Similarities Across Libraries: Making a Case for Leveraging Test Suites

Devika Sondhi[*], Divya Rani[+], Rahul Purandare[*]
[*] IIIT Delhi, India | [+] GITA, Bhubaneswar, India

12th IEEE International Conference on Software Testing, Verification and Validation
April 2019
Xi'an, China

Research supported by the Prime Minister's Fellowship, India with Microsoft Research as the industry partner

Introduction

- Mining test suites associated with similar functions across libraries, to generate test cases.
  - **Investigate the extent of existence of similar functions across libraries implemented in same or different languages.**
  - Effectiveness of using tests associated with functions to reveal defects in other similar functions.
  - Challenges in automating this testing process.

Introduction

- Mining test suites associated with similar functions across libraries, to generate test cases.
  - Investigate the extent of existence of similar functions across libraries implemented in same or different languages.
  - **Effectiveness of using tests associated with functions to reveal defects in other similar functions.**
  - Challenges in automating this testing process.

Introduction

- Mining test suites associated with similar functions across libraries, to generate test cases.
  - Investigate the extent of existence of similar functions across libraries implemented in same or different languages.
  - Effectiveness of using tests associated with functions to reveal defects in other similar functions.
  - **Challenges in automating this testing process.**

Motivating Example

NLTK library

```
>>> pos_tag(word_tokenize("John's big idea isn't all that bad."), tagset='universal')
[('John', 'NOUN'), ("'s", 'PRT'), ('big', 'ADJ'), ('idea', 'NOUN'), ('is', 'VERB'),
("n't", 'ADV'), ('all', 'DET'), ('that', 'DET'), ('bad', 'ADJ'), ('.', '.')]
```

Polyglot library

Failed example:

Text("John's big idea isn't all that bad.").pos_tags

Got:

[("John's", 'NUM'), ('big', 'ADJ'), ('idea', 'NOUN'), ("isn't", 'CONJ'), ('all', 'DET'), ('that', 'DET'), ('bad', 'ADJ'), ('.', 'PUNCT')]

Past Work

Mining similar codes for

- program comprehension[1], reusability, example retrieval[2], rapid prototyping[3], and discovering code theft[4]

[1]Sager et al., MSR'06: Detecting similar java classes using tree algorithms, [2]Bajracharya et al., FSE'10: Leveraging usage similarity for effective retrieval of examples in code repositories, [3]McMillan et al., ICSE'12: Recommending source code for use in rapid software prototypes, [4]Liu et al., KDD'06: Gplag: Detection of software plagiarism by program dependence graph analysis

Past Work

Mining source code for bug detection?

➢ restricted to code clones[1] or softwares/libraries implemented in same language[2]

[1]Jiang et al., FSE'07: Context-based detection of clone-related bugs

[2]Taneja et al., ASE'10: Mitv: Multiple-implementation testing of user-input validators for web applications

Objective 1

Investigate the extent of existence of similar functions across libraries implemented in same or different languages.

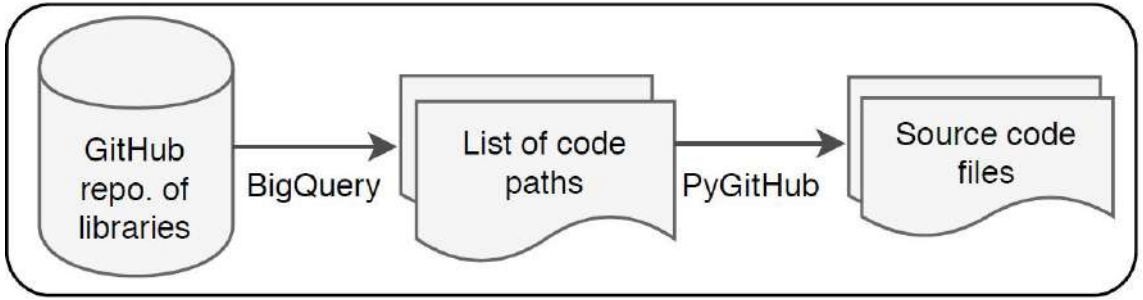
Data Extraction

- Dataset: 31,716 query functions over 20 pairs of training and testing set of libraries
  - function= (documentation, function name)

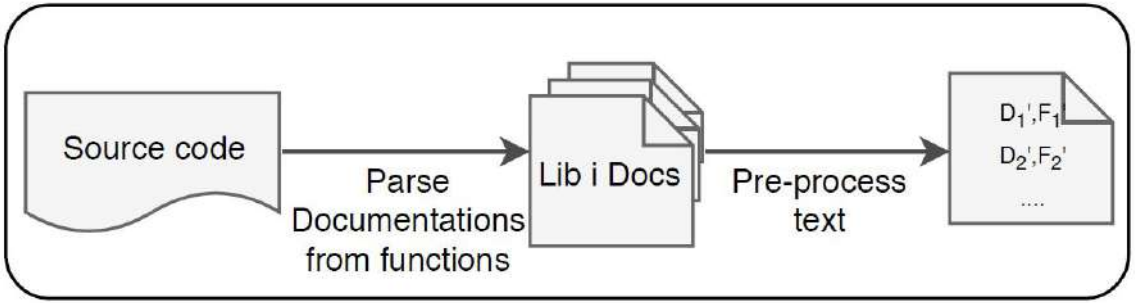
	Theme					
Language	**String and Numeric Manipulation**		**Natural Language Processing**		**Structural Validation**	
Java	Apache Commons Lang [1,443]	Google Guava [27,127]	Stanford CoreNLP [5,269]	Apache OpenNLP [650]	Apache Commons Validator [79]	Apache Commons IO [504]
Python	CPython [20,022]	Python String Utils [10]	NLTK [6,914]	Polyglot [967]	Validators [202]	Urllib3 [1,730]

Match Extractor Framework

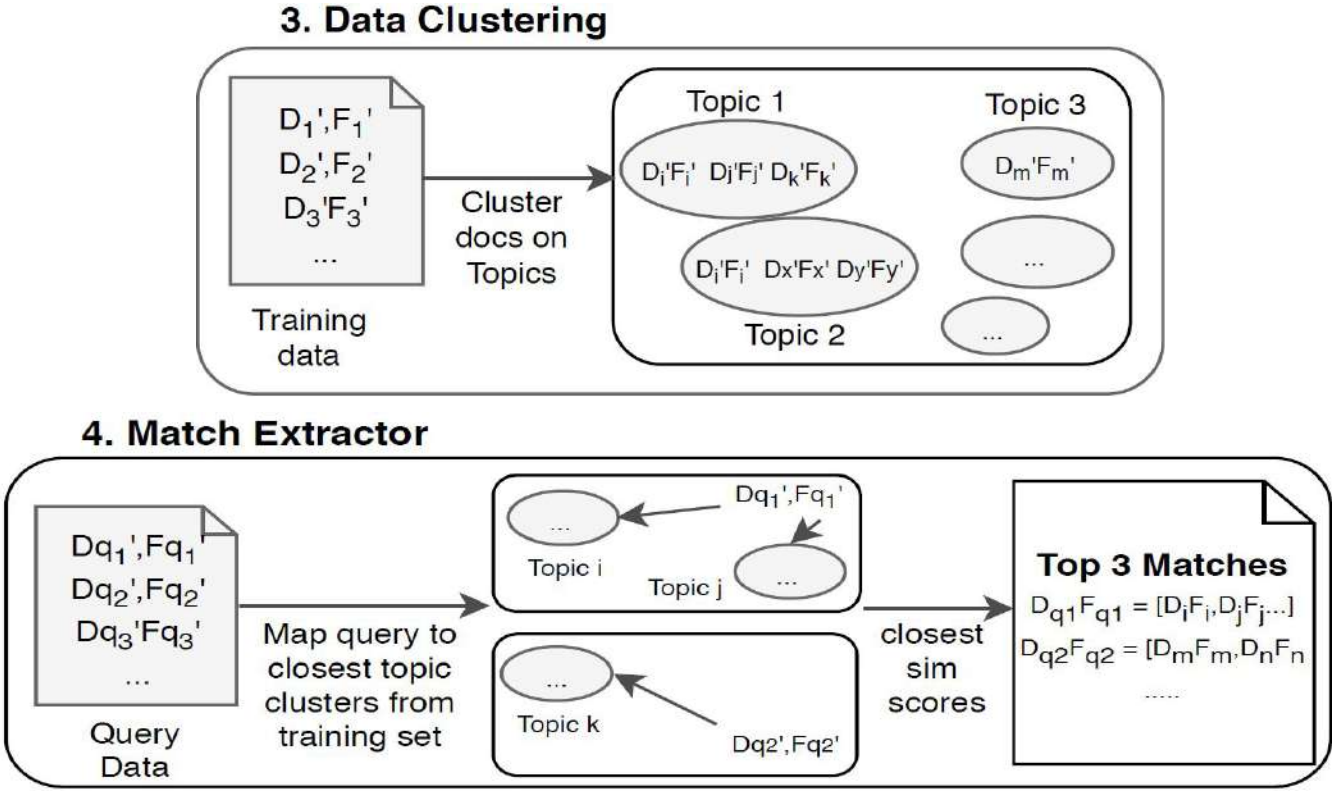
1. Raw Data Extraction



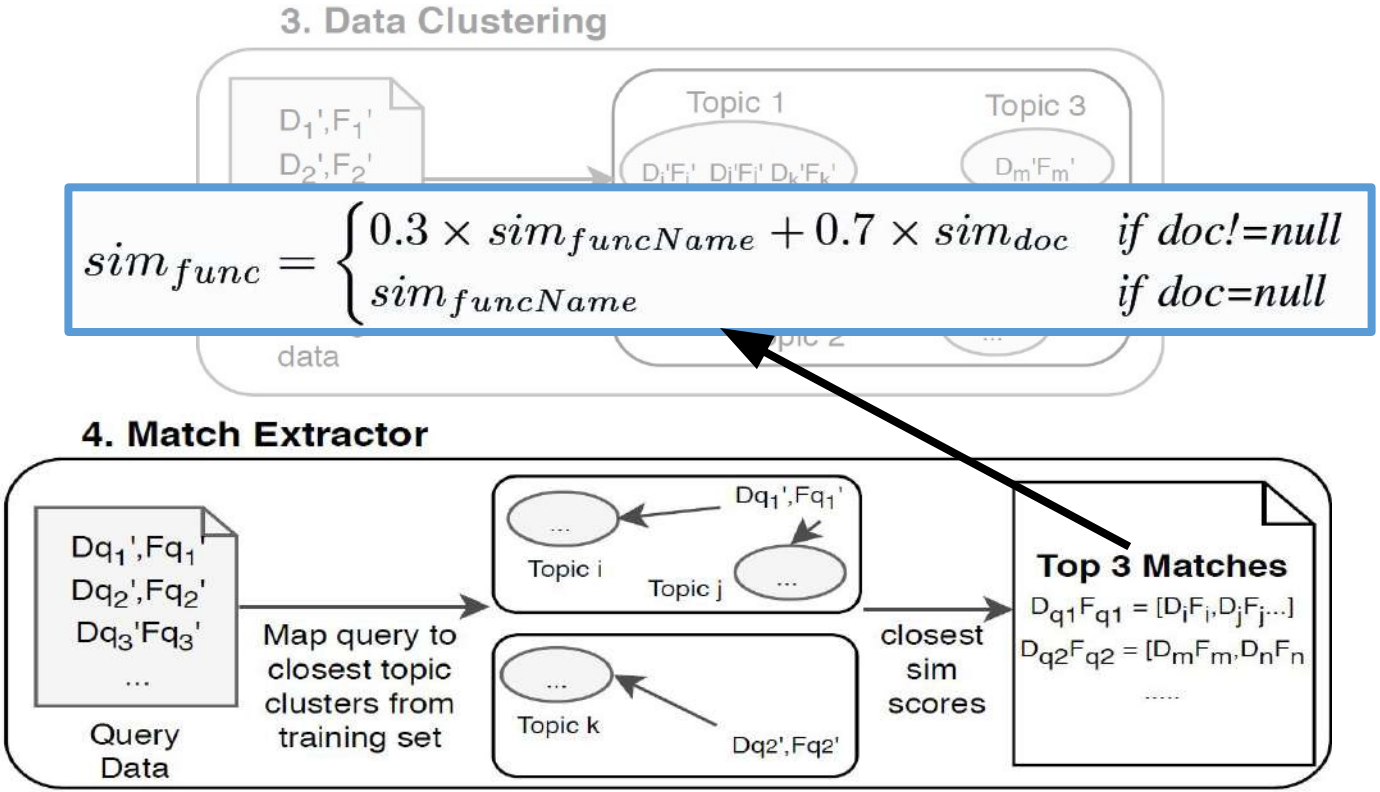
2. Data Refinement



Match Extractor Framework



Match Extractor Framework



Objective 2

Effectiveness of using tests associated with functions to reveal defects in other similar functions.

Data Analysis

- Extract test cases associated with correctly mapped function pairs
- Translating test assertions for the matched function across library
- Log failed assertions to verify with the documentation

```
public void testEmailUserName()   {
assertFalse(validator.isValid("john56789.john56789.john56789.john5678
9.john56789.john56789.john5@example.com"));}
```

```
@pytest.mark.parametrize(('value',), [
('john56789.john56789.john56789.john56789.john56789.john56789.john5@e
xample.com',)])
def test_invalid_email(value):
    assert isinstance(email(value), ValidationFailure)
```

Study Results

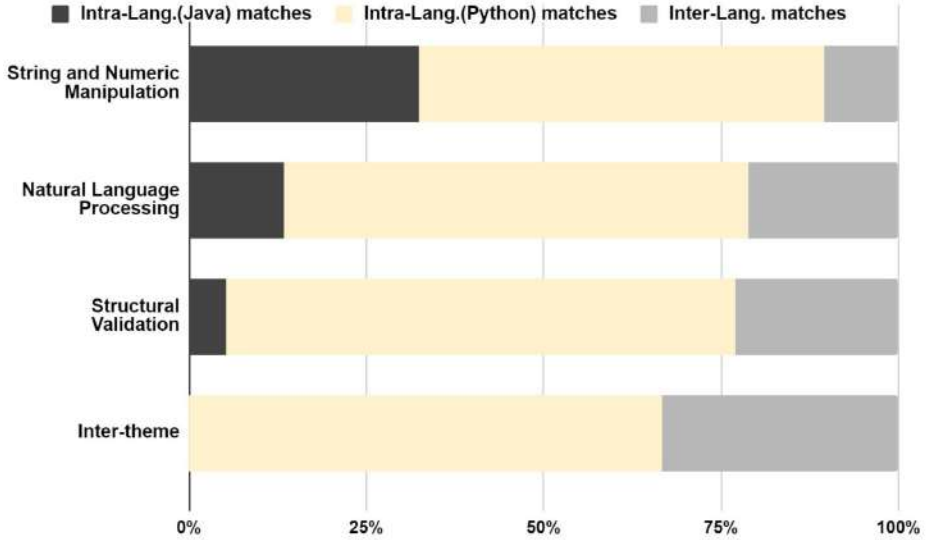
1. **How often do similar functions exist across libraries?**
2. Does the language and theme of the libraries influence the extent of similarity matches?

Study Results

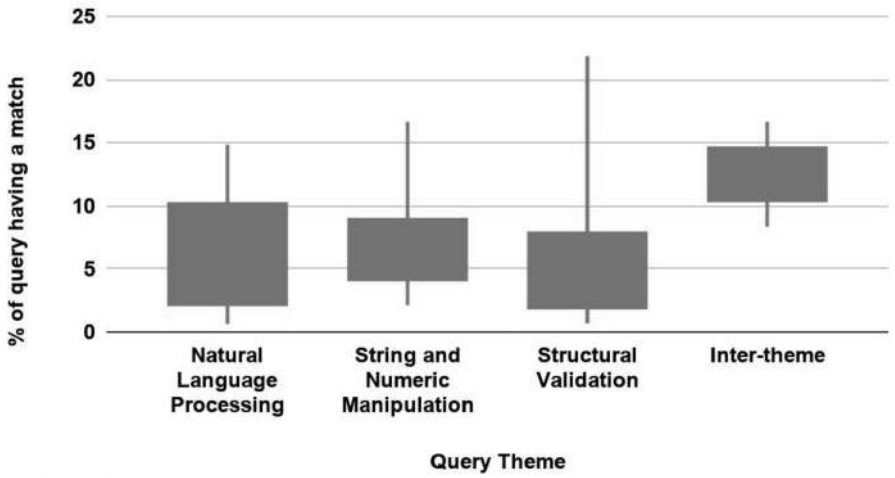
1. How often do similar functions exist across libraries?
2. **Does the language and theme of the libraries influence the extent of similarity matches?**

Study Results I

1,067 valid matches extracted for 31,716 query functions.



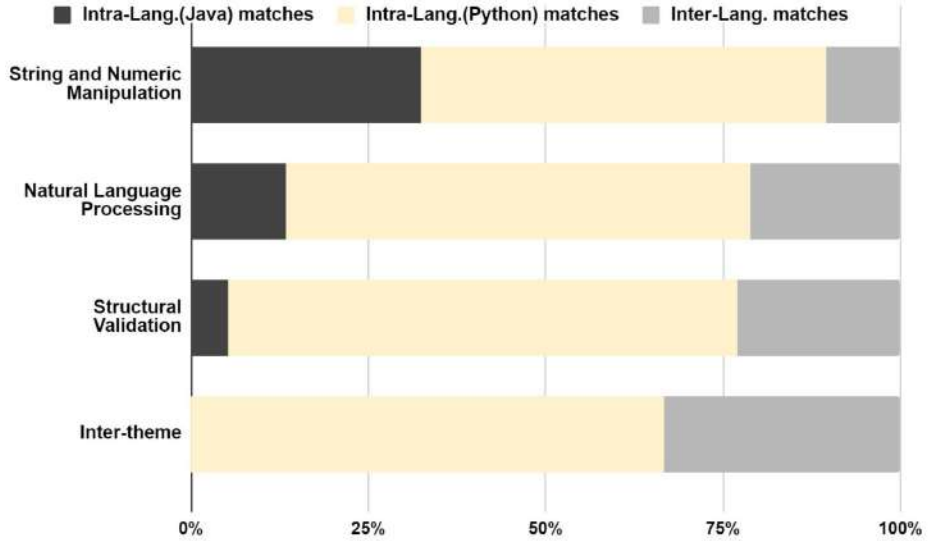
(a) Distribution of valid matches for queries within and across languages



(b) Distribution of query to match conversion across themes

Study Results I

1,067 valid matches extracted for 31,716 query functions.



(a) Distribution of valid matches for queries within and across languages

- Intra-language matches more than inter-language matches.
- The constructs and the paradigm of a language suit certain type of functionalities.

Nature of Similarity

- Exactly same functionality
  - *Example:* Validate an IP address
  - Useful for code reuse opportunities
- Superset-subset relation
  - *Example:* Union on sets of ‘objects’ to union on sets of integers
  - Useful from the perspective of improving or enhancing one implementation using another

Nature of Similarity

- High functional overlap, difference in data structures or data types
  - *Example:* Dictionary in Python and hashmap in Java implementations
  - Useful to assess the performance of the two functions
- Functionally similar but varying algorithm or APIs
  - *Example:* Similarity metrics such as path-similarity, Leacock-Chodorow similarity, and Resnik similarity
  - Useful for performance assessment of one implementation over the other

Study Results

3. **Does leveraging test suite for a function result in defect exposure in another similar function?**
4. Can user reported issues and pull requests in a library serve as a source to expose defects across another library?

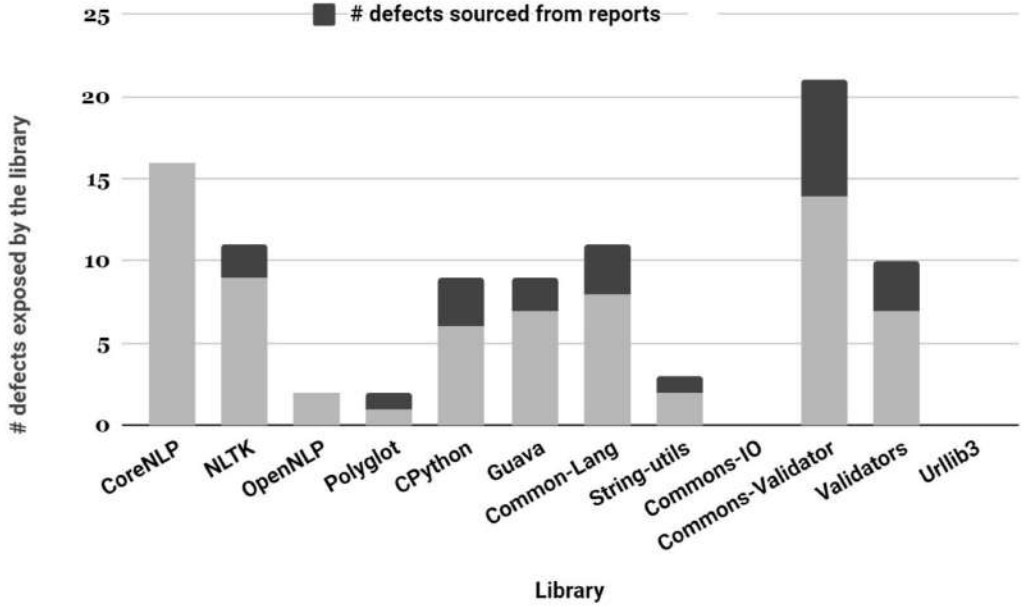
Study Results

3. Does leveraging test suite for a function result in defect exposure in another similar function?
4. **Can user reported issues and pull requests in a library serve as a source to expose defects across another library?**

Study Results II

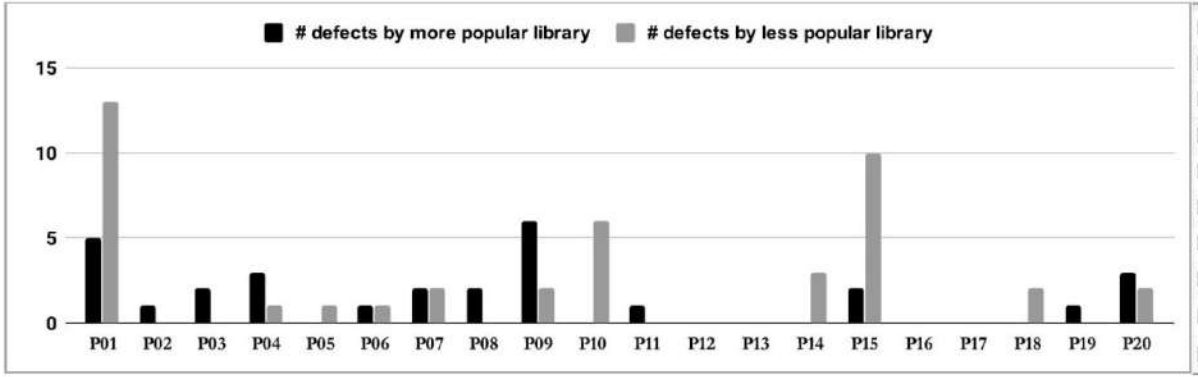
1,067 matches resulted in

- **72 defects** from 46 matches
- 40 defects validated by developers
- 30.6% defect-revealing test cases sourced from issue reports or pull requests



(a) Contribution by libraries in revealing defects and the distribution of defects sourced from issues and pull requests

Other Inferences: Influence of popularity



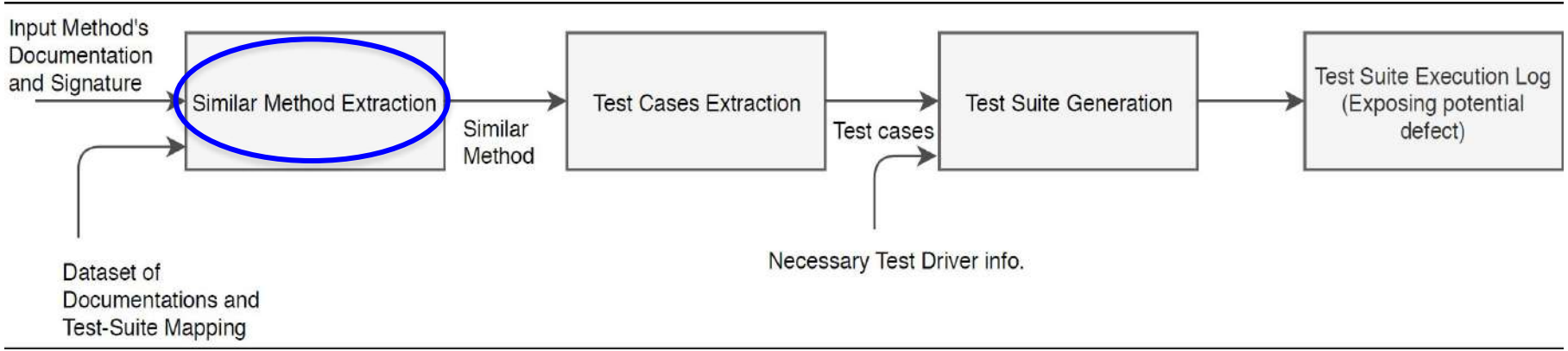
P01	NLTK;CoreNLP	P11	Guava;String-utils
P02	CoreNLP;OpenNLP	P12	Commons-Lang;String-utils
P03	CoreNLP;Polyglot	P13	CommonsIO;CommonsValidator
P04	NLTK;OpenNLP	P14	Urllib3;Commons-Validator
P05	Polyglot;OpenNLP	P15	Validators;Commons-Validator
P06	NLTK;Polyglot	P16	Urllib3;Commons-IO
P07	CPython;Commons-Lang	P17	Validators;Commons-IO
P08	CPython;String-utils	P18	Urllib3;Validators
P09	Guava;CPython	P19	Commons-Validator;String-utils
P10	Guava;Commons-Lang	P20	Validators;String-utils

Contribution of defects by each pair of training and testing library. Library name pair convention followed in the plot is <more popular library>;<less popular library>

Lesser popular libraries could also be beneficial in revealing defects in libraries that are more popular.

Challenges in Automation

Automated



Challenges in Automation

- Improve Match Extractor's precision
  - F1 score of 0.21, owing to a low precision of 0.12 (at recall of 0.89), at a threshold similarity score of 0.34.
  - Inclusion of programmatic features such as tokens extraction from statements.
    - Need lightweight analysis to minimize language dependency.
- Varying test frameworks across libraries, especially across languages

Challenges in Automation

- Test Extraction and Generation: Variation in function signatures
  - ***Difference in input data type:*** **`CharSequence` and `Strings`**
  - *Difference in the order of parameters:* `F1(A1,B1)` and `F2(B2,A2)`
    - Heuristics to map the appropriate parameters to generate tests.
  - *Different number of parameters:* `F1(A1,B1,C1)` and `F2(A2,B2)`
    - Extracting relevant parameter(s); Substituting the input for extra parameter(s)
  - *Difference in the source of input:*
    - `Splitter.on(String separator).split(CharSequence sequence)` and `split(String str, String separatorChars)`

Challenges in Automation

- Test Extraction and Generation: Variation in function signatures
  - *Difference in input data type:* `CharSequence` and `Strings`
  - ***Difference in the order of parameters:*** **`F1(A1,B1)` and `F2(B2,A2)`**
    - **Heuristics to map the appropriate parameters to generate tests.**
  - *Different number of parameters:* `F1(A1,B1,C1)` and `F2(A2,B2)`
    - Extracting relevant parameter(s); Substituting the input for extra parameter(s)
  - *Difference in the source of input:*
    - `Splitter.on(String separator).split(CharSequence sequence)` and `split(String str, String separatorChars)`

Challenges in Automation

- Test Extraction and Generation: Variation in function signatures
  - *Difference in input data type:* `CharSequence` and `Strings`
  - *Difference in the order of parameters:* `F1(A1,B1)` and `F2(B2,A2)`
    - Heuristics to map the appropriate parameters to generate tests.
  - ***Different number of parameters:* `F1(A1,B1,C1)` and `F2(A2,B2)`**
    - **Extracting relevant parameter(s); Substituting the input for extra parameter(s)**
  - *Difference in the source of input:*
    - `Splitter.on(String separator).split(CharSequence sequence)` and `split(String str, String separatorChars)`

Challenges in Automation

- Test Extraction and Generation: Variation in function signatures
  - *Difference in input data type:* `CharSequence` and `Strings`
  - *Difference in the order of parameters:* `F1(A1,B1)` and `F2(B2,A2)`
    - Heuristics to map the appropriate parameters to generate tests.
  - *Different number of parameters:* `F1(A1,B1,C1)` and `F2(A2,B2)`
    - Extracting relevant parameter(s); Substituting the input for extra parameter(s)
  - ***Difference in the source of input:***
    - **`Splitter.on(String separator).split(CharSequence sequence)` and `split(String str, String separatorChars)`**

Summary

- An average of 7.4% of the functions in a queried library resulted in a matching (similar) function in another library.
- The study exposed 72 defects obtained from 4.3% of the extracted matches.

Favourable observations to build automated testing tools that leverage software mining.

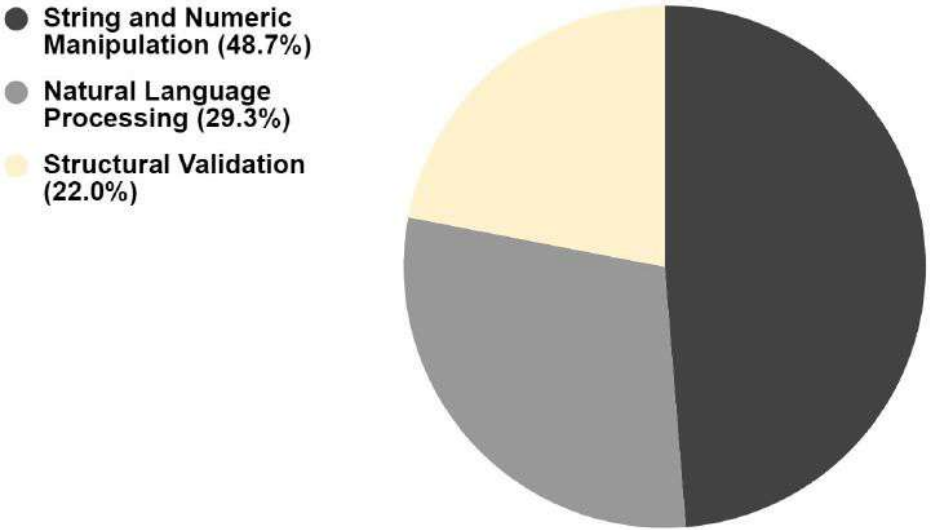
Questions?

Annexure

Why Low Precision?

- Query functions with no actual match in the training library: The extractor still returned some match that contained common key terms
  - Hence, low precision but a high recall
- Context specific: query as “with decimals parsing”; match as “Parses the specified string as a signed decimal integer value...”
  - decimal point vs. signed integer with base 10 (referred to as decimal)
- Highly generic description: *delete* vs. *deleteAll*
  - Could be deletion on anything

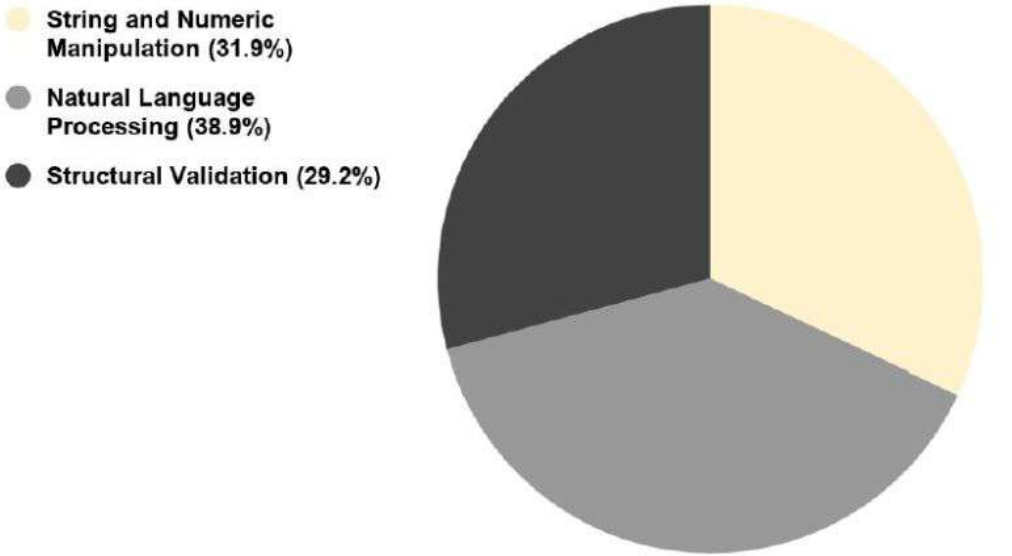
Other inferences: Matches across themes



(a) Distribution of valid matches across themes

- No indication of a major dominance of any theme in finding the matches.
- The idea of leveraging similarities may be explored for diverse themes.

Other inferences: Defects across themes



Distribution of defects across themes

- No theme found to be dominating in the share of defects revealed in it.
- Approach may be applicable to diverse themes.

Defect Samples

Defective function	Test taken from	Defect description
MTEFileReader. lemma_sents in NLTK	Morphology.lemma in CoreNLP	Lemmatizer cannot handle contractions such as *'ll*, and personal pronouns such as 'her' that should be lemmatized to 'she', but incorrectly returns 'her'
Sentence.sentiment in CoreNLP	SentimentIntensityAnalyzer. polarity_scores in NLTK	The input "Sentiment analysis with VADER has never been this good ." is incorrectly assigned as negative sentiment.
WordUtils.wrap in Commons-lang	TextWrapper.wrap in Cpython	The behavior is inconsistent across the two tests about whether the leading space of string should be retained or not while wrapping. The string " This is a sentence with leading whitespace." retains the leading whitespace over wrap length 50, but removes it at a wrap length 30.
Link to issue report:	https://bugs.python.org/issue15510	

Threats to Validity

- Repeatability of observations: Topic coherence score (Cv) of at least 0.49 on every pair of libraries used to obtain the topic models.
  - Low standard deviation on the coherence score, not exceeding 0.01, as observed over five runs of topic modelling for every pair of libraries
    - Indicating results unlikely to change significantly with different runs of the LDA
- On generalization of study: 31,716 queries sourced from 12 libraries in 2 languages and 3 themes.
  - languages that support multiple paradigms and typing (statically vs. dynamically typed) ; commonly used in the software projects (as listed by GitHub in 2017*)
  - 3 diverse themes capturing 4 libraries from each theme

* https://octoverse.github.com/

References

1. C. Liu, C. Chen, J. Han, and P. S. Yu, “Gplag: Detection of software plagiarism by program dependence graph analysis,” in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ser. KDD ’06. New York, NY, USA: ACM, 2006, pp. 872–881.
2. S. K. Bajracharya, J. Ossher, and C. V. Lopes, “Leveraging usage similarity for effective retrieval of examples in code repositories,” in Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE ’10. New York, NY, USA: ACM, 2010, pp. 157–166.
3. T. Sager, A. Bernstein, M. Pinzger, and C. Kiefer, “Detecting similar java classes using tree algorithms,” in Proceedings of the 2006 International Workshop on Mining Software Repositories, ser. MSR ’06. New York, NY, USA: ACM, 2006, pp. 65–71.
4. C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher, “Recommending source code for use in rapid software prototypes,” in Proceedings of the 34th International Conference on Software Engineering, ser. ICSE ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 848–858.
5. L. Jiang, Z. Su, and E. Chiu, “Context-based detection of clone-related bugs,” in Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ser. ESEC-FSE ’07. New York, NY, USA: ACM, 2007, pp. 55–64.
6. K. Taneja, N. Li, M. R. Marri, T. Xie, and N. Tillmann, “Mitv: Multiple-implementation testing of user-input validators for web applications,” in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ser. ASE ’10. New York, NY, USA: ACM, 2010, pp. 131–134.