Usage and Refactoring Studies Of Python Regular Expressions

Carl Chapman

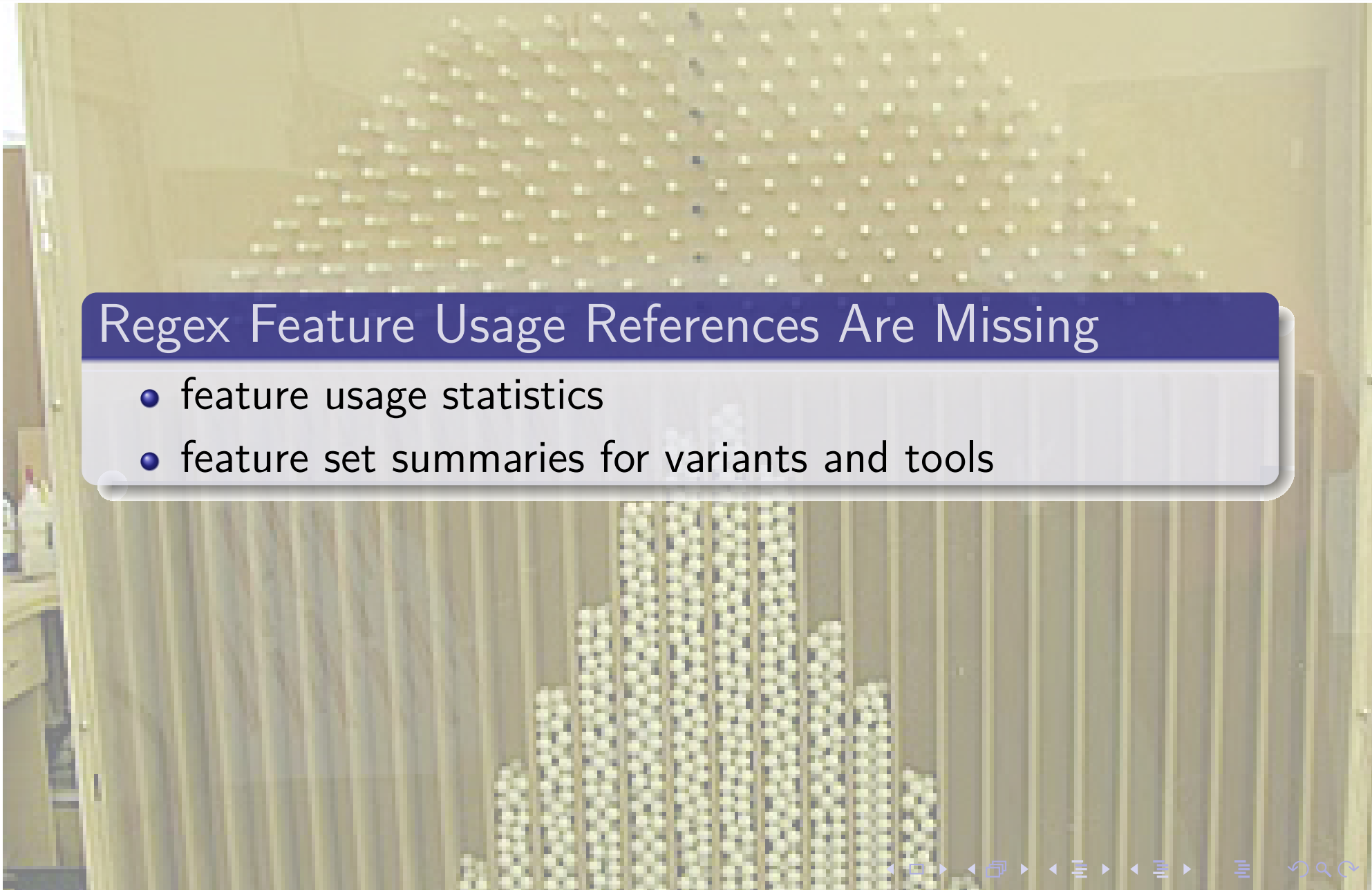
Iowa State University

carlallenchapman@gmail.com

13 April, 2016

Overview

What Features Are Used Most Frequently?



Regex Feature Usage References Are Missing

- feature usage statistics
- feature set summaries for variants and tools

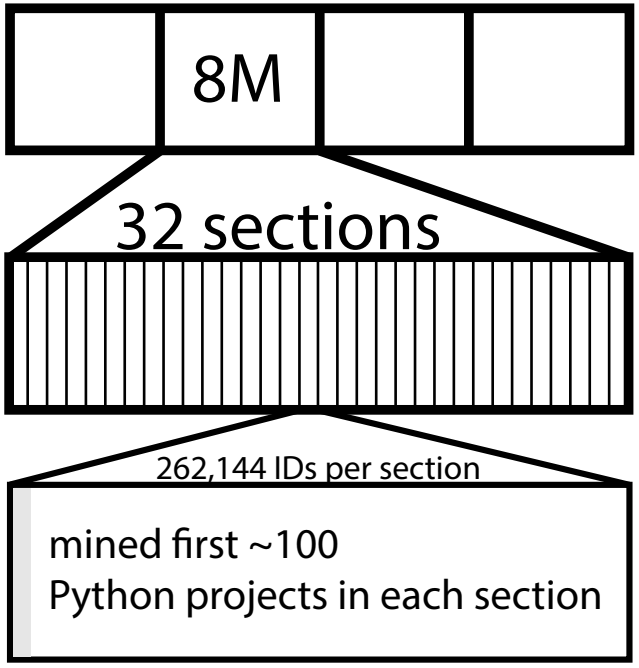
Why Python?

Python has most commonly shared features, few rarely shared features.

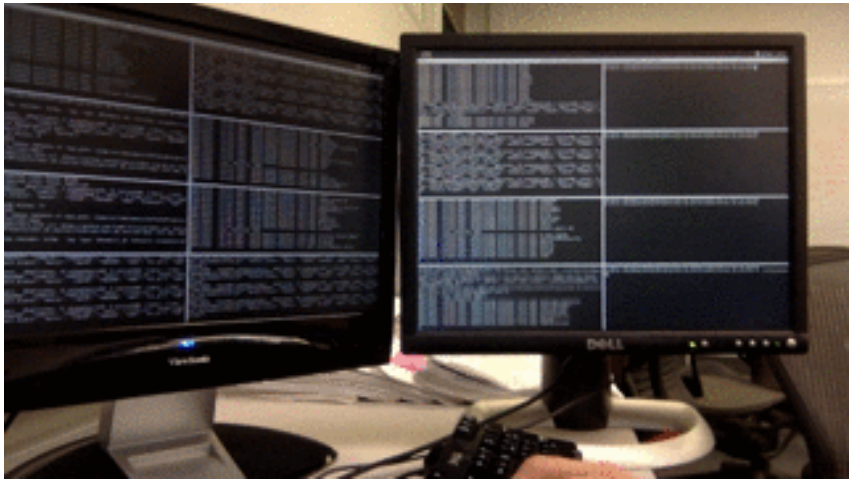
Code	Example	Python	Perl	.Net	Ruby	Java	RE2	JavaScript	POSIX ERE
ADD	z+	●	●	●	●	●	●	●	●
CG	(caught)	●	●	●	●	●	●	●	●
KLE	.*	●	●	●	●	●	●	●	●
CCC	[aeiou]	●	●	●	●	●	●	●	●
ANY	.	●	●	●	●	●	●	●	●
RNG	[a-z]	●	●	●	●	●	●	●	●
STR	^	●	●	●	●	●	●	●	●
END	$	●	●	●	●	●	●	●	●
NCCC	[^qwxf]	●	●	●	●	●	●	●	●
WSP	\s	●	●	●	●	●	●	●	●
OR	a|b	●	●	●	●	●	●	●	●
DEC	\d	●	●	●	●	●	●	●	●
WRD	\w	●	●	●	●	●	●	●	●
QST	z?	●	●	●	●	●	●	●	●
LZY	z+?	●	●	●	●	●	●	●	●
NCG	a(?:b)c	●	●	●	●	●	●	●	●
PNG	(?P<name>x)	●	●	●	●	●	●	●	●
SNG	z{8}	●	●	●	●	●	●	●	●
NWSP	\S	●	●	●	●	●	●	●	●
DBB	z{3,8}	●	●	●	●	●	●	●	●
NLKA	a(?!yz)	●	●	●	●	●	●	●	●
WNW	\b	●	●	●	●	●	●	●	●
NWRD	\W	●	●	●	●	●	●	●	●
LWB	z{15,}	●	●	●	●	●	●	●	●
LKA	a(?=bc)	●	●	●	●	●	●	●	●
OPT	(?i)CasE	●	●	●	●	●	●	●	●
NLKB	(?<!x)yz	●	●	●	●	●	●	●	●
LKB	(?<=a)bc	●	●	●	●	●	●	●	●
ENDZ	\Z	●	●	●	●	●	●	●	●
BKR	\1	●	●	●	●	●	●	●	●
NDEC	\D	●	●	●	●	●	●	●	●
BKRN	(P?=name)	●	●	●	●	●	●	●	●
VWSP	\v	●	●	●	●	●	●	●	●
NWNW	\B	●	●	●	●	●	●	●	●

Code	Example	Python	Perl	.Net	Ruby	Java	RE2	JavaScript	POSIX ERE
RCUN	(?n)	●	●	●	●	●	●	●	●
RCUZ	(?R)	●	●	●	●	●	●	●	●
GPLS	\g{+1}	●	●	●	●	●	●	●	●
GBRK	\g{name}	●	●	●	●	●	●	●	●
GSUB	\g<name>	●	●	●	●	●	●	●	●
KBRK	\k<name>	●	●	●	●	●	●	●	●
IFC	(?(cond)X)	●	●	●	●	●	●	●	●
IFEC	(?(cnd)X|else)	●	●	●	●	●	●	●	●
ECOD	(?{code})	●	●	●	●	●	●	●	●
ECOM	(?#comment)	●	●	●	●	●	●	●	●
PRV	\G	●	●	●	●	●	●	●	●
LHX	\uFFFF	●	●	●	●	●	●	●	●
POSS	a?+	●	●	●	●	●	●	●	●
NNCG	(?<name>X)	●	●	●	●	●	●	●	●
MOD	(?i)z(?-i)z	●	●	●	●	●	●	●	●
ATOM	(?>X)	●	●	●	●	●	●	●	●
CCCI	[a-z&&[^f]]	●	●	●	●	●	●	●	●
STRA	\A	●	●	●	●	●	●	●	●
LNLZ	\Z	●	●	●	●	●	●	●	●
FINL	\z	●	●	●	●	●	●	●	●
QUOT	\Q...\E	●	●	●	●	●	●	●	●
JAVM	\p{javaMirrored}	●	●	●	●	●	●	●	●
UNI	\pL	●	●	●	●	●	●	●	●
NUNI	\PS	●	●	●	●	●	●	●	●
OPTG	(?flags:re)	●	●	●	●	●	●	●	●
EREQ	[[=o=]]	●	●	●	●	●	●	●	●
PXCC	[:alpha:]	●	●	●	●	●	●	●	●
TRIV	[^]	●	●	●	●	●	●	●	●
CCSB	[a-f-[c]]	●	●	●	●	●	●	●	●
VLKB	(?<=ab.+)	●	●	●	●	●	●	●	●
BAL	(?<close-open>)	●	●	●	●	●	●	●	●
NCND	(?(<n>)X|else)	●	●	●	●	●	●	●	●
BRES	(?|(A)|(B))	●	●	●	●	●	●	●	●
QNG	(?'name're)	●	●	●	●	●	●	●	●

Project Selection



3,898 total Python projects found
(out of 42,000 inspected IDs)



Out of 3,898 pseudo-randomly selected Python projects, 1,645 contained one or more utilization.

Utilizations of the re module

function pattern flags

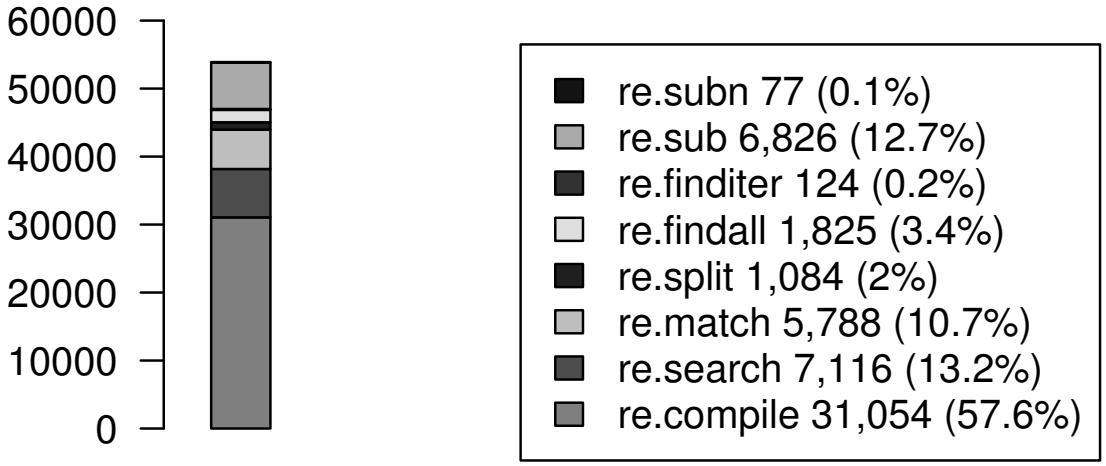
```
r1 = re.compile("(0|-?[1-9][0-9]*)$", re.MULTILINE)
```

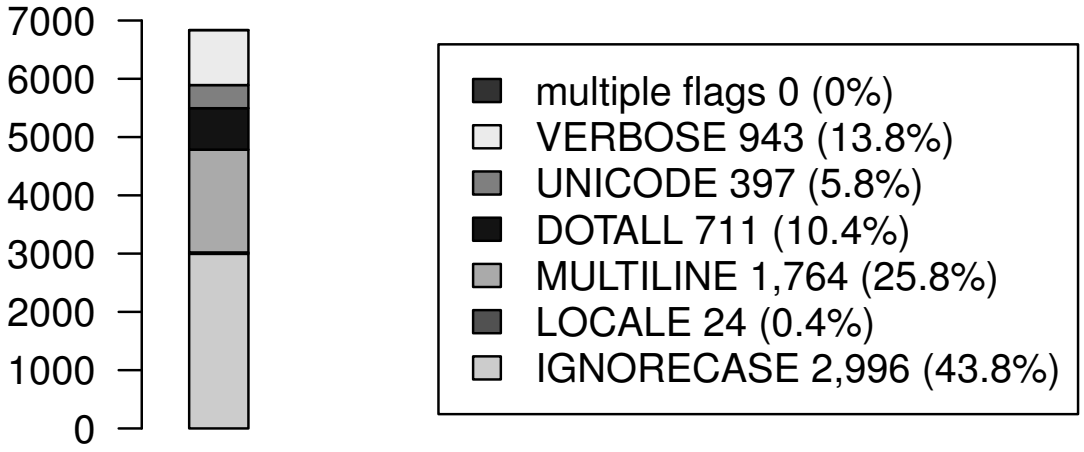
function which function of the re module is called?

pattern string used to specify regex behavior

flags modifies the regex engine

re module insights





Filtering Utilizations And Patterns

53,894 unique utilizations observed.

12.7% use behavioral flags

6.5% were non-static patterns

43,525 utilizations remain

13,711 distinct normalized patterns

73 had unsupported Unicode characters

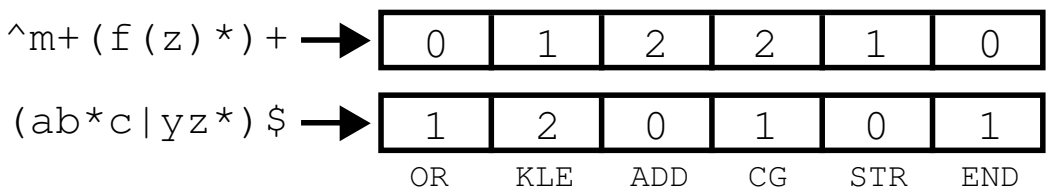
17 had non-Python features

22 had various errors

2 had ECOM feature - too rare to include

13,597 usable patterns remain for analysis

PCRE Parsing Patterns



All Python features are recognizable by PCRE

Feature Statistics

Rank	Code	Example	% Projects	NProjects	NFiles	NPatterns	MaxTokens
1	ADD	`z+`	73.2	1,204	9,165	6,003	30
2	CG	`(caught)`	72.6	1,194	9,559	7,130	17
3	KLE	`.*`	66.8	1,099	8,163	6,017	50
4	CCC	`[aeiou]`	62.4	1,026	7,648	4,468	42
5	ANY	`.`	61.1	1,005	6,277	4,657	60
6	RNG	`[a-z]`	51.6	848	5,092	2,631	50
7	STR	`^`	51.4	846	5,458	3,563	12
8	END	`$`	50.3	827	5,393	3,169	12
9	NCCC	`[^qwxf]`	47.2	776	3,947	1,935	15
10	WSP	`\s`	46.3	762	4,704	2,846	32
11	OR	`a|b`	43	708	3,926	2,102	15
12	DEC	`\d`	42.1	692	4,198	2,297	24
13	WRD	`\w`	39.5	650	2,952	1,430	13
14	QST	`z?`	39.2	645	3,707	1,871	35
15	LZY	`z+?`	36.8	605	2,221	1,300	12
16	NCG	`a(?:b)c`	24.6	404	1,709	791	28
17	PNG	`(?P<name>x)`	21.5	354	1,475	915	16

Rank	Code	Example	% Projects	NProjects	NFiles	NPatterns	MaxTokens
18	SNG	`z{8}`	20.7	340	1,267	581	17
19	NWSP	`\S`	16.4	270	776	484	10
20	DBB	`z{3,8}`	14.5	238	647	367	11
21	NLKA	`a(?!yz)`	11.1	183	489	131	3
22	WNW	`\b`	10.1	166	438	248	36
23	NWRD	`\W`	10	165	305	94	6
24	LWB	`z{15,}`	9.6	158	281	91	3
25	LKA	`a(?=bc)`	9.6	158	358	112	4
26	OPT	`(?i)CasE`	9.4	154	377	231	2
27	NLKB	`(?<!x)yz`	8.3	137	296	94	4
28	LKB	`(?<=a)bc`	7.3	120	255	80	4
29	ENDZ	`\Z`	5.5	90	149	89	1
30	BKR	`\1`	5.1	84	129	60	4
31	NDEC	`\D`	3.5	58	92	36	6
32	BKRN	`(P?=name)`	1.7	28	44	17	2
33	VWSP	`\v`	0.9	15	16	13	2
34	NWNW	`\B`	0.7	11	11	4	2

Feature Statistics - Top 8

Rank	Code	Example	% Projects	NProjects	NFiles	NPatterns	MaxTokens
1	ADD	`z+`	73.2	1,204	9,165	6,003	30
2	CG	`(caught)`	72.6	1,194	9,559	7,130	17
3	KLE	`.*`	66.8	1,099	8,163	6,017	50
4	CCC	`[aeiou]`	62.4	1,026	7,648	4,468	42
5	ANY	`.`	61.1	1,005	6,277	4,657	60
6	RNG	`[a-z]`	51.6	848	5,092	2,631	50
7	STR	`^`	51.4	846	5,458	3,563	12
8	END	`$`	50.3	827	5,393	3,169	12

Ranked features: Languages

Rank	Code	Example	Python	Perl	.Net	Ruby	Java	RE2	JavaScript	POSIX	ERE
1	ADD	`z+`	●	●	●	●	●	●	●	●	
2	CG	`(caught)`	●	●	●	●	●	●	●	●	
3	KLE	`.*`	●	●	●	●	●	●	●	●	
4	CCC	`[aeiou]`	●	●	●	●	●	●	●	●	
5	ANY	`.`	●	●	●	●	●	●	●	●	
6	RNG	`[a-z]`	●	●	●	●	●	●	●	●	
7	STR	`^`	●	●	●	●	●	●	●	●	
8	END	`$`	●	●	●	●	●	●	●	●	
9	NCCC	`[^qwxf]`	●	●	●	●	●	●	●	●	
10	WSP	`\s`	●	●	●	●	●	●	●		●
11	OR	`a|b`	●	●	●	●	●	●	●	●	
12	DEC	`\d`	●	●	●	●	●	●	●		●
13	WRD	`\w`	●	●	●	●	●	●	●		●
14	QST	`z?`	●	●	●	●	●	●	●	●	
15	LZY	`z+?`	●	●	●	●	●	●	●		●
16	NCG	`a(?:b)c`	●	●	●	●	●	●	●		●
17	PNG	`(?P<name>x)`	●	●	●	●	●	●	●	●	

Rank	Code	Example	Python	Perl	.Net	Ruby	Java	RE2	JavaScript	POSIX	ERE
18	SNG	`z{8}`	●	●	●	●	●	●	●	●	
19	NWSP	`\S`	●	●	●	●	●	●	●		●
20	DBB	`z{3,8}`	●	●	●	●	●	●	●	●	
21	NLKA	`a(?!yz)`	●	●	●	●	●	●	●		●
22	WNW	`\b`	●	●	●	●	●	●	●		●
23	NWRD	`\W`	●	●	●	●	●	●	●		●
24	LWB	`z{15,}`	●	●	●	●	●	●	●	●	
25	LKA	`a(?=bc)`	●	●	●	●	●	●	●		●
26	OPT	`(?i)CasE`	●	●	●	●	●	●	●		●
27	NLKB	`(?<!x)yz`	●	●	●	●	●	●	●		●
28	LKB	`(?<=a)bc`	●	●	●	●	●	●	●		●
29	ENDZ	`\Z`	●	●	●	●	●	●	●	●	
30	BKR	`\1`	●	●	●	●	●	●	●	●	
31	NDEC	`\D`	●	●	●	●	●	●	●		●
32	BKRN	`(P?=name)`	●	●	●	●	●	●	●	●	
33	VWSP	`\v`	●	●	●	●	●	●	●	●	
34	NWNW	`\B`	●	●	●	●	●	●	●	●	

Ranked features: Languages - Notable Missing Features

Rank	Code	Example	Python	Perl	.Net	Ruby	Java	RE2	JavaScript	POSIX ERE
21	NLKA	`a(?!yz)`	●	●	●	●	●	⬤	●	⬤
22	WNW	`\b`	●	●	●	●	●	●	●	⬤
23	NWRD	`\W`	●	●	●	●	●	●	●	⬤
24	LWB	`z{15,}`	●	●	●	●	●	●	●	●
25	LKA	`a(?=bc)`	●	●	●	●	●	⬤	●	⬤
26	OPT	`(?i)CasE`	●	●	●	●	●	●	⬤	⬤
27	NLKB	`(?<!x)yz`	●	●	●	●	●	⬤	⬤	⬤
28	LKB	`(?<=a)bc`	●	●	●	●	●	⬤	⬤	⬤
29	ENDZ	`\Z`	●	⬤	⬤	⬤	⬤	⬤	⬤	⬤
30	BKR	`\1`	●	●	●	●	●	⬤	●	●
31	NDEC	`\D`	●	●	●	●	●	●	●	⬤

Ranked features: Analysis Tools

Rank	Code	Example	Brics	Hampi	Rex	Automata.Z3
1	ADD	`z+`	●	●	●	●
2	CG	`(caught)`	●	●	●	●
3	KLE	`.*`	●	●	●	●
4	CCC	`[aeiou]`	●	●	●	●
5	ANY	`.`	●	●	●	●
6	RNG	`[a-z]`	●	●	●	●
7	STR	`^`	●	●	●	●
8	END	`$`	●	●	●	●
9	NCCC	`[^qwxf]`	●	●	●	●
10	WSP	`\s`	●	●	●	●
11	OR	`a|b`	●	●	●	●
12	DEC	`\d`	●	●	●	●
13	WRD	`\w`	●	●	●	●
14	QST	`z?`	●	●	●	●
15	LZY	`z+?`	●	●	●	●
16	NCG	`a(?:b)c`	●	●	●	●
17	PNG	`(?P<name>x)`	●	●	●	●

Rank	Code	Example	Brics	Hampi	Rex	Automata.Z3
18	SNG	`z{8}`	●	●	●	●
19	NWSP	`\S`	●	●	●	●
20	DBB	`z{3,8}`	●	●	●	●
21	NLKA	`a(?!yz)`	●	●	●	●
22	WNW	`\b`	●	●	●	●
23	NWRD	`\W`	●	●	●	●
24	LWB	`z{15,}`	●	●	●	●
25	LKA	`a(?=bc)`	●	●	●	●
26	OPT	`(?i)CasE`	●	●	●	●
27	NLKB	`(?<!x)yz`	●	●	●	●
28	LKB	`(?<=a)bc`	●	●	●	●
29	ENDZ	`\Z`	●	●	●	●
30	BKR	`\1`	●	●	●	●
31	NDEC	`\D`	●	●	●	●
32	BKRN	`\g<name>`	●	●	●	●
33	VWSP	`\v`	●	●	●	●
34	NWNW	`\B`	●	●	●	●

What Are Regexes Used For?

Non-Anecdotal Knowledge About Usage Is Missing

- task categories
- behavioral categories

How to Categorize Regex Usages

1. ~~thorough inspection of 55K utilizations~~
2. ~~unguided manual categorization of 4,694 regexes (in 2 or more projects), without objective basis~~
3. ~~cluster by syntactic similarity like Jaccard or longest substring~~
4. ~~formal analytical subsumption, using hampi (94%?) cannot get it to work~~
5. ~~formal analytical subsumption, using brics (30% or less)~~
6. Chosen technique: cluster by behavioral similarity using Rex (61%)

Measuring Behavioral Similarity

```
Pattern A matches 100/100 of A's strings
Pattern B matches  90/100 of A's strings
Pattern A matches  50/100 of B's strings
Pattern B matches 100/100 of B's strings
```

	A	B
A	1.0	0.9
B	0.5	1.0

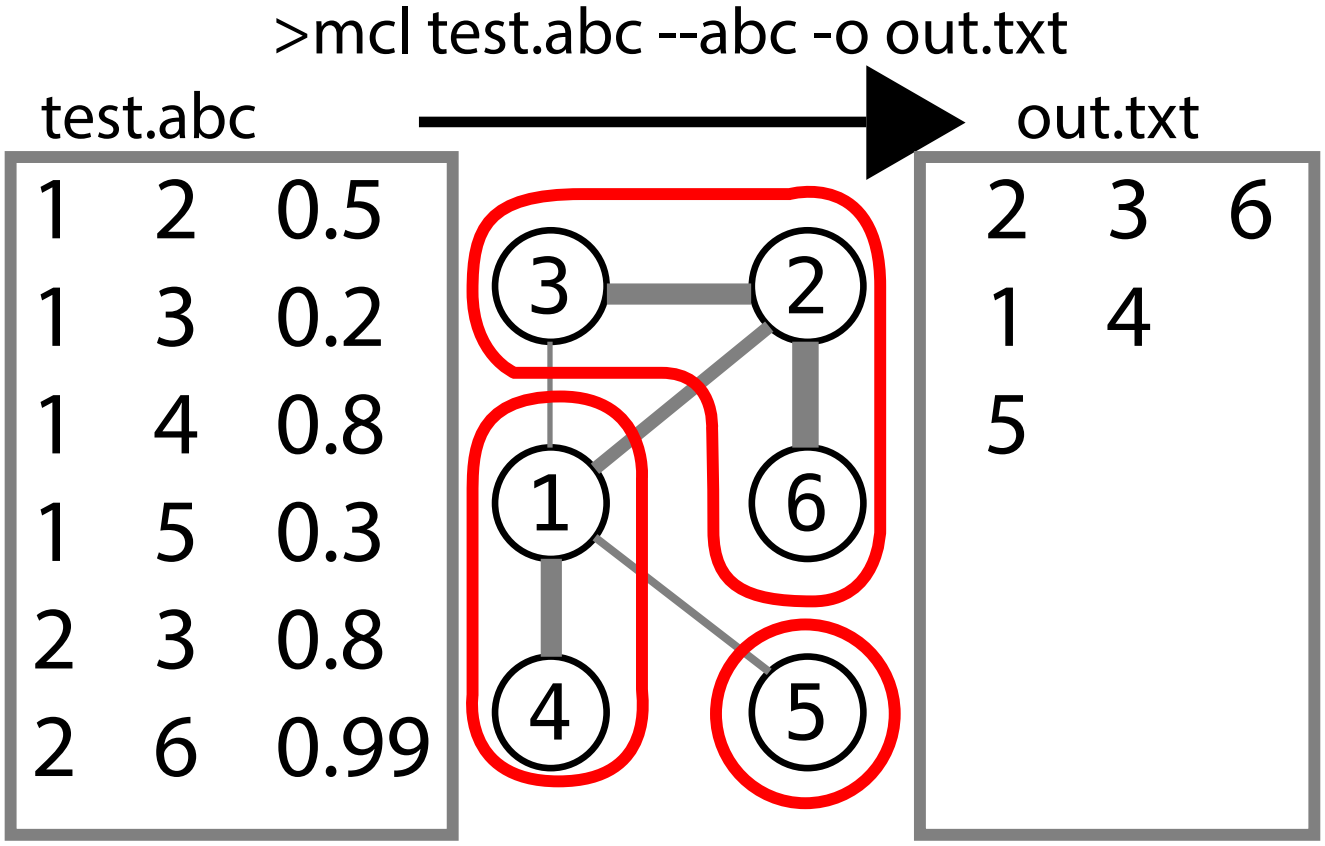
	A	B	C	D
A	1.0	0.0	0.9	0.0
B	0.2	1.0	0.8	0.7
C	0.6	0.8	1.0	0.2
D	0.0	0.6	0.1	1.0

→

	A	B	C	D
A	1.0			
B	0.1	1.0		
C	0.75	0.8	1.0	
D	0.0	0.65	0.15	1.0

Rex generates 400 strings
for each regex.
Convert scores to half-matrix
to make *.abc file for mcl.

MCL example


>mcl test.abc --abc -o out.txt
test.abc
1 2 0.5
1 3 0.2
1 4 0.8
1 5 0.3
2 3 0.8
2 6 0.99
3
2
1
6
4
5
out.txt
2 3 6
1 4
5

Clustering Results

Example Cluster

Index	Pattern	NProjects	Index	Pattern	NProjects
1	`\s*([^: ]*)\s*:(.*)`	9	7	`[:]`	6
2	`:+`	8	8	`([^:]+):(.*)`	6
3	`(:)`	8	9	`\s*:\s*`	4
4	`(:+)`	8	10	`\:`	2
5	`(:)(:*)`	8	11	`^([^:]*):[^:]*$`	2
6	`^([^:]*): *(.*)`	8	12	`^[^:]*:([^:]*)$`	2

From 2,871 distinct regexes
186 clusters where size ≥ 2
2,042 unclustered regexes

Six Categories Of Clusters

Category	Clusters	Patterns	Projects	% Projects
Multi Matches	21	237	295	40%
Specific Char	17	103	184	25%
Anchored Patterns	20	85	141	19%
Two or More Chars	16	40	120	16%
Content of Parens	10	46	111	15%
Code Search	15	27	92	13%

Code Search `.*rlen=([0-9]+)`

Two Or More Chars `@[a-z]+`

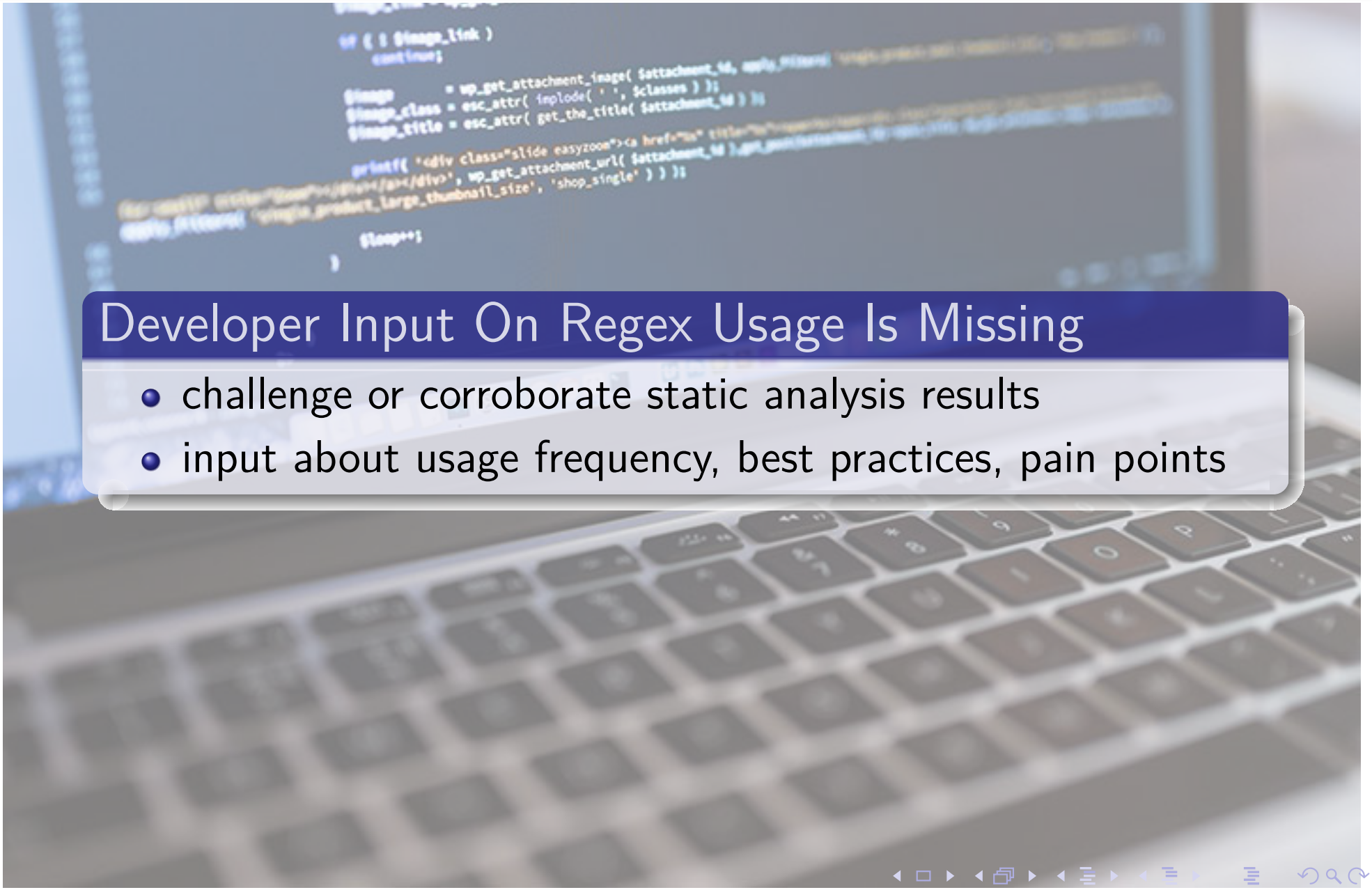
Bracket Parsing `<(.+)>`, `<[^>]*?>`

Specific Char `:+`, `}`, `%`

Anchored `^[-_A-Za-z0-9]+$`

Multiple Alternatives `(\s)`, `,|;`

How Do Developers Say They Use Regexes?



Developer Input On Regex Usage Is Missing

- challenge or corroborate static analysis results
- input about usage frequency, best practices, pain points

Feature Usage Is Consistent With Analysis

	CG	STR or END	LZY	WNW	look-arounds
very frequently	2	1	0	1	0
frequently	4	9	2	3	1
occasionally	9	5	6	6	2
rarely	2	2	2	2	5
very rarely	1	1	4	6	7
never	0	0	4	0	3
avg	5.8	6.1	4	4.8	3.5

Ranked Order: CG (2), STR/END (7,8), LZY (15), WNW (22), look-arounds (21, 25, 27, 28)

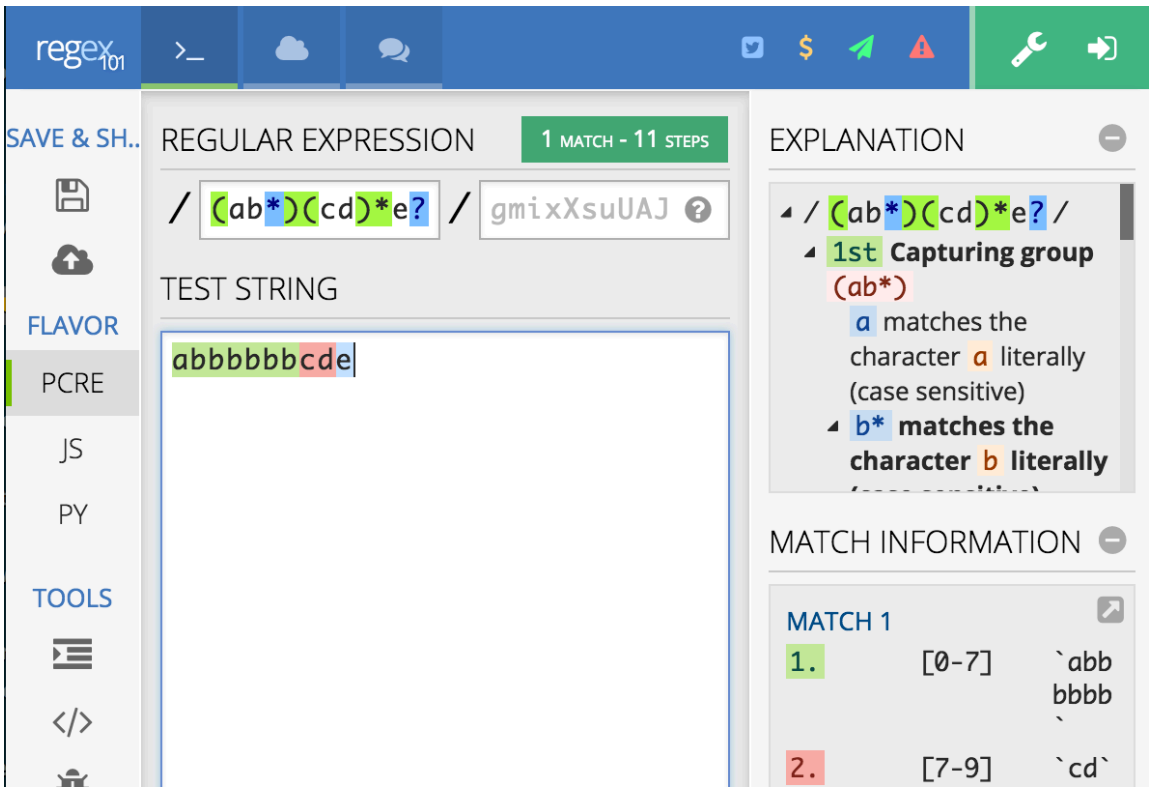
Task Frequencies are Mostly Consistent With Behavioral Categories

	Capturing	Counting Lines	Counting All	Finding	Filtering	Single Char	Parse User Input	Parse Gener-ated	Other
v. freq	1	1	1	3	0	0	2	2	0
freq.	9	2	3	7	1	0	5	1	1
occ.	3	5	4	3	8	1	5	4	0
rarely	5	3	3	4	2	3	3	3	0
v. rarely	0	3	4	1	5	5	3	5	1
never	0	4	3	0	2	9	0	3	16
avg	3.3	2.0	2.2	3.4	2.1	**0.8**	3	2.1	0.3

Developers said they did not frequently search for a single character.

Regex Testing

	Always	V. Freq	Freq.	Occ.	Rarely	V. Rarely	Never
test code	4	**7**	**5**	1	0	0	1
test regex	3	4	**5**	**5**	1	1	0



50% say they use testing tools like www.regex101.com

Usage Frequency - By Technical Environment

Heaviest regex use is in command line tools and text editors.

Language/Environment	**0**	**1-5**	**6-10**	**11-20**	**21-50**	**51+**
General (e.g., Java)	1	6	5	3	1	2
Scripting (e.g., Perl)	5	4	3	3	2	1
Query (e.g., SQL)	15	2	0	0	1	0
Command line (e.g., grep)	2	5	3	2	0	6
Text editor (e.g., IntelliJ)	2	5	0	5	1	5

Ephemeral vs Persistent Users

Task	Persistence Freq.	Ephemeral Freq.	Difference
Counting substrings that match a pattern	3	1.7	1.2
Parsing user input	3.6	2.7	0.9
Capturing parts of strings	3.8	3.1	0.7
Parsing generated text	2.4	1.9	0.5
Locating content within a file or files	3.6	3.2	0.4
Filtering collections (lists, tables, etc.)	2.2	1.9	0.3
Counting lines that match a pattern	1.8	2.1	-0.3

Code	Persistent Freq.	Ephemeral Freq.	Difference
LKA, NLKA, LKB, NLKB	3.2	2.2	1.0
LZY	3	2.8	0.2
STR, END	4.4	4.4	0
CG	4.2	4.2	0
WNW	3.4	3.5	-0.1

Pain Points

hard to compose (11)

...very difficult to write them since I've never read up on them.

...trickiness to getting the expression right

hard to read (7)

long ones can be hard to read

Readability. Edge cases.

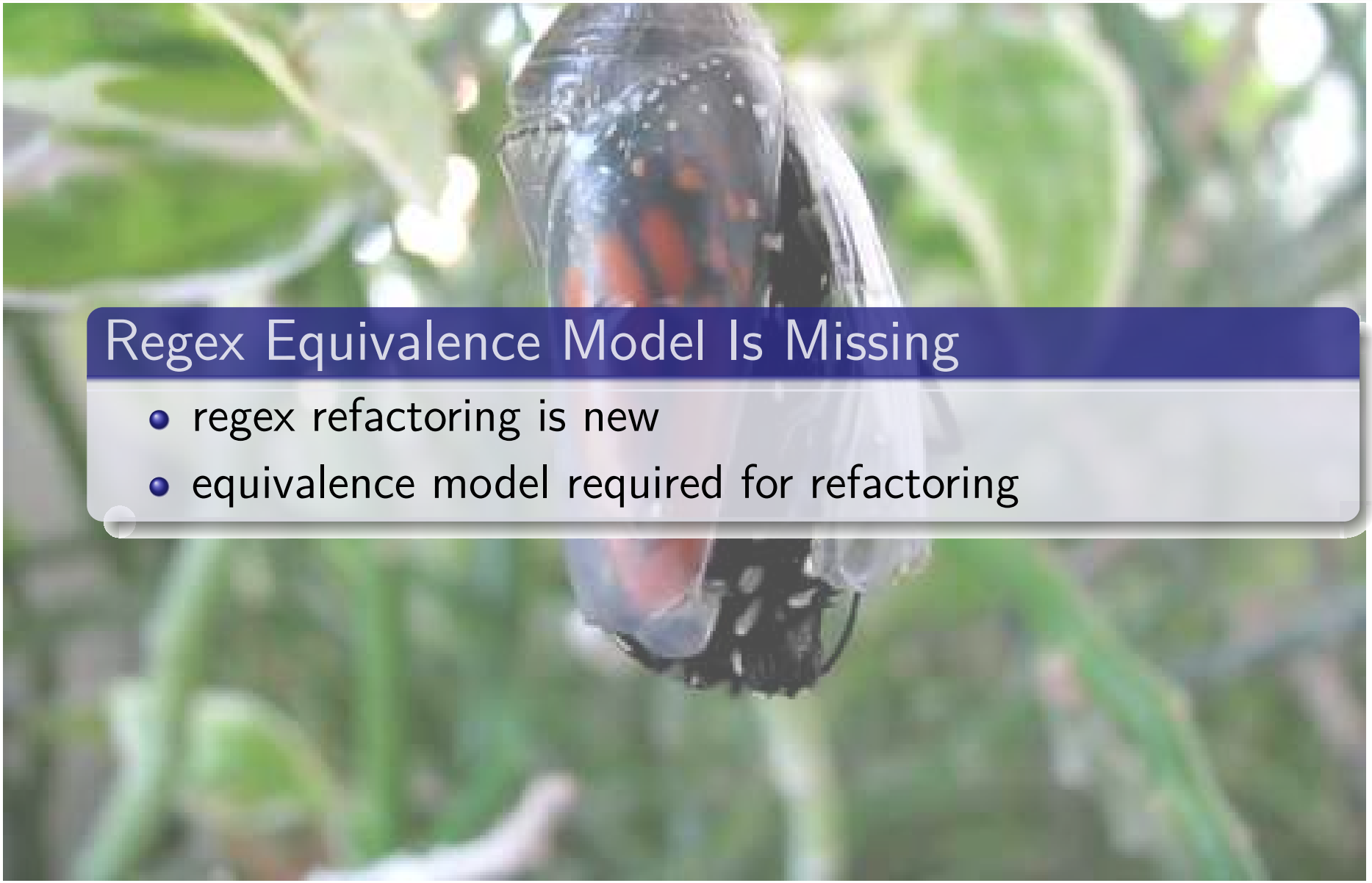
It is terrible to read (especially later after initial development)

inconsistency across implementations (3)

Differences in implementation across languages

Some regexes work differently (or don't work) in some languages.

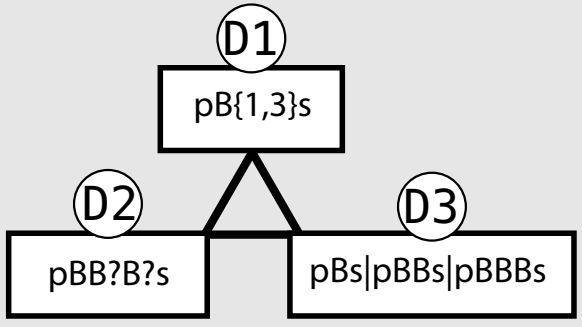
What Different Ways Can A Regex Be Represented?



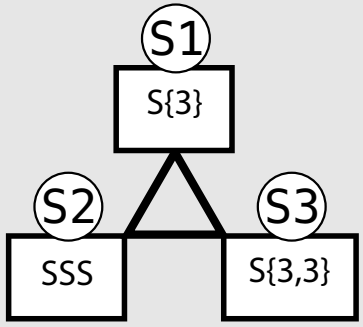
Regex Equivalence Model Is Missing

- regex refactoring is new
- equivalence model required for refactoring

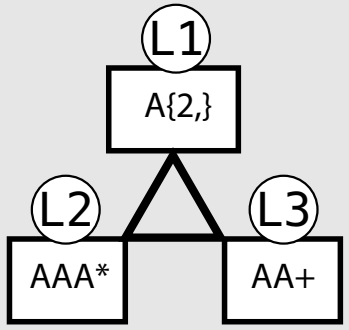
Regex Equivalence Classes



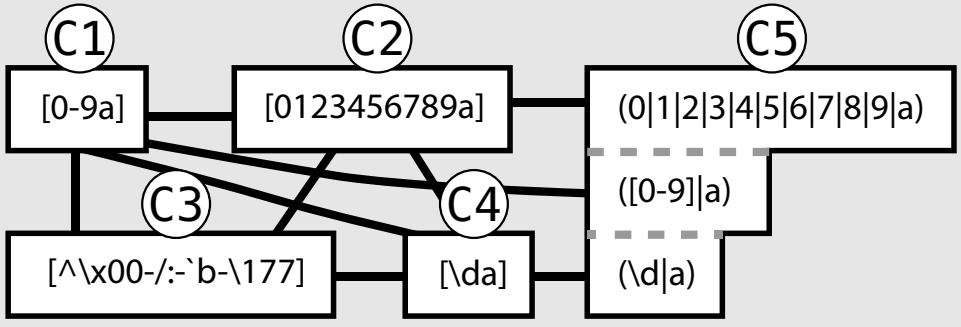
DBB GROUP
using the abstract `pB{1,3}s' where B is any pattern,
p and s are any (possibly empty) prefix, suffix



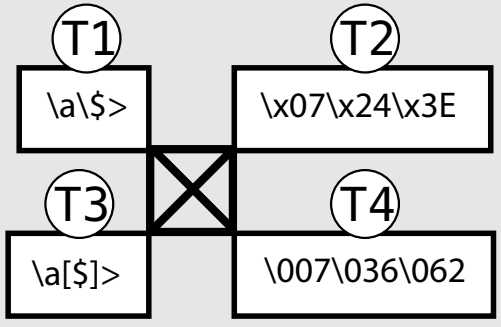
SNG GROUP
using the abstract `S{3}'
where S is any pattern



LWB GROUP
using the abstract `A{2,}'
where A is any pattern



CCC GROUP
using the concrete patern `[0-9a]' and assuming an ASCII charset



LIT GROUP
using the concrete pattern `\a\$>'
and assuming an ASCII charset

Example Equivalences

LIT : `[\072\073]` ≡ `[}{]`

DBB : `((q4f)?ab)` ≡ `(q4fab|ab)`

CCC : `tri[a-f]3` ≡ `tri(a|b|c|d|e|f)3`

LWB : `zaa*` ≡ `za+`

SNG : `[aeiou]{2}` ≡ `[aeiou][aeiou]`

How to decide which representation is preferred?

How Does The Community Prefer To Represent Regexes?

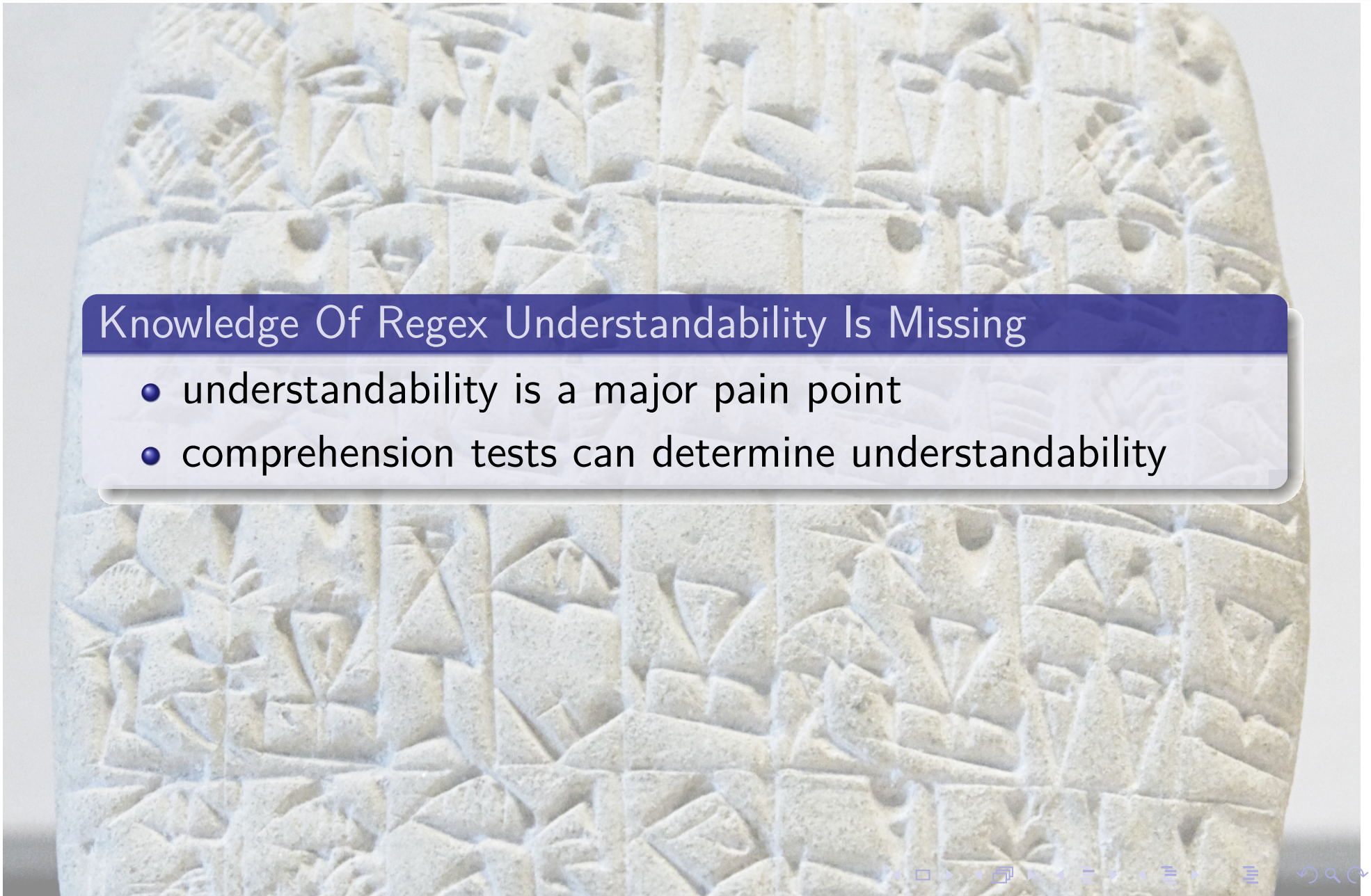
Knowledge Of Community Standards Is Missing

- programmers probably choose the best representation
- conformance to community standards eases comprehension

Node Membership Counts

Node	Description	Example	NRegexes	% Regexes	NProjects	% Projects
C1	CCC using RNG	`^[1-9][0-9]*$`	2,479	18.2%	810	52.5%
C2	CCC listing all chars	`[aeiouy]`	1,903	14.0%	715	46.3%
C3	any NCCC	`[^A-Za-z0-9.]+`	1,935	14.2%	776	50.3%
C4	CCC using defaults	`[-+\d.]`	840	6.2%	414	26.8%
C5	CCC as an OR	`(@|<|>|-|!)`	245	1.8%	239	15.5%
D1	repetition like {M,N}	`^x{1,4}$`	346	2.5%	234	15.2%
D2	zero-or-one repetition	`^http(s)?://`	1,871	13.8%	646	41.8%
D3	repetition using OR	`^(Q|QQ)\<(.+)\>$`	10	.1%	27	1.7%
T1	not in T2, T3 or T4	`get_tag`	12,482	91.8%	1,485	96.2%
T2	has HEX like \xF5	`[\x80-\xff]`	479	3.5%	243	15.7%
T3	wrapped chars like [$]	`([*]|[:])`	307	2.3%	268	17.4%
T4	has OCT like \0177	`[\041-\176]+:$`	14	.1%	37	2.4%
L1	repetition like {M,}	`(DN)[0-9]{4,}`	91	.7%	166	10.8%
L2	kleene star repetition	`\s*(#.*)?$`	6,017	44.3%	1,097	71.0%
L3	additional repetition	`[A-Z][a-z]+`	6,003	44.1%	1,207	78.2%
S1	repetition like {M}	`^[a-f0-9]{40}$`	581	4.3%	340	22.0%
S2	sequential repetition	`ff:ff:ff:ff:ff:ff`	3,378	24.8%	861	55.8%
S3	repetition like {M,M}	`U[\dA-F]{5,5}`	27	.2%	32	2.1%

What Representations Are More Understandable?

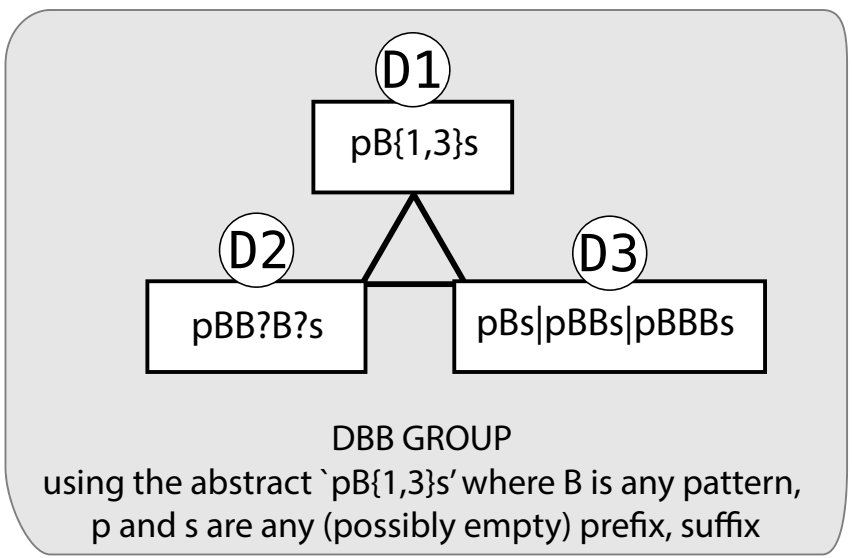


Knowledge Of Regex Understandability Is Missing

- understandability is a major pain point
- comprehension tests can determine understandability

Covering Edges

14 edges covered by 35 regex pairs
Matching challenge for each regex given to 30 participants



D1	`((q4f){0,1}ab)`
D2	`((q4f)?ab)`
D3	`(q4fab|ab)`
TRUE	`"ab"`
FALSE	`"fq4f"`
TRUE	`"xyzq4fab"`
TRUE	`"zlmab"`
FALSE	`"qfa4"`

Testing Comprehension

180 participants answered one composition and five string matching questions for 10 regexes: 10,800 data points total.

Subtask 7. Regex Pattern: `'((q4f)?ab)'`

7.A	`'qfa4'`	○ matches	● not a match	○ unsure
7.B	`'fq4f'`	○ matches	● not a match	○ unsure
7.C	`'zlmab'`	○ matches	○ not a match	● unsure
7.D	`'ab'`	○ matches	○ not a match	● unsure
7.E	`'xyzq4fab'`	● matches	○ not a match	○ unsure

7.F Compose your own string that contains a match: 4q4fab

Combined Edge Info

Index	Nodes	Pairs	Match1	Match2	H_0^{match}	Compose1	Compose2	H_0^{comp}
E1	T1 – T4	2	80%	60%	0.001	87%	37%	$<$**0.001**
E2	D2 – D3	2	78%	87%	**0.011**	88%	97%	0.085
E3	C2 – C5	4	85%	86%	0.602	88%	95%	**0.063**
E4	C2 – C4	1	83%	92%	**0.075**	60%	67%	0.601
E5	L2 – L3	2	86%	91%	0.118	97%	100%	0.159
E6	D1 – D2	2	84%	78%	0.120	93%	88%	0.347
E7	C1 – C2	2	94%	90%	0.121	93%	90%	0.514
E8	T2 – T4	2	84%	81%	0.498	65%	52%	0.141
E9	C1 – C5	2	94%	90%	0.287	93%	93%	1.000
E10	T1 – T3	3	88%	86%	0.320	72%	76%	0.613
E11	D1 – D3	2	84%	87%	0.349	93%	97%	0.408
E12	C1 – C4	6	87%	84%	0.352	86%	83%	0.465
E13	C3 – C4	2	61%	67%	0.593	75%	82%	0.379
E14	S1 – S2	3	85%	86%	0.776	88%	90%	0.638

Four Significant Understandability Differences, $\alpha = 0.1$

Code	Regex	Match	Compose	Refactoring	Node	Regex	Match	Compose
T4	`([\072\073])`	66%	50%	$\overrightarrow{T4T1}$	T1	`([:;])`	81%	87%
T4	`([\0175\0173])`	54%	23%		T1	`([}{])`	79%	87%
D2	`((q4f)?ab)`	79%	83%	$\overrightarrow{D2D3}$	D3	`(q4fab|ab)`	85%	97%
D2	`(deedo(do)?)`	77%	93%		D3	`(deedo|deedodo)`	90%	97%
C2	`([:;])`	81%	87%	$\overrightarrow{C2C5}$	C5	`(:|;)`	94%	100%
C2	`no[wxyz]5`	87%	90%		C5	`no(w|x|y|z)5`	94%	97%
C2	`([}{])`	79%	87%		C5	`(\{|\})`	70%	93%
C2	`tri[abcdef]3`	93%	90%		C5	`tri(a|b|c|d|e|f)3`	86%	90%
C2	`[\t\r\f\n ]`	83%	60%	$\overrightarrow{C2C4}$	C4	`[\s]`	92%	67%

Four Noteworthy Trends

Node	Regex	Match	Compose	Ref.	Node	Regex	Match	Compose
L2	`zaa*`	87%	97%	$\overrightarrow{L2L3}$	L3	`za+`	91%	100%
L2	`RR*`	86%	97%		L3	`R+`	92%	100%
D2	`((q4f)?ab)`	79%	83%	$\overrightarrow{D2D1}$	D1	`((q4f){0,1}ab)`	83%	97%
D2	`(deedo(do)?)`	77%	93%		D1	`(dee(do){1,2})`	85%	90%
C2	`tri[abcdef]3`	93%	90%	$\overrightarrow{C2C1}$	C1	`tri[a-f]3`	94%	97%
C2	`no[wxyz]5`	87%	90%		C1	`no[w-z]5`	93%	90%
T4	`xyz[\0133-\0140]`	71%	33%	$\overrightarrow{T4T2}$	T2	`xyz[\x5b-\x5f]`	79%	60%
T4	`t[\072-\073]+p`	90%	70%		T2	`t[\x3a-\x3b]+p`	89%	70%

Detailed Refactoring Recommendations

LIT Group

$\overrightarrow{T4T1}$ by understandability (E1) and community.

Example: `[\072\073]` ⇒ `[:;]`

$\overrightarrow{T4T2}$ by community, and trends in understandability (E8)

Example: `[\072\073]` ⇒ `[\x3A\x3B]`

$\overrightarrow{T3T1}$ by community

Example: `[^]` ⇒ `\^`

$\overrightarrow{T2T1}$ by community

Example: `[\x3A\x3B]` ⇒ `[:;]`

CCC Group

$\overrightarrow{C2C1}$ by community, and trends in understandability (E7).

Example: `[abcdef]` ⇒ `[a-f]`

$\overrightarrow{C4C1}$, $\overrightarrow{C5C1}$ by community.

Example: `[\w]` ⇒ `[0-9a-zA-Z_]`

Example: `(0|1|2|3|4|5)` ⇒ `[0-5]`

$\overrightarrow{C4C2}$ for community standards.

Example: `[\s]` ⇒ `[\t\n\r\c\v ]`

$\overrightarrow{C2C4}$ for understandability (E4).

Example: `[\t\n\r\c\v ]` ⇒ `[\s]`

$\overrightarrow{C5C2}$ for community standards.

Example: `(a|e|i|o|u)` ⇒ `[aeiou]`

$\overrightarrow{C2C5}$ for understandability (E3).

Example: `[aeiou]` ⇒ `(a|e|i|o|u)`

DBB Group

$\overrightarrow{D2D3}$ for understandability (E2).

Example: `(ab)?c` ⇒ `abc|c`

$\overrightarrow{D3D2}$ for community.

Example: `abc|c` ⇒ `(ab)?c`

$\overrightarrow{D2D1}$ for understandability (E6).

Example: `(ab)?c` ⇒ `(ab){0,1}c`

$\overrightarrow{D1D2}$ for community.

Example: `(ab){0,1}c` ⇒ `(ab)?c`

$\overrightarrow{D3D1}$ for community.

Example: `abc|c` ⇒ `(ab){0,1}c`

LWB, SNG Groups

$\overrightarrow{L1L2}$ by community standard.

Example: `[a\d]{2,}` ⇒ `[a\d][a\d][a\d]*`

$\overrightarrow{L2L3}$ for understandability (E5).

Example: `ZZ*` ⇒ `Z+`

SNG community standard is complicated by double-letters in S2.
SNG understandability differences were not significant, only S1 – S2 measured.

Regex Refactoring

- Testing and considering more edges in isolation
- being cognizant of number of repetitions
- size of the repeated element, and size before/after
- for CCC: limitations due to defaults, ranges and invisible characters
-

References

John Smith (2012) Title of the publication Journal Name 12(3), 45 – 678.

Questions?