- Singapore 3rd uni.
- Number of students:
  - 7000+ (UG)
  - 1000+ (PG)
- Schools:
  - Information Systems
  - Economics
  - Law
  - Business
  - Accountancy
  - Social Science

A Brief Self-Introduction

SOAR

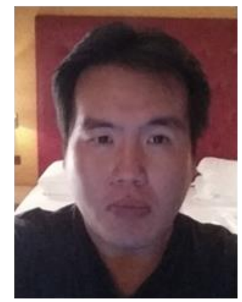
SOFTWARE ANALYTICS RESEARCH GROUP

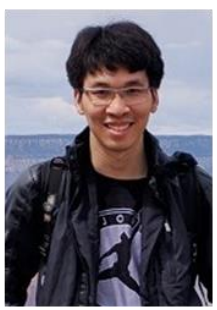
https://soarsmu.github.io/

@soarsmu

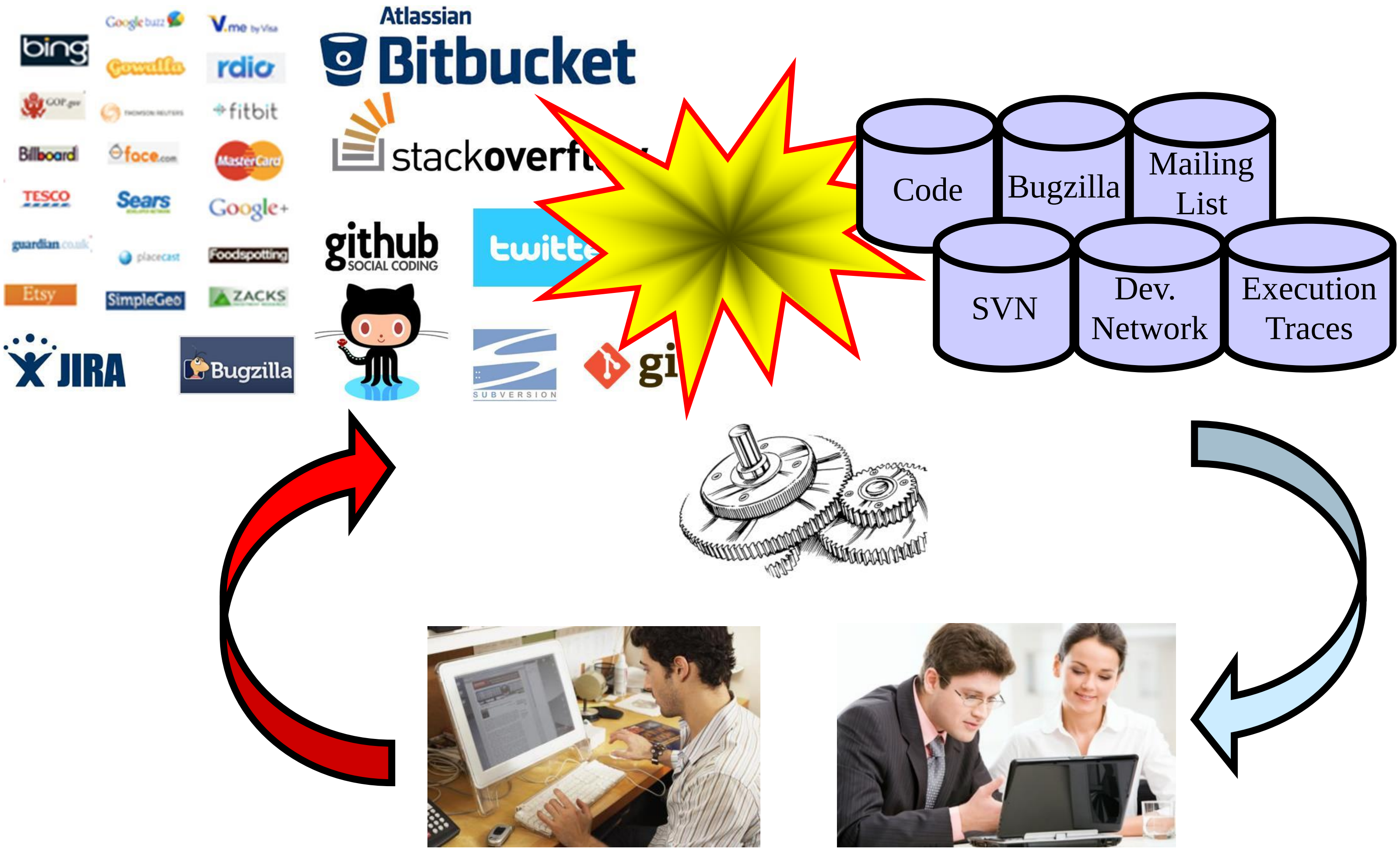
A Brief Self-Introduction

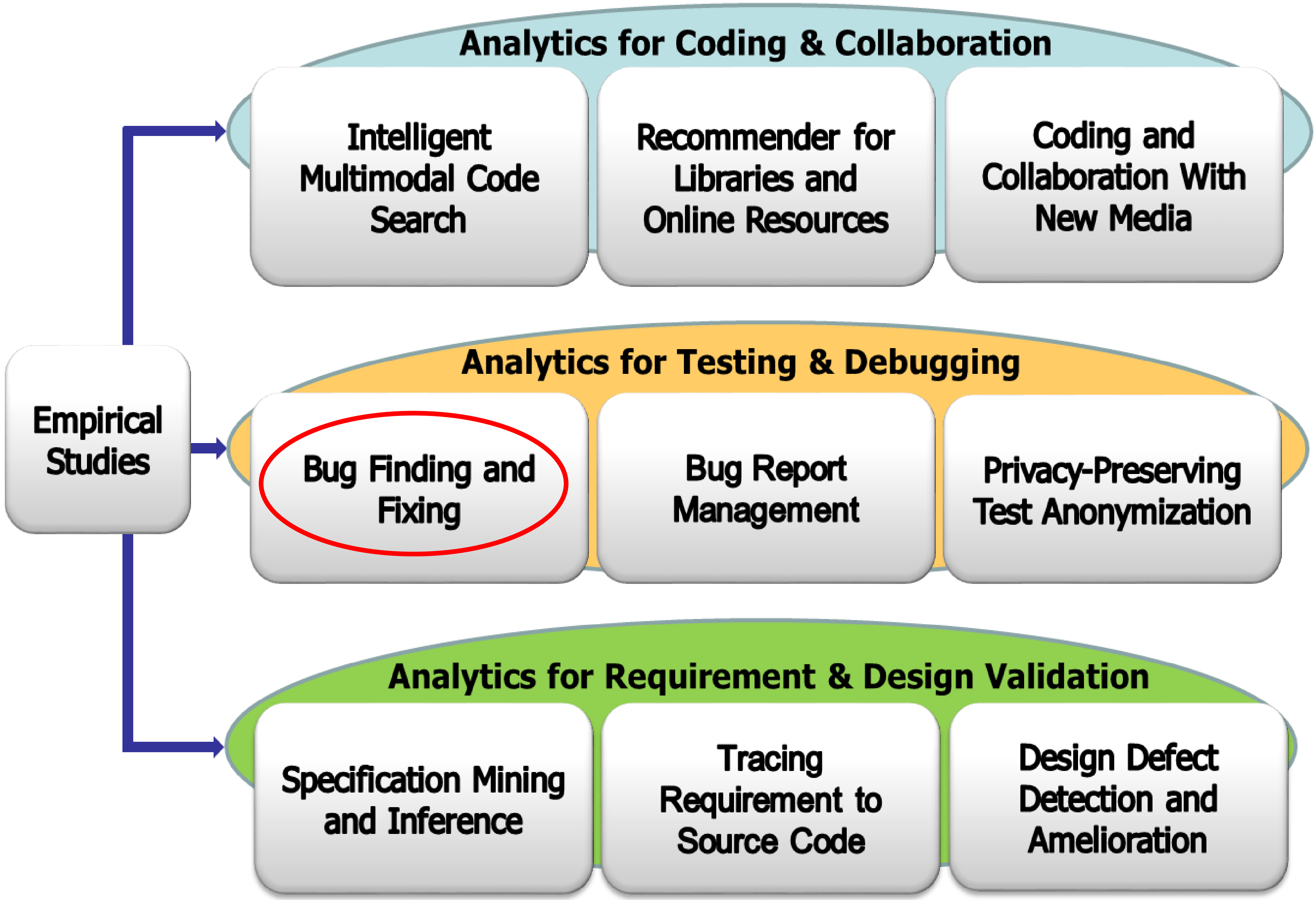
     

A Brief Self-Introduction




Analytics for Coding & Collaboration
Intelligent Multimodal Code Search
Recommender for Libraries and Online Resources
Coding and Collaboration With New Media
Empirical Studies
Analytics for Testing & Debugging
Bug Finding and Fixing
Bug Report Management
Privacy-Preserving Test Anonymization
Analytics for Requirement & Design Validation
Specification Mining and Inference
Tracing Requirement to Source Code
Design Defect Detection and Amelioration

Motivation

- Software bugs cost the U.S. economy 59.5 billion dollars annually (Tassey, 2002)
- Software debugging is an expensive and time consuming task in software projects
  - Testing and debugging account 30-90% of the labor expended on a project (Beizer, 1990)

Debugging

"Identify and remove error from (computer hardware or software)" – Oxford Dictionary

Buggy Code Identification
(aka. Bug/Fault Localization)

Program Repair

School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Information Retrieval and Spectrum Based Bug Localization: Better Together

Tien-Duy B. Le, Richard J. Oentaryo, and David Lo

School of Information Systems

Singapore Management University

10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering (*ESEC-FSE 2015*), Bergamo, Italy

IR-Based Bug Localization

Bug Report

IR-Based Bug Localization Technique

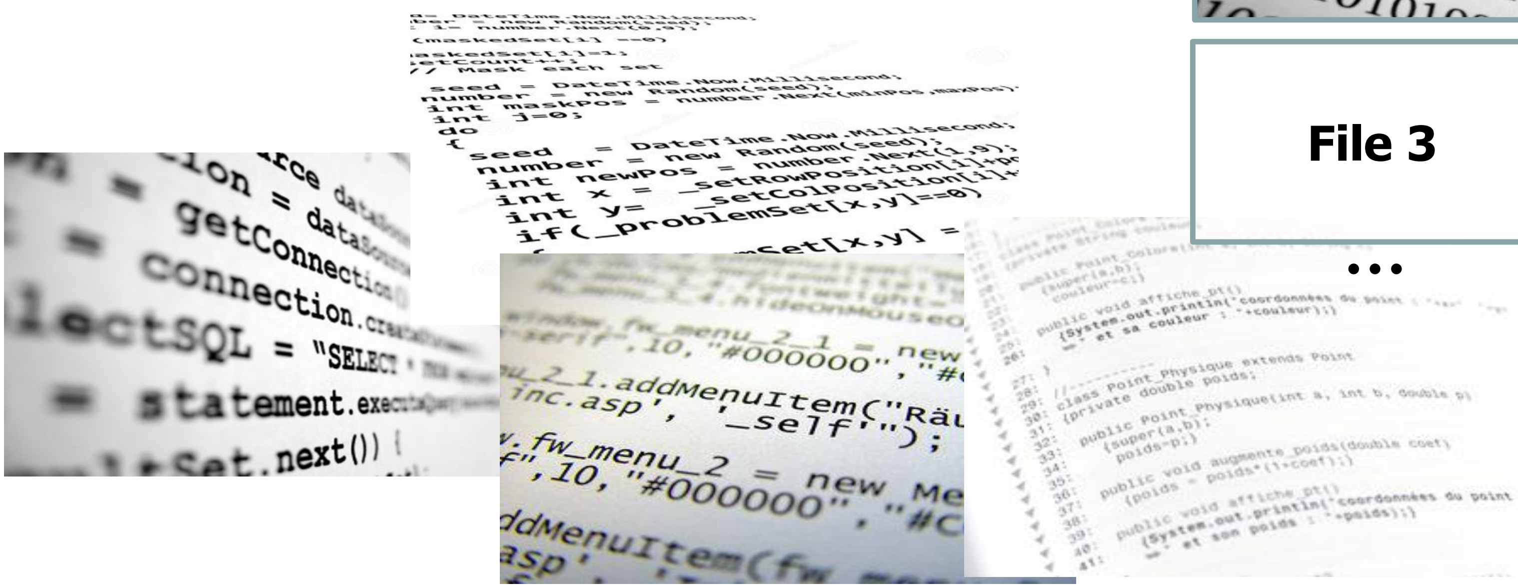
Ranked List of Files

File 1

File 2

File 3

…

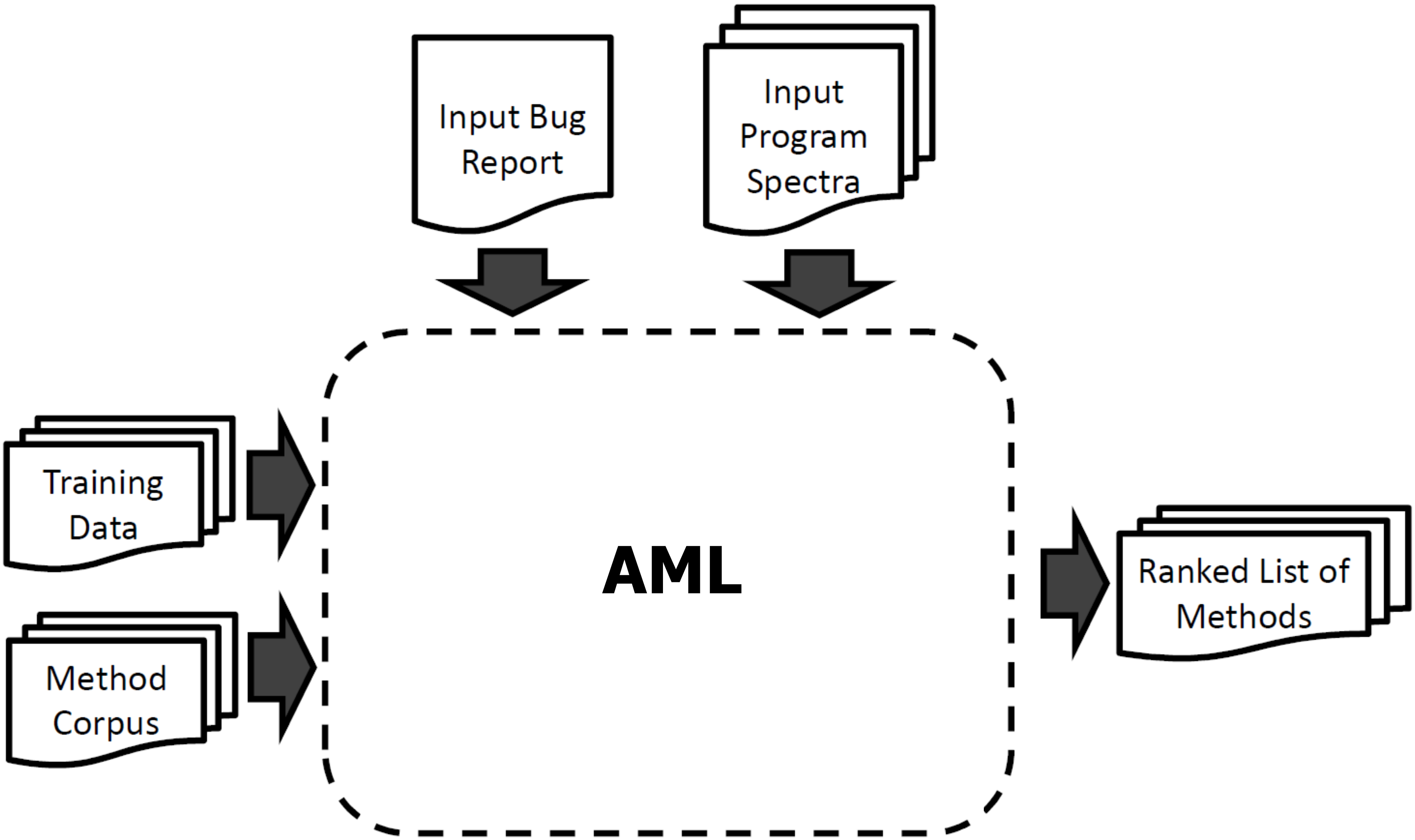


(Thousands of) Source Code Files

Spectrum-Based Bug Localization

Block ID	Program Elements	T15	T16	T17	T18
1	int count; int n; Ele *proc; List *src_queue, *dest_queue; if (prio >= MAXPRIO) /*maxprio=3*/	●	●	●	●
2	{return;}		●	●	●
3	src_queue = prio_queue[prio]; dest_queue = prio_queue[prio+1]; count = src_queue->mem_count; *if (count > 1) /* Bug*//* expected : count>0*/ {*	●	●	●	●
4	n = (int) (count*ratio + 1); proc = find_nth(src_queue, n); if (proc) {		●	●	
5	src_queue = del_ele(src_queue, proc); proc->priority = prio; dest_queue = append_ele(dest_queue, proc); } }}		●	●	
	Status of Test Case Execution :	Pass	Pass	Pass	Fail

AML: Adaptive Multi-Modal Bug Localization



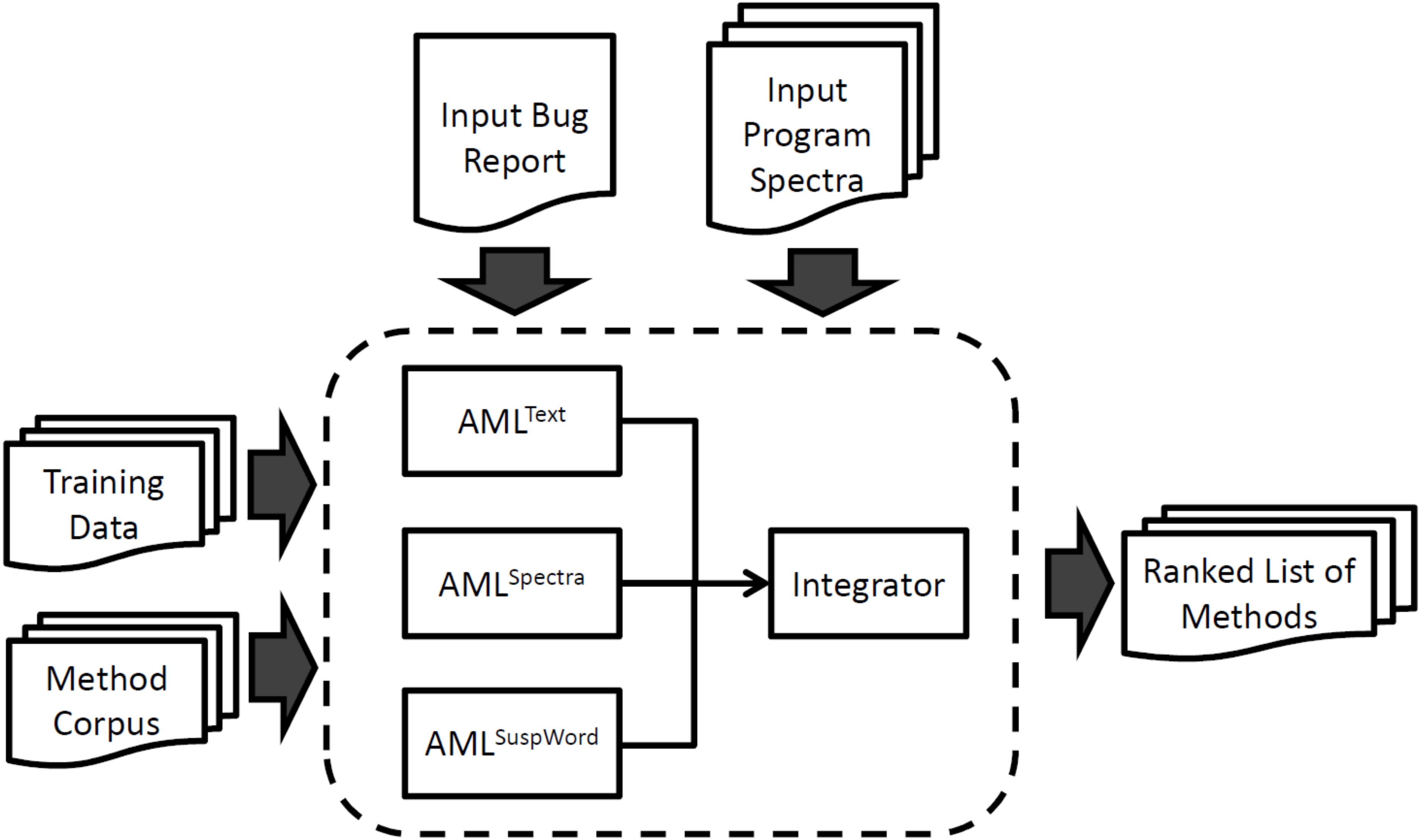
AML: Main Features

- *Adaptive* Bug Localization
  - Instance-specific vs. one-size-fits-all
  - Each bug is considered individually
  - Various parameters are tuned *adaptively*
    - Based on individual characteristics

AML: Main Features

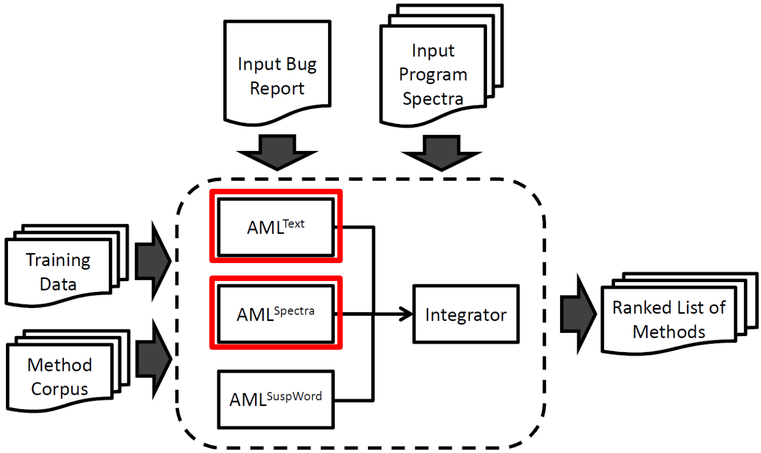
- New *word weighting scheme*
  - Based on *suspiciousness* inferred from spectra
  - Nicely integrates bug reports + spectra
  - "future research … automatically highlight terms … related to a failure" (Parnin and Orso, 2011)

AML: Adaptive Multi-Modal Bug Localization



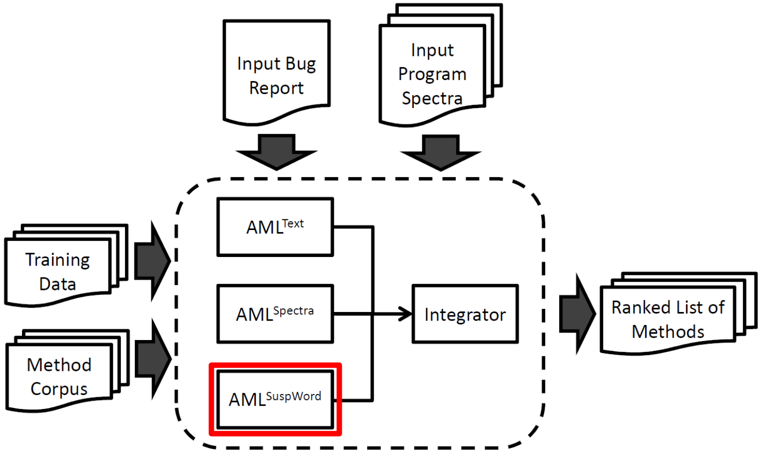
AML^{Text} and AML^{Spectra}

- AML^{Text}: use standard IR-based bug localization technique
  - Use VSM
- AML^{Spectra}: use standard spectrum-based bug localization technique
  - Use Tarantula



AML^{SuspWord} - Intuition

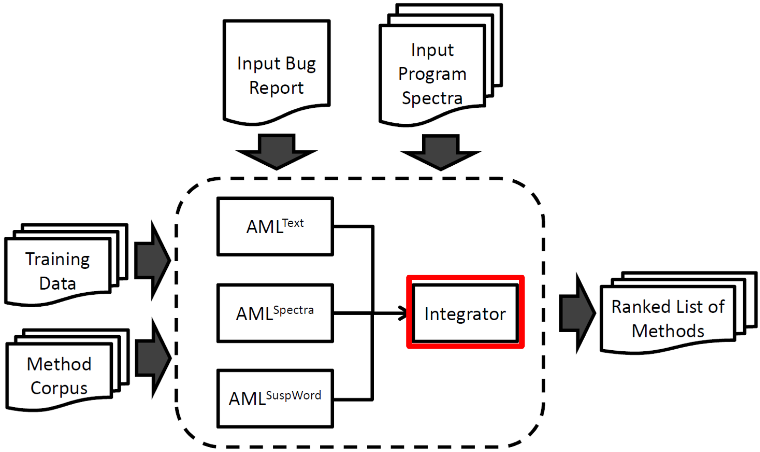
- Word suspiciousness
  - For a bug, some words (in bug reports and files) are more *suspicious* (indicative of the bug)
  - Computed from program spectra
- Method suspiciousness is inferred from those of its constituent words



Integrator

$$f(x_i, \theta) = \alpha \times \mathrm{AML}^{\mathrm{Text}}(b, m) + \beta \times \mathrm{AML}^{\mathrm{Spectra}}(p, m) + \gamma \times \mathrm{AML}^{\mathrm{SuspWord}}(b, p, m)$$

- Three parameters are tuned adaptively
  - Find the most similar *k* historical fixed reports
  - Find a near-optimal set of parameter values
    - Optimize performance for the *k* reports



Dataset

Project	#Bugs	Time Period	Average # Methods
AspectJ	41	03/2005 – 02/2007	14,218.39
Ant	53	12/2001 – 09/2013	9,624.66
Lucene	37	06/2006 – 01/2011	10,220.14
Rhino	26	12/2007 – 12/2011	4,839.58

Baselines

- LR^A, LR^B (Ye et al., FSE'14)
- MULTRIC (Xuan and Monperrus, ICSME'14)
- PROMESIR (Poshyvanyk et al., TSE'07)
- DIT^A, DIT^B (Dit et al., EMSE'13)

Evaluation Metrics

- **Top N**: Number of bugs whose buggy methods are successfully localized at top-N positions
- **MAP** (Mean Average Precision):

$$AP = \sum_{k=1}^{M} \frac{P(k) \times pos(k)}{number\ of\ buggy\ methods}$$

$$P(k) = \frac{\#faulty\ methods\ in\ the\ top\ k}{k}$$

Top-N Scores

Top	Project	AML	P	D^A	D^B	L^A	L^B	M
1	AspectJ	7	4	4	3	0	0	0
	Ant	9	7	3	3	1	11	2
	Lucene	11	8	7	7	1	7	4
	Rhino							2
	Over							8
5	Aspec							1
	Ant							7
	Lucen							13
	Rhino							8
	Over							29
10	Aspec							2
	Ant	31	28	20	20	19	32	15
	Lucene	29	21	20	19	10	24	16
	Rhino	19	14	7	7	3	12	11
	Overall	92	72	51	49	32	68	44

Locate **47.62%, 31.48%, and 27.78%** more bugs than the best performing baseline at top-1, 5, and 10 positions.

MAP Scores

Project	AML	P	D^A	D^B	L^A	L^B	M
AspectJ	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.016
Ant	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.077
Lucene	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.188
Rhino	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.172
Overall	0.237	0.184	0.118	0.112	0.043	0.127	0.113

Improve MAP by at least **28.80%**.

Takeaway

- Multiple data sources can be leveraged to locate buggy code
  - Bug reports
  - Execution traces
- IR-based and spectrum-based bug localization can be merged together to boost effectiveness
- An adaptive solution that tunes itself given a target bug to locate can outperform a one-size-fits all solution

Debugging

"Identify and remove error from (computer hardware of software)" – Oxford Dictionary

Buggy Code Identification
(aka. Bug/Fault Localization)

Program Repair

School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

History Driven Program Repair

Xuan-Bach D. Le[1], David Lo[1], and Claire Le Goues[2]

[1]Singapore Management University

[2]Carnegie Mellon University

23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016), Osaka, Japan

Program Repair Tools

Test Cases

Mutates buggy program
to create repair candidates

Candidate passing
all test cases

E.g., GenProg, PAR, etc

Issues of Existing Repair Tools

- Test-driven approaches: overfitting, nonsensical patches

```
// Human fix: fa * fb > 0
If (fa * fb >= 0){
   throw new ConvergenceException("..");
}
```

- Long computation time to produce patches
- Lack of knowledge on bug fix history
  - PAR: **manually learned** fix patterns

History Driven Program Repair

Test Cases

Mutates buggy program to create repair candidates

Candidates:
- frequently occur in the knowledge base
- pass negative tests

Knowledge base: Learned bug fix behaviors from history

Fast

Avoid nonsensical patches

Our Framework (HDRepair)

Phase I: Bug Fix History Extraction

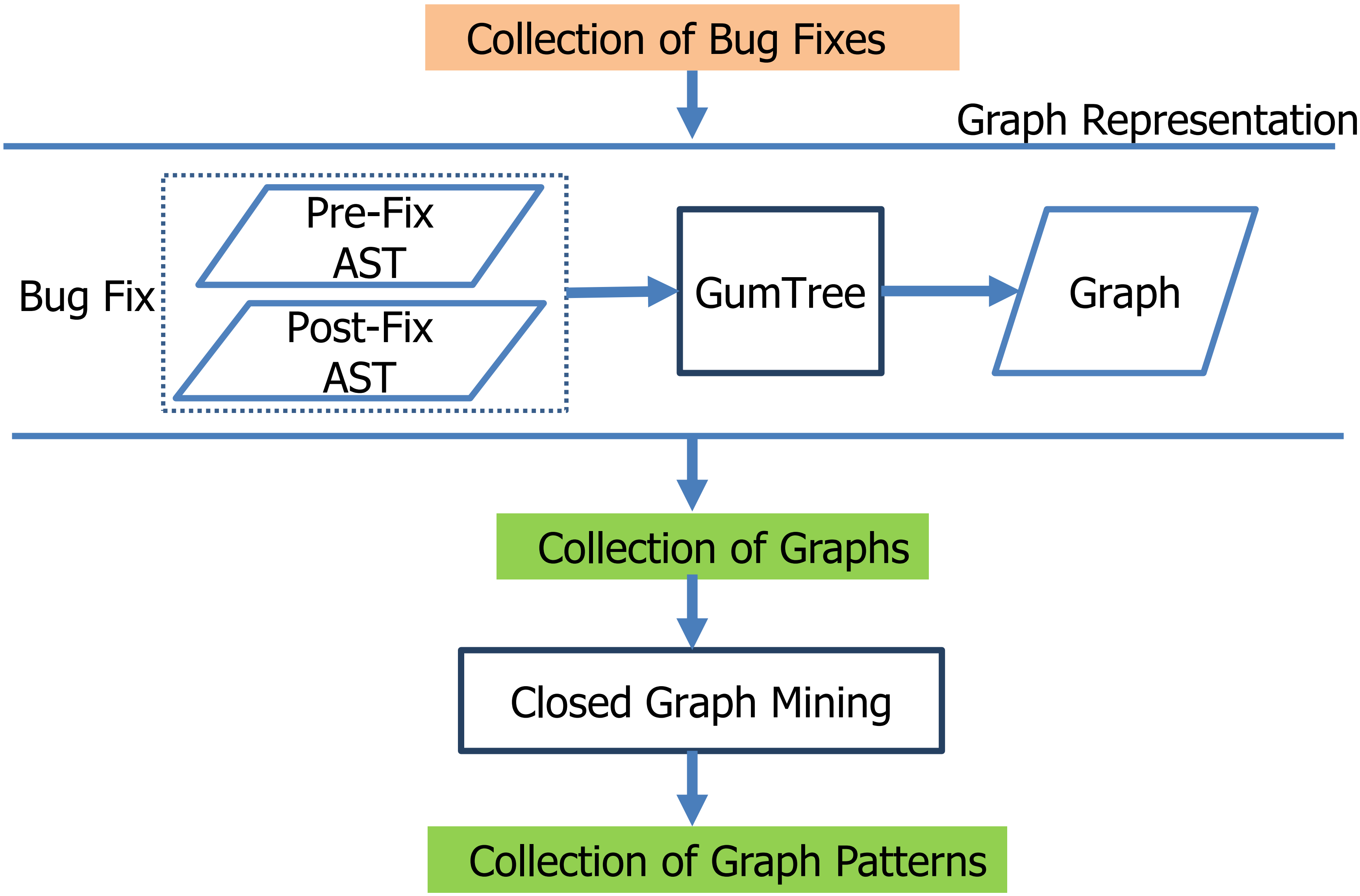
Phase II: Bug Fix History Mining

Phase III: Bug Fix Generation

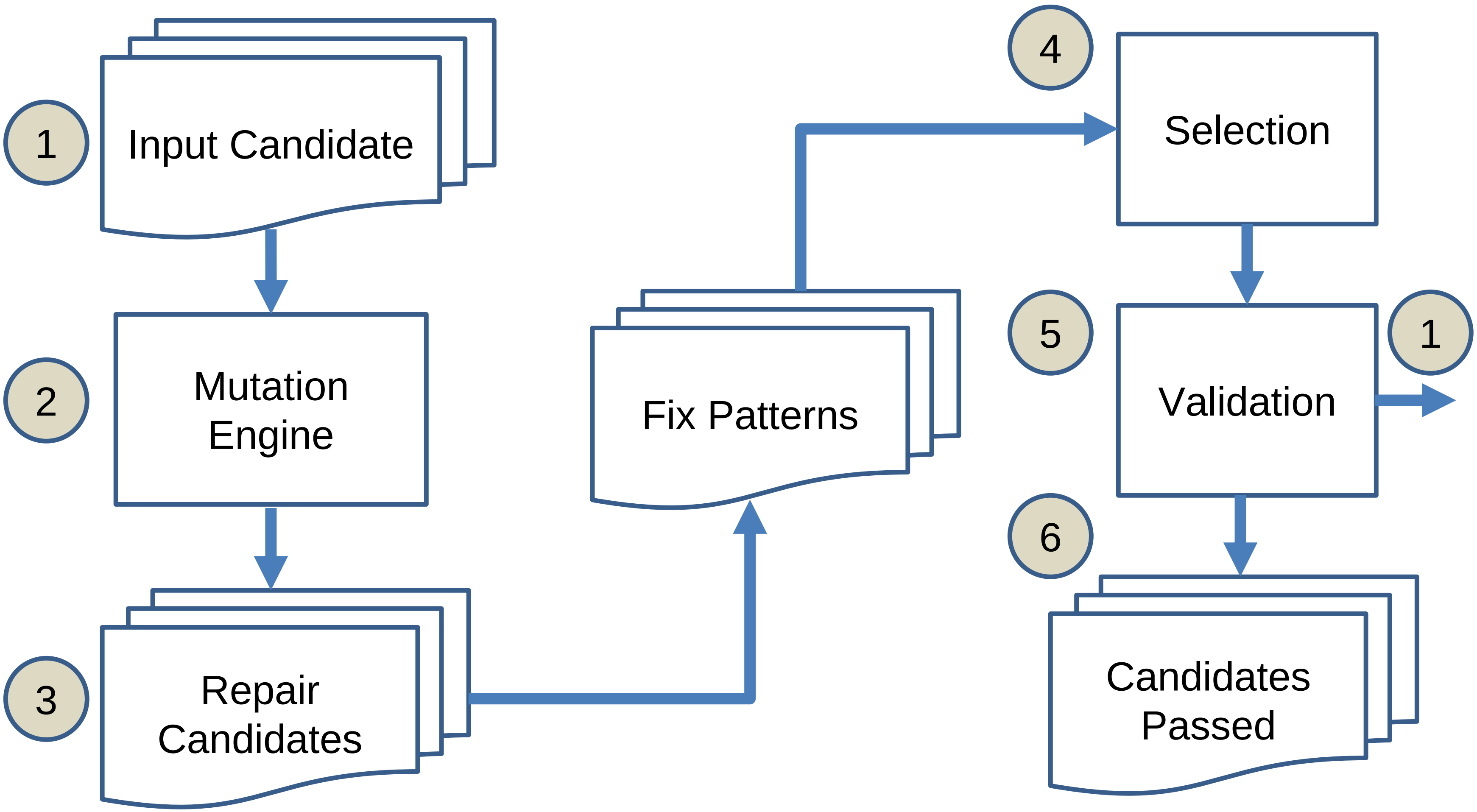
Phase I – Bug Fix History Extraction

- Active, large and popular Java projects
  - Updated until 2014, >= 5 stars, >= 100MBs
- Likely bug-fix commits
  - Commit message: *fix, bug fix, ~~fix typo~~, ~~fix build~~, ~~non fix~~*
  - Submission of at least one test case
  - Change no more than two source code lines
- Result: 3,000 bug fixes from 700+ projects

Phase II – Bug Fix History Mining



Phase III – Bug Fix Candidate Generation

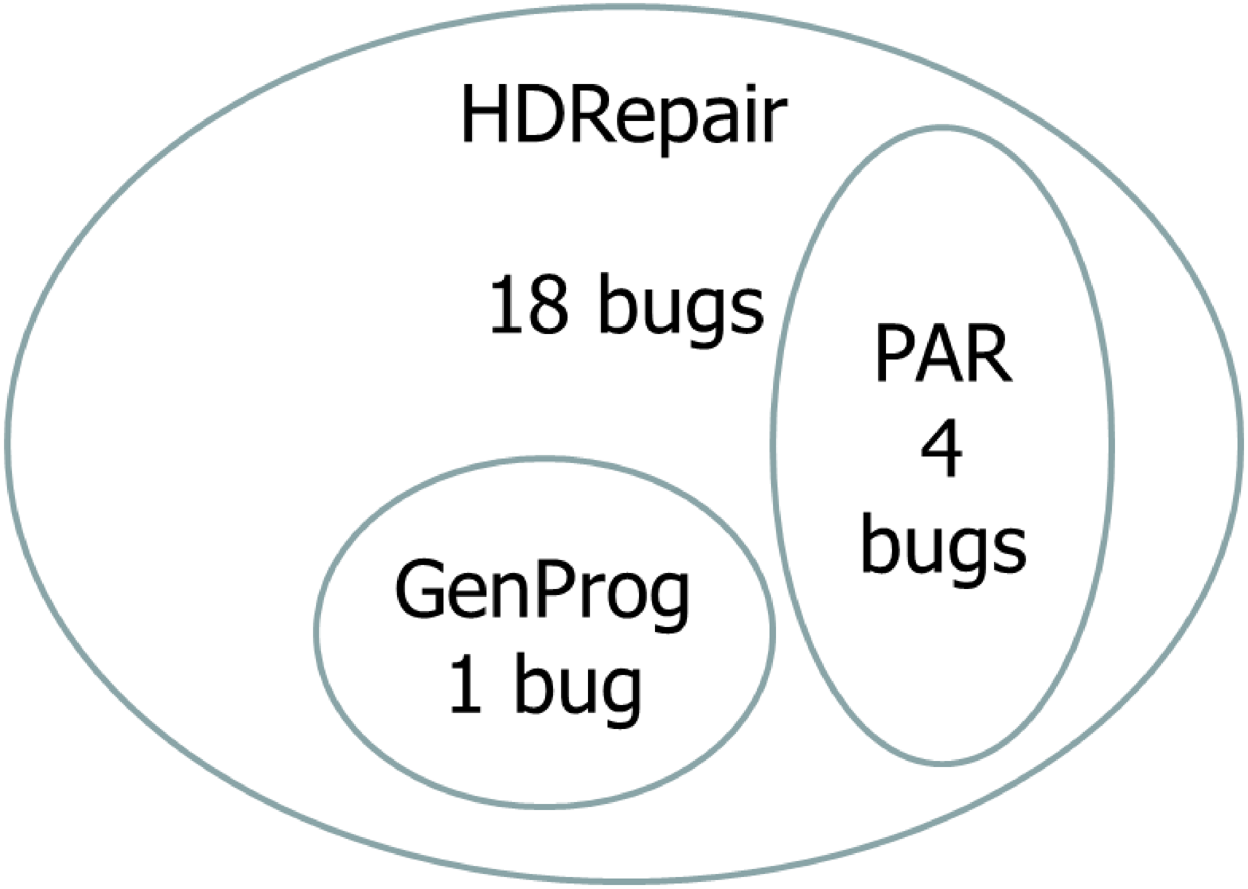


Experiment - Data

Program	#Bugs	#Bugs Exp
JFreeChart	26	5
Closure Compiler	133	29
Commons Math	106	36
Joda Time	27	2
Commons Lang	65	18
Total	**357**	**90**

Subset of Defects4J: bugs whose fixes involve fewer than 5 changed lines

Number of Bugs Correctly Fixed



Failure Cases

- **Plausible vs Correct Fixes**
  - Plausible fix passes all tests, but does not conform to certain desired behaviors

```
//Fix by human and our approach: change condition to fa * fb > 0.0
if (fa * fb >= 0.0) {
    //Plausible fix by GenProg
-   throw new ConvergenceException("...")
}
```

Failure Cases

- **Timeout**
  - PAR and GenProg both have operators but timeout

```
for(Node finallyNode : cfa.finallyMap.get(parent)){
-   cfa.createEdge(fromNode, Branch.UNCOND, finallyNode);
+   cfa.createEdge(fromNode, Branch.ON_EX, finallyNode);
}
```

School of Information Systems

SMU SINGAPORE MANAGEMENT UNIVERSITY

CDRep: Automatic Repair of Cryptographic Misuses in Android Applications

Siqi Ma[1], David Lo[1], Teng Li[2], Robert H. Deng[1]

[1]Singapore Management University, Singapore

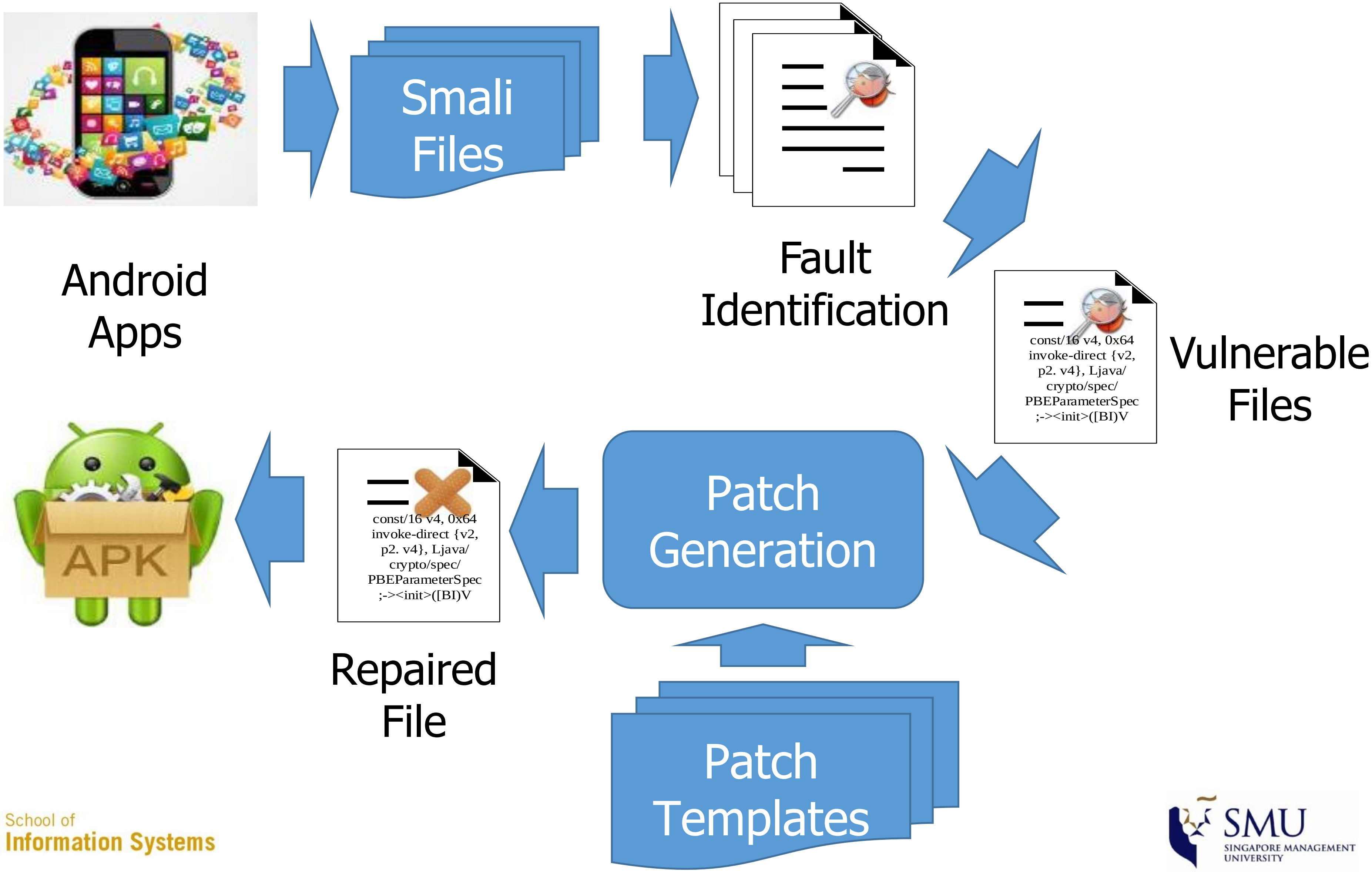
[2]Xidian University, China

11th ACM Symposium on Information, Computer and Communications Security (*AsiaCCS 2016*), Xian, China

What is a Cryptographic Misuse?

#	Cryptographic Misuse	Patch Scheme
1	ECB mode	CTR mode
2	A constant IV for CBC encryption	A randomized IV for CBC encryption
3	A constant secret key	A randomized secret key
4	A constant salt for PBE	A randomized salt for PBE
5	Iteration < 1,000 in PBE	Iterations = 1,000
6	A constant to seed SecureRandom	SecureRandom.nextBytes()
7	MD5 hash function	SHA-256 hash function

CDRep: How Does Our System Work?



Evaluation data

#	Misuse Type	# of Apps from Google Play	# of Apps from SlideMe	# of Apps
1	Use ECB mode	402	485	887
2	Use a constant IV for CBC encryption	379	600	979
3	Use a constant secret key	357	525	882
4	Use a constant salt for PBE	4	3	7
5	Set # iteration < 1,000	7	4	10
6	Use a constant to seed SecureRandom	17	218	235
7	Use MD5 hash function	1359	4224	5582

Evaluation Results – Success Rate

#	# of Apps	# of Selected Apps	Team Acceptance	# of Developer Response	Developer Acceptance
1	887	100	91 (91%)	21	13 (61.9%)
2	979	110	92 (83.6%)	16	10 (62.5%)
3	882	100	83 (83%)	23	18 (78.2%)
4	7	7	5 (71.4%)	3	2 (66.7%)
5	10	10	10 (100%)	4	4 (100%)
6	235	235	212 (90.2%)	20	15 (75%)
7	5582	700	700 (100%)	143	138 (96.5%)

Takeaway

- Various kinds of bugs, including security loopholes, can be automatically repaired
- A knowledge base can significantly boost the effectiveness of existing techniques
  - Built automatically by mining version control systems and bug tracking systems
  - Built manually by identifying a number of common cases
- Knowledge base can reduce the likelihood of constructing nonsensical patches

What's Needed For Practitioners' Adoption?

School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

Practitioners’ Expectations on Automated Fault Localization

Pavneet Singh Kochhar[1], Xin Xia[2], David Lo[1], Shanping Li[2]

[1]Singapore Management University

[2]Zhejiang University

25th ACM International Symposium on Software Testing and Analysis (ISSTA 2016), Saarbrucken, Germany

Practitioners Survey

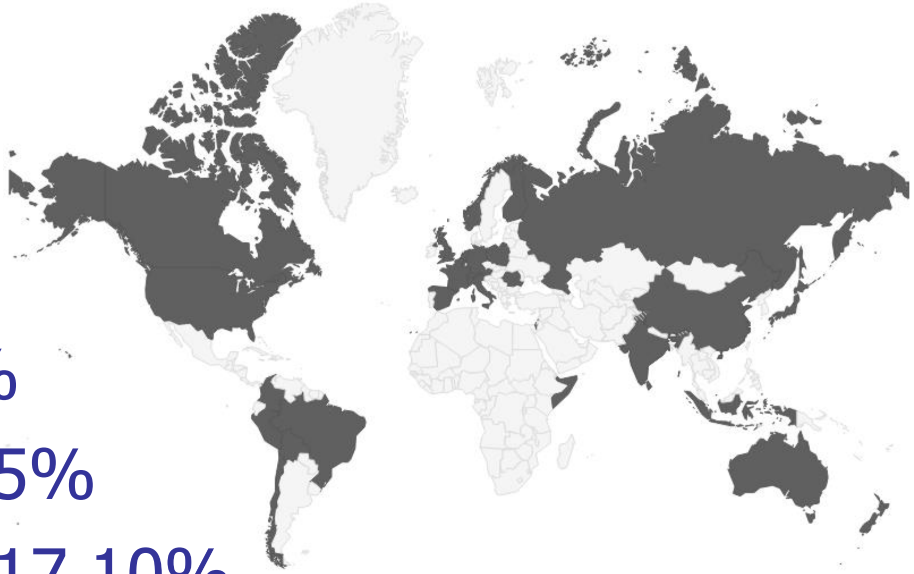
- Multi-pronged strategy:
  - Our contacts in IT industry
  - Email 3300 practitioners on github SOCIAL CODING
- We receive 386 responses

Survey Demographics

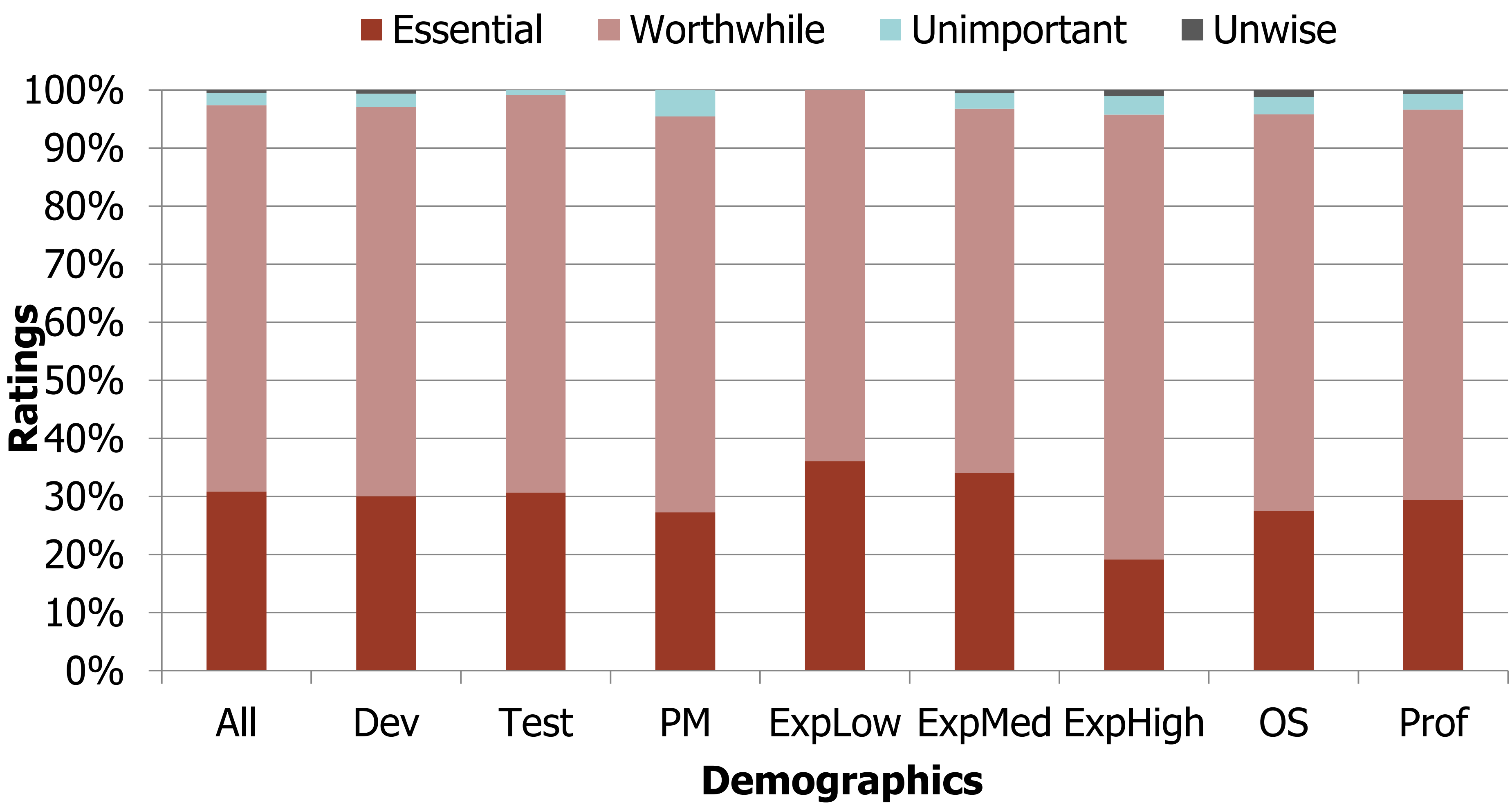
- 33 countries

- Job roles
  - Software dev. – 80.83%
  - Software testing – 30.05%
  - Project management – 17.10%

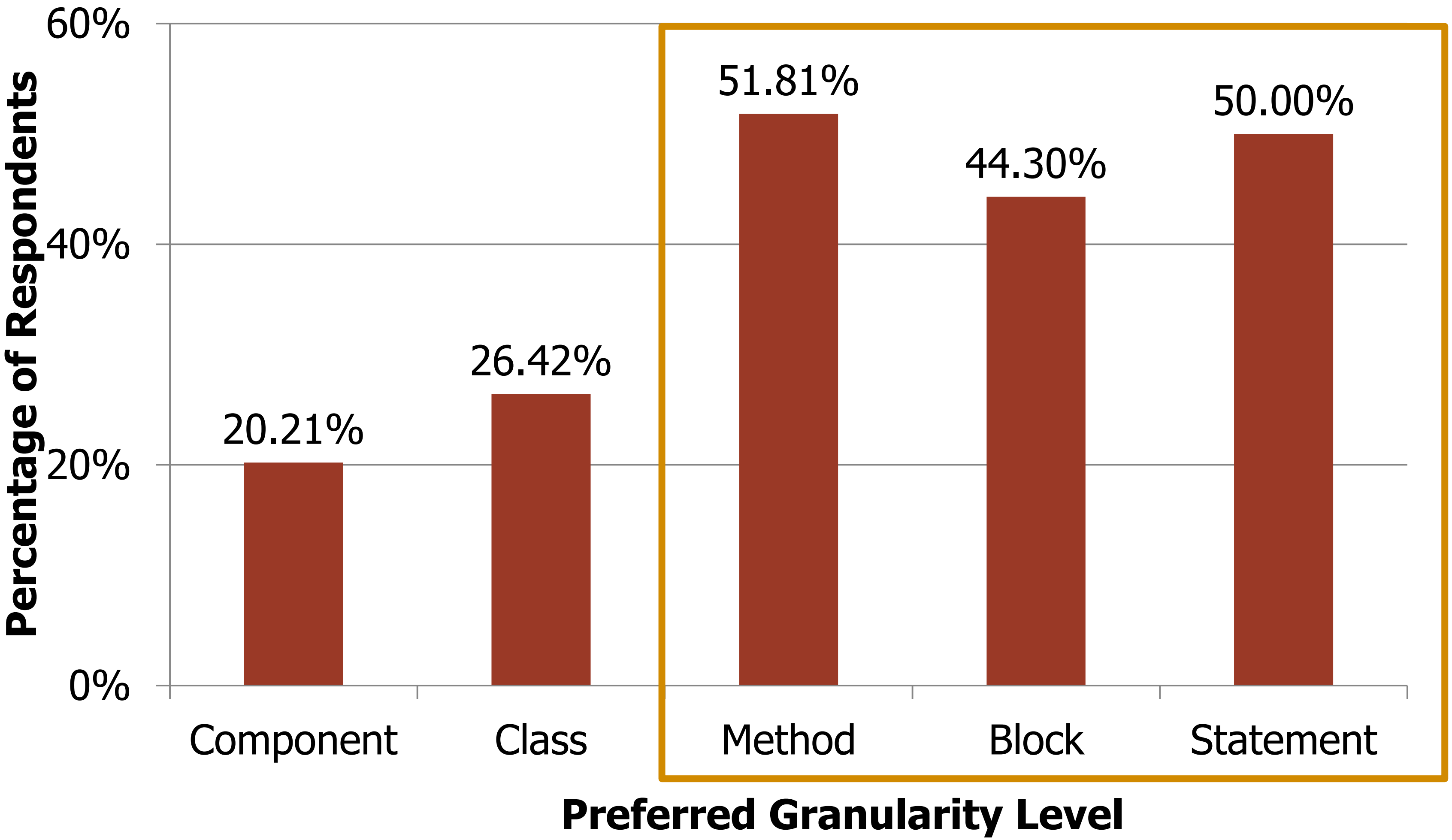
- Professional – 78.13%, Open-source – 44.24%



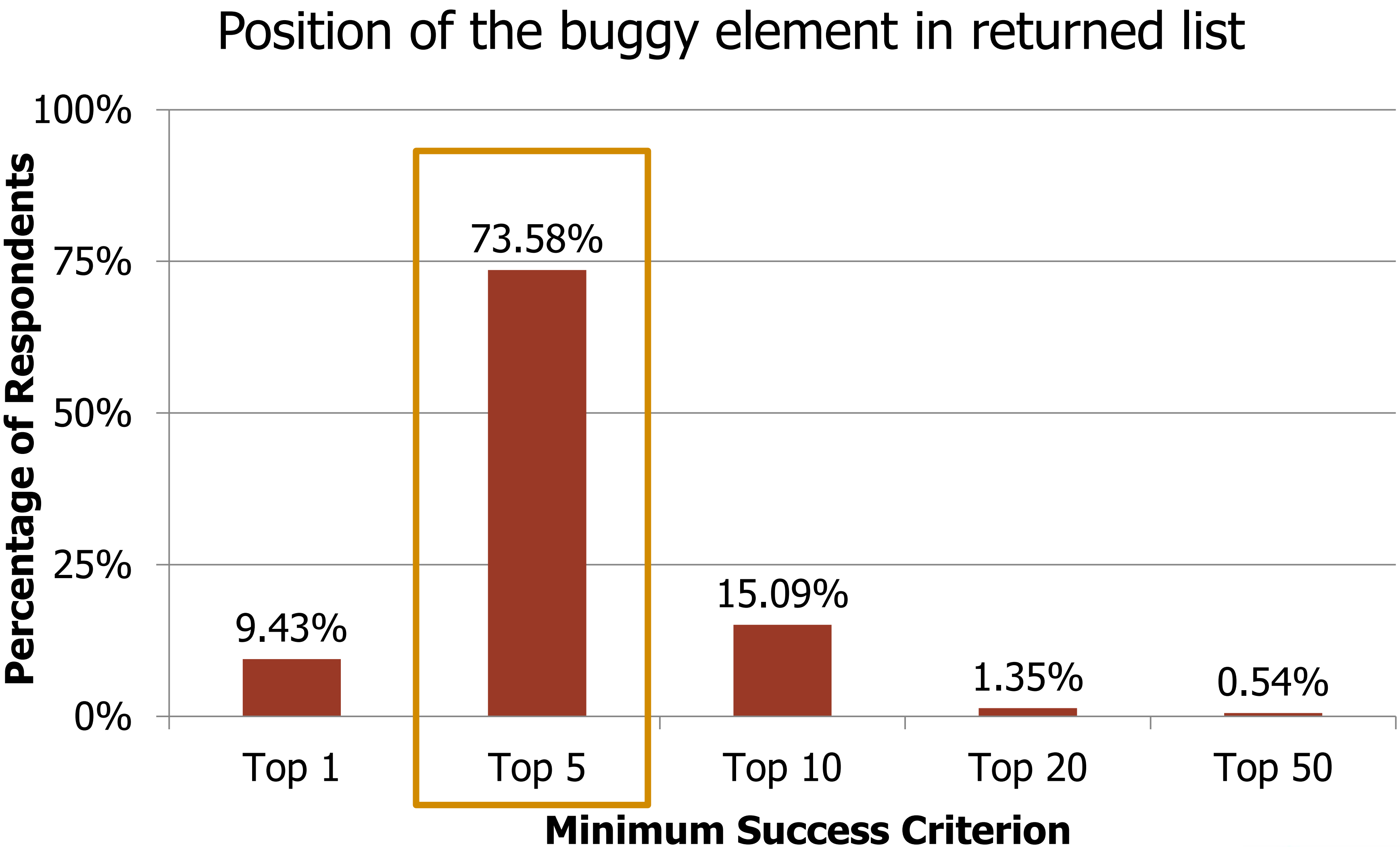
#1: Fault Localization Research is Valued



#2: Go for Finer Granularity



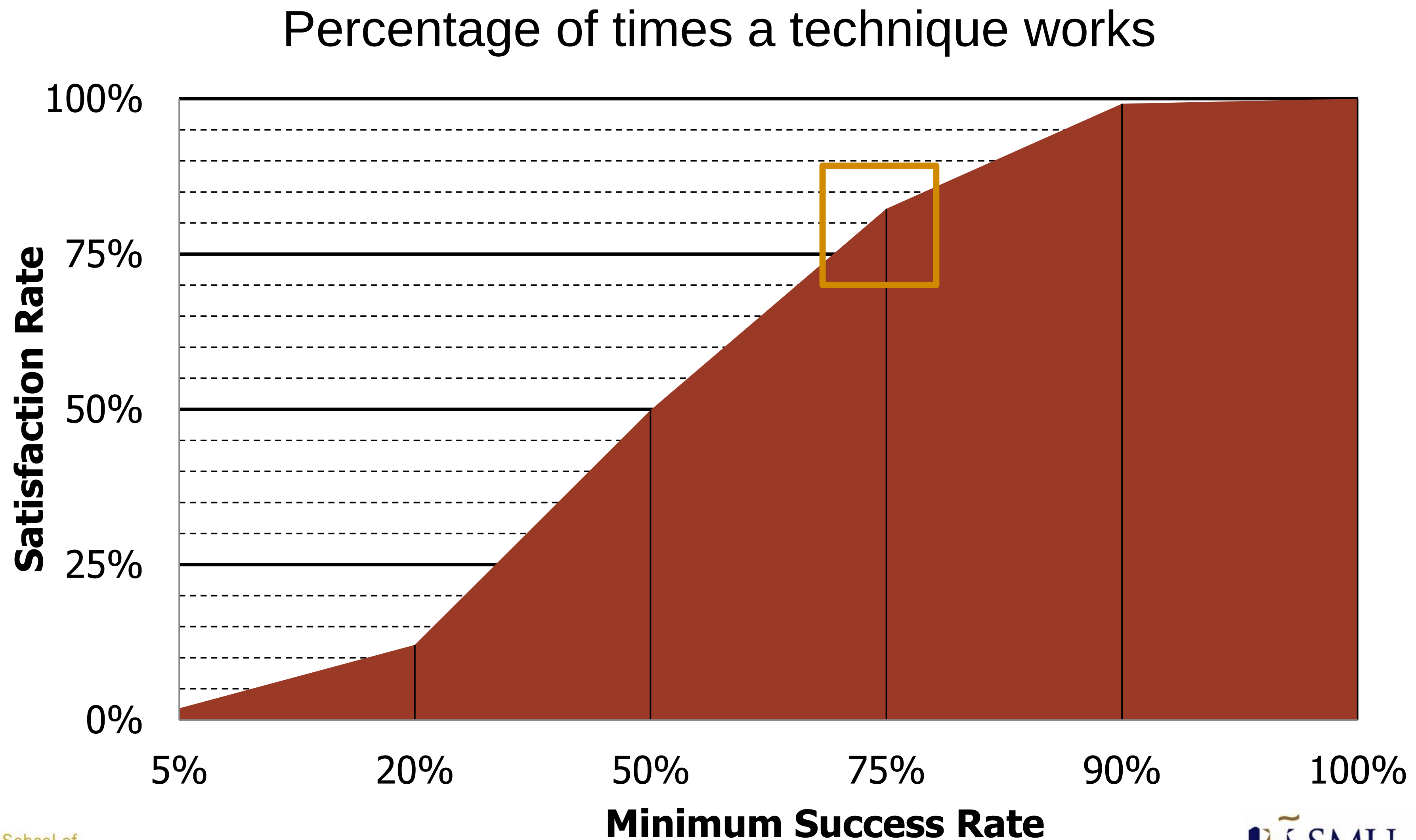
#3: Focus on the Top-5 Returned Results



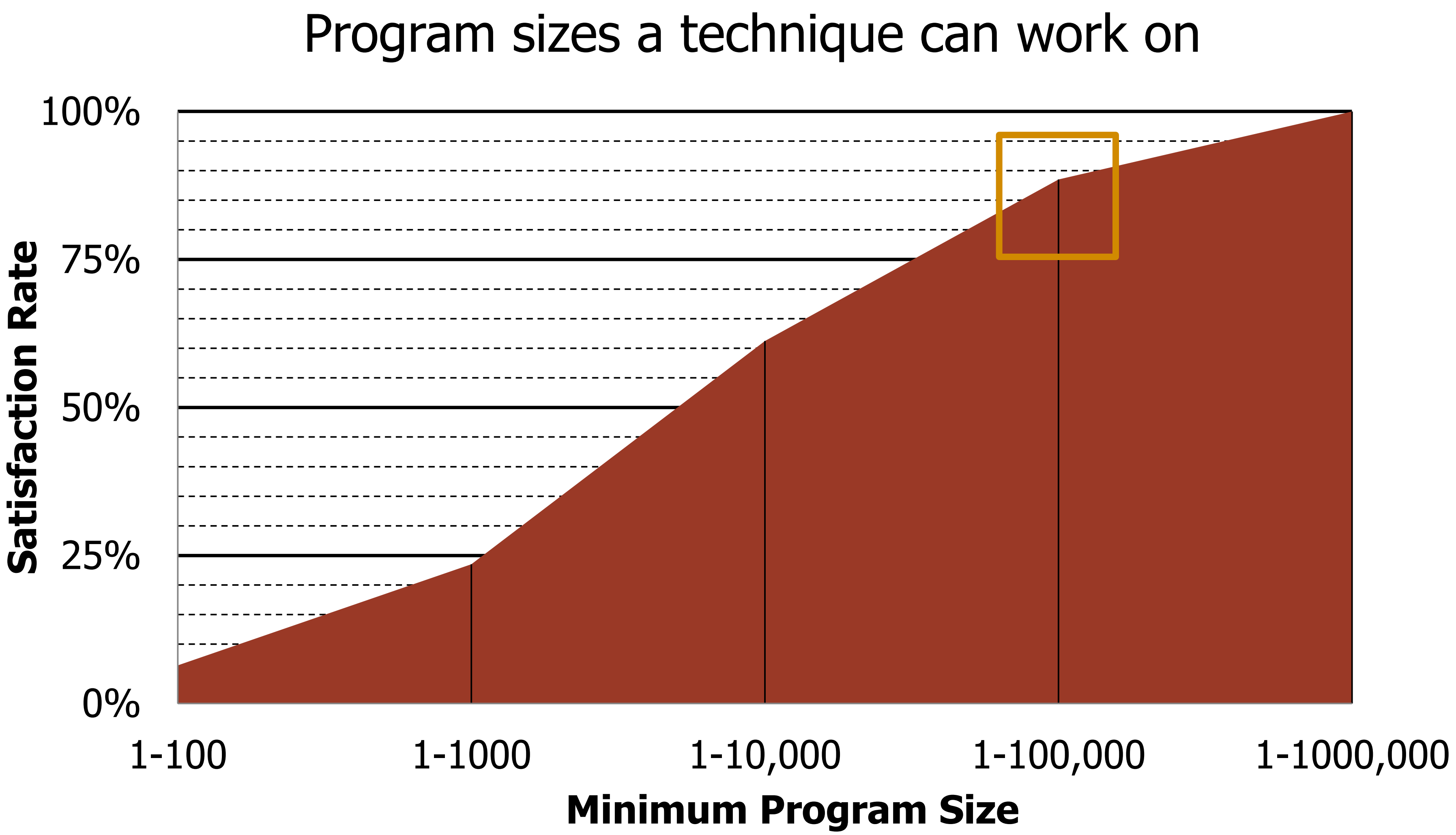
School of Information Systems

SMU Singapore Management University

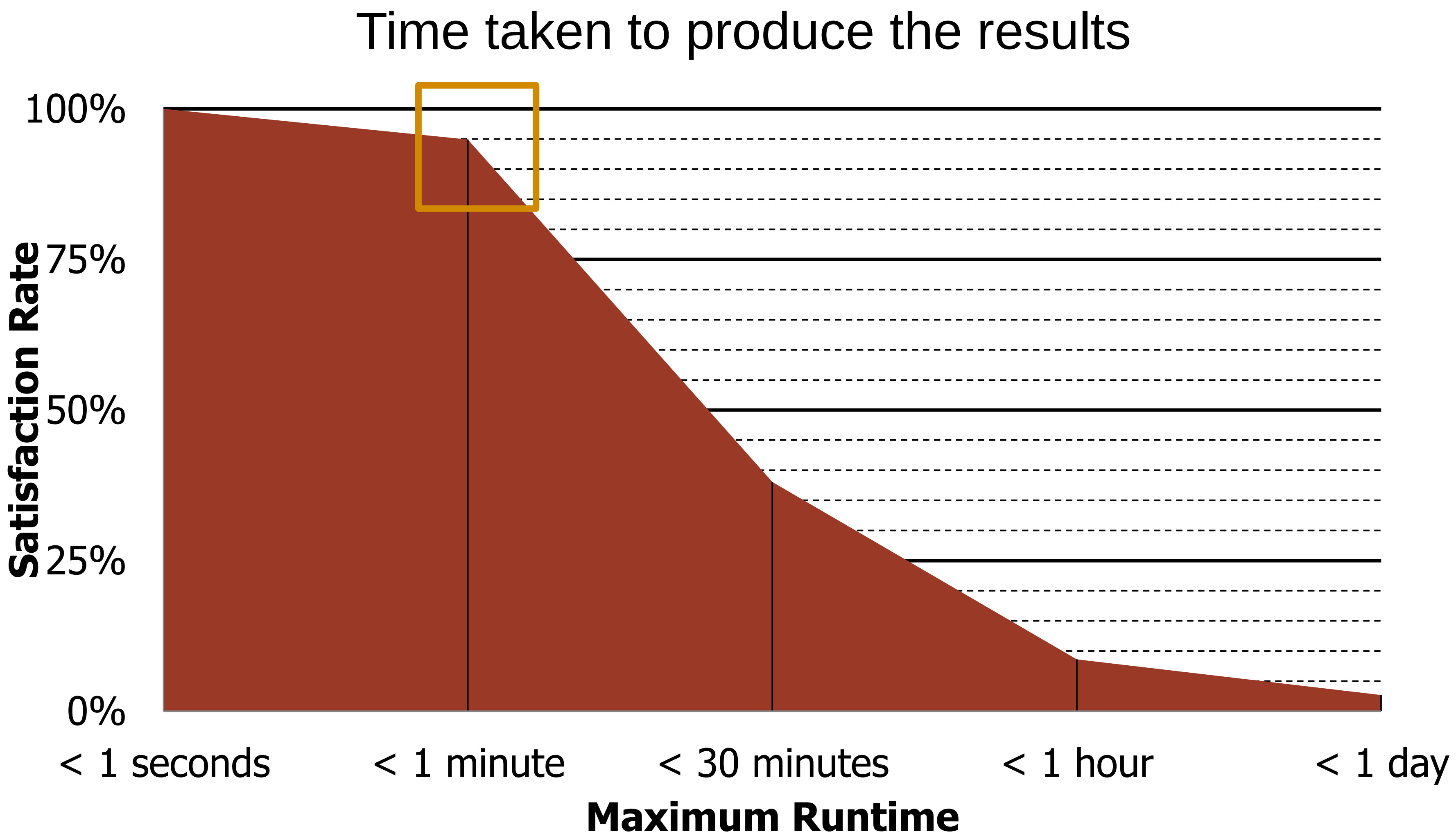
#4a: Needs to Work for 3 Out of 4 Cases



#4b: Need to Deal with 100kLOC



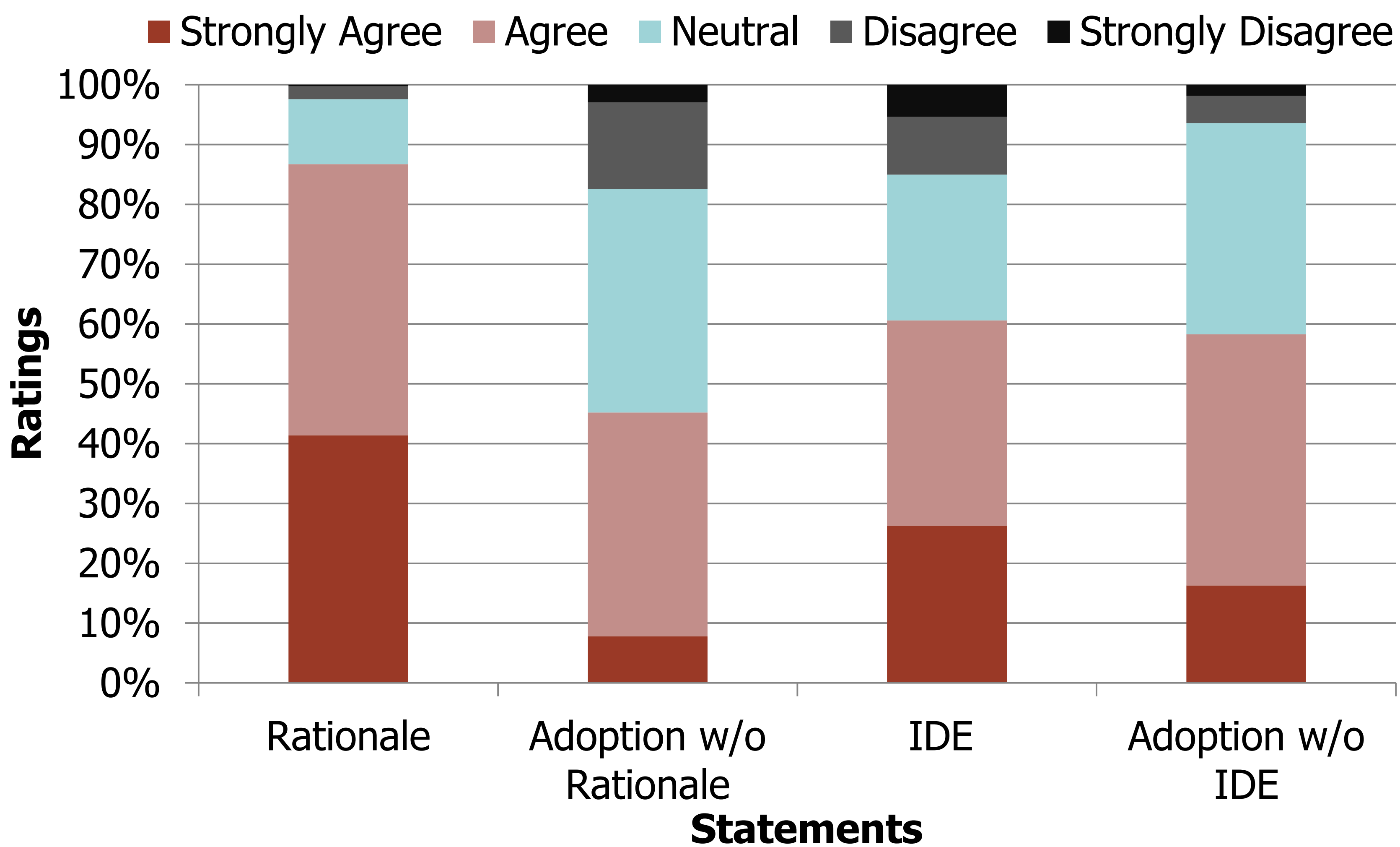
#4c: Need to Produce Results Within a Minute



School of Information Systems

SMU SINGAPORE MANAGEMENT UNIVERSITY

#5: Provide Rationales and IDE Integration



School of Information Systems

SMU SINGAPORE MANAGEMENT UNIVERSITY

Takeaway

- Practitioners need automated debugging tools and highly value research in this area
- Practitioners have a high bar of adoption
  - No existing techniques have fully met developers' expectations (e.g., >75% satisfaction rate)
- Future work needs to be done to improve:
  - Reliability, scalability, efficiency
  - To eventually overcome adoption thresholds
- Future work is needed to integrate research tools to IDEs, and provide rationale beyond recommendations.

Summary

- Automated tools are needed to help in debugging
- Bug/fault localization identifies buggy code
  - *Combine* debugging hints to boost performance
  - Bugs are not all alike; *adaptive solution* is needed
- Automated repair removes errors from buggy code
  - Automatically/manually constructed *knowledge base* can be used to avoid nonsensical patches
- Future work: overcome adoption barriers
  - Identifying adoption thresholds is the first step
  - Community-wide effort is needed to overcome them

Job Openings

Several postdocs, research engineers, visiting students, and PhD students needed for 3 funded projects starting in Jan/Mar 2017.

Please Consider Joining Us

Thank you!

Questions? Comments? Advice?

davidlo@smu.edu.sg