Extending the Identification of Object-Oriented Variability Implementations using Usage Relationships

Johann Mortara[1] Xhevahire Tërnava[2] Philippe Collet[1]
Anne-Marie Dery-Pinna[1]

[1]Université Côte d'Azur, CNRS, I3S, France
[2]Université de Rennes 1, INRIA/IRISA, France

SPLC, REVE, September 07, 2021

Context

- Most modern object-oriented systems are variability-rich
  - their variability is hardly documented or made explicit in code
  - lack of approaches on identifying their variability
  - lack of approaches on representing (visualizing) their variability

Context

- Most modern object-oriented systems are variability-rich
  - ⊛ their variability is hardly documented or made explicit in code
  - ⊛ lack of approaches on identifying their variability
  - ⊛ lack of approaches on representing (visualizing) their variability

⇒ *symfinder*: a tooled approach (Java [1] and C++ [2])

- ⊛ Identifying variability implementation places (Variation Points with Variants) in single codebase object-oriented systems
  - * based on the property of symmetry in 7 traditional techniques

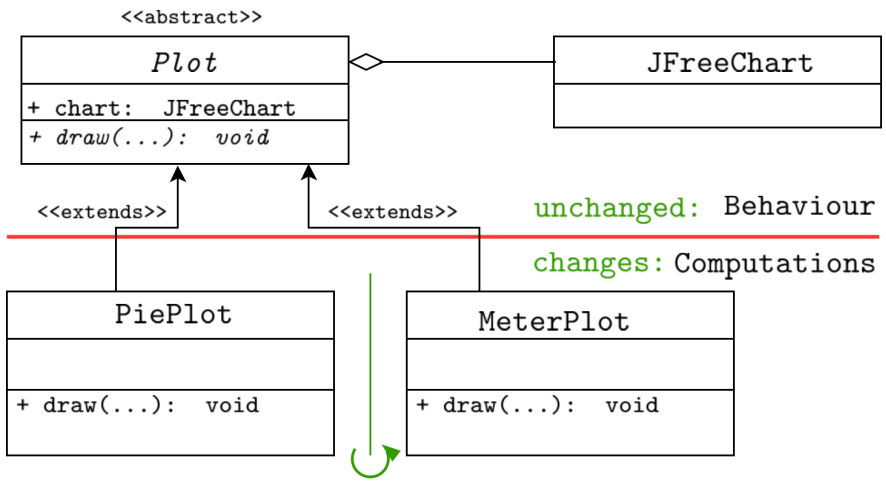
- ⊛ Visualizing of Variation Points (VPs) with Variants
  - * based on their density

[1] https://doi.org/10.1145/3336294.3336311

[2] https://doi.org/10.1145/3382026.3431251

Example of symmetry in object-oriented techniques

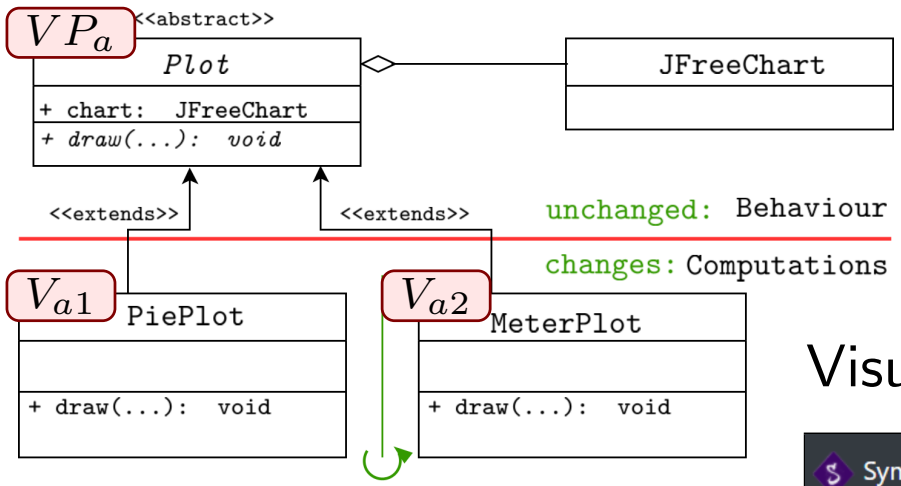
Variability in JFreeChart:



- Most of the traditional techniques can be interpreted in terms of symmetry

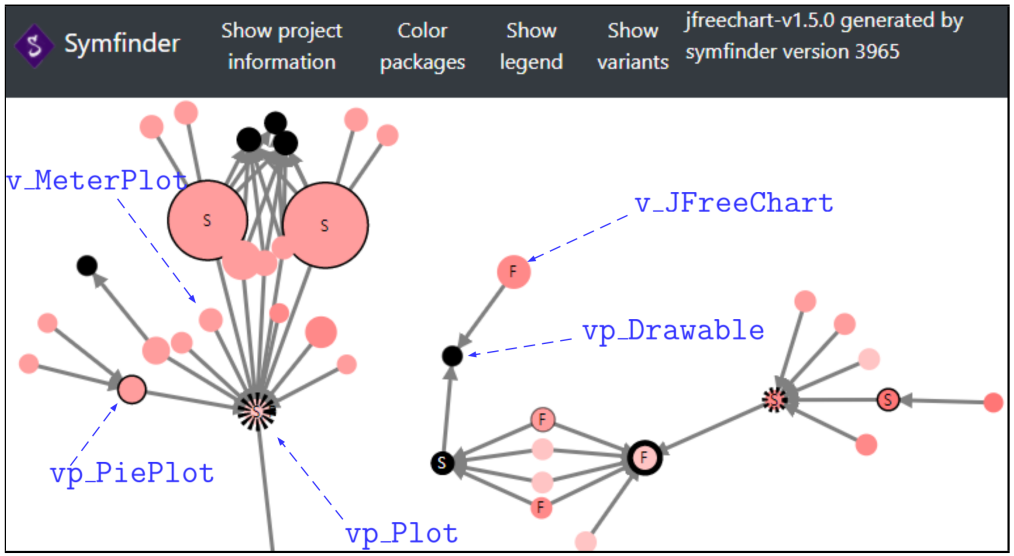
Example of symmetry in object-oriented techniques

Variability in JFreeChart:



- Most of the traditional techniques can be interpreted in terms of symmetry

Visualization by *symfinder*



Problem statement

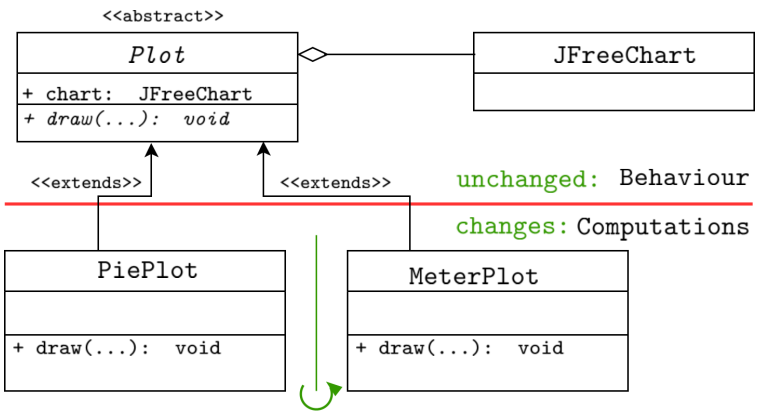
symfinder: Applied: $>$ 15 real open-source systems

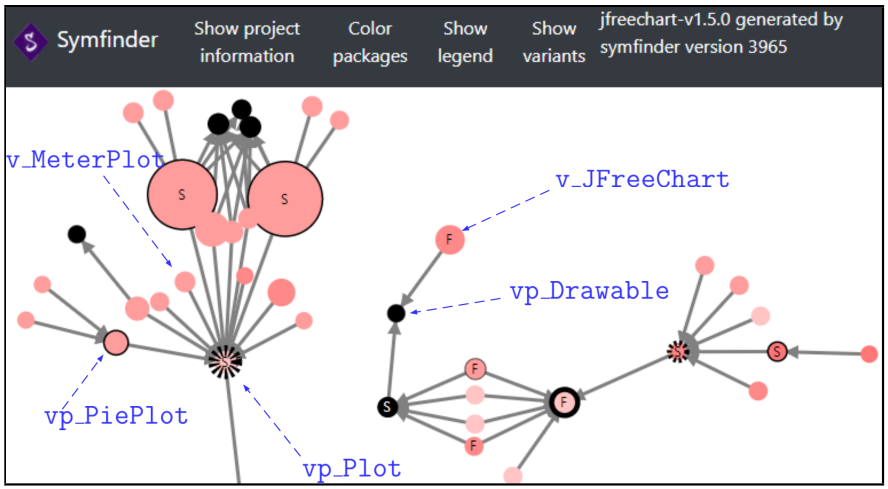
- Identified: 200 - 11K potential VPs with Variants
- Precision: potential $\Rightarrow$ real VPs with Variants

Issue 1: Identifying inheritance relationships is not enough

- Composition of instances is not taken into account

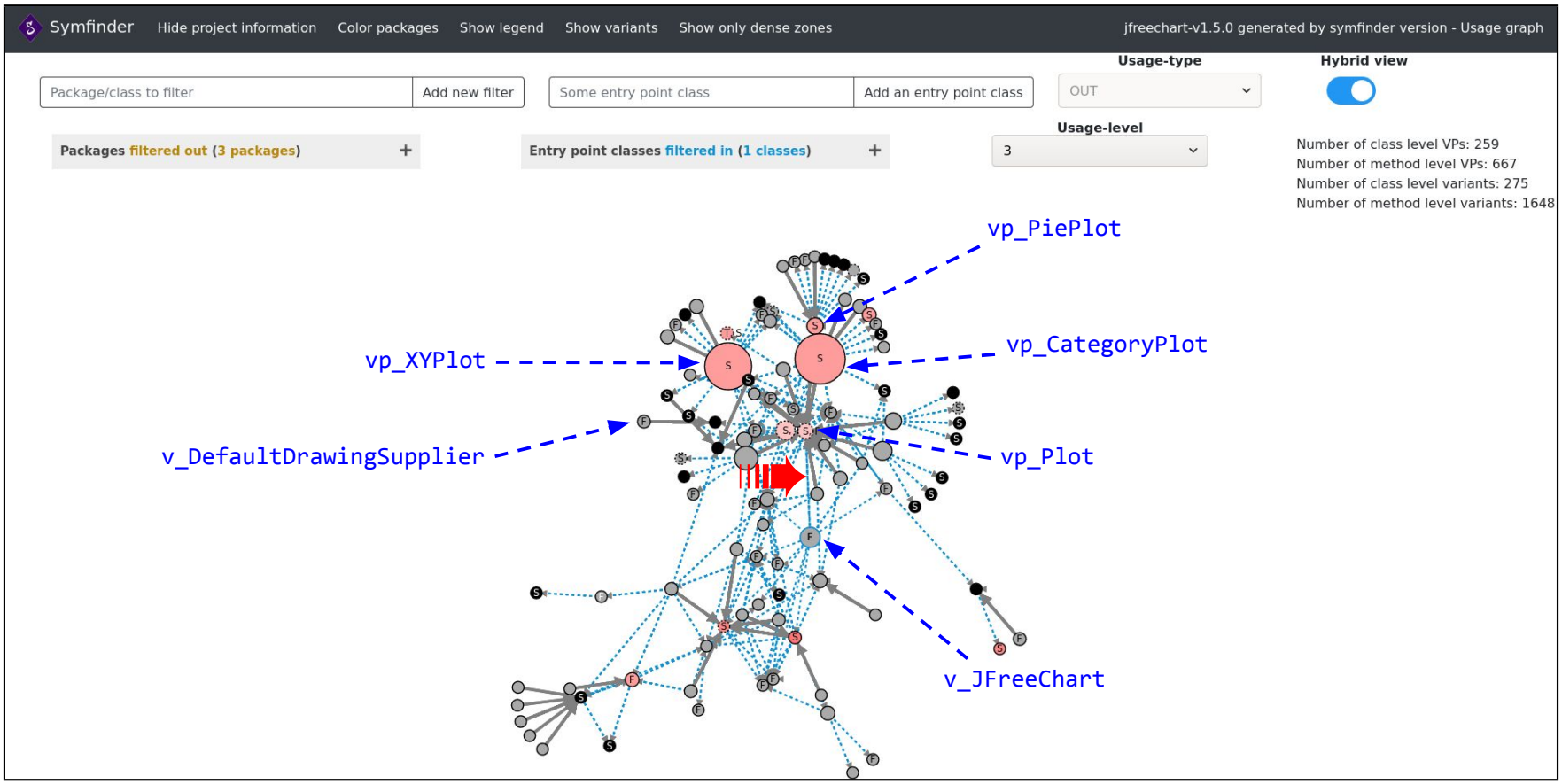
Issue 2: Entry points are missing: for browsing the visualization





symfinder-2

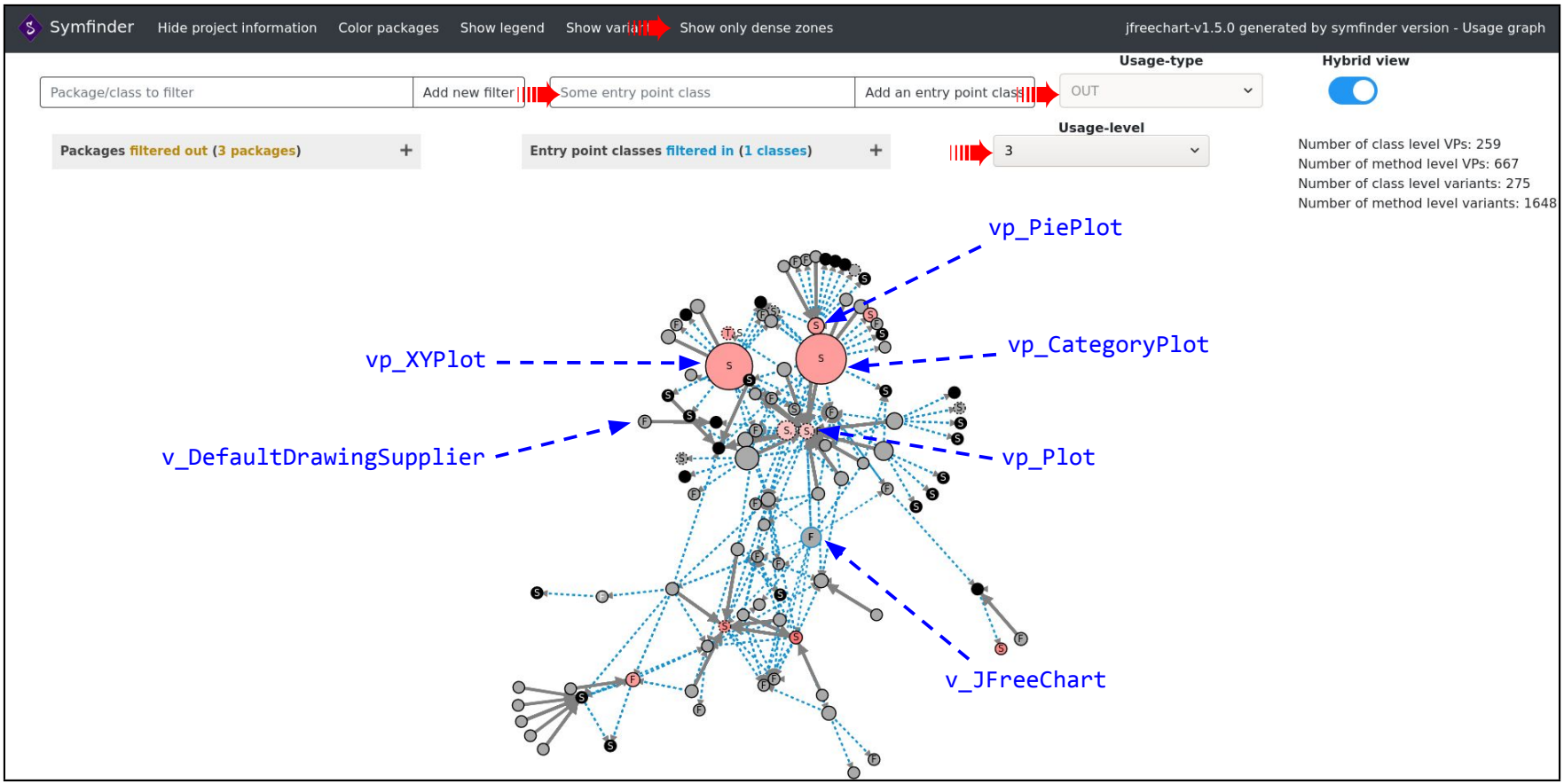
Extension 1: handling usage relationships (+7 variability implementation techniques)



symfinder-2

Extension 2: handling the entry points

- 4 user-defined entry points (➡)
- Automatically defined entry points (using the system's API)



Evaluation

- In 10 open-source Java-based systems, regarding 4 research questions

Subject system	Commit	LoC	# *vp*-s	# *variants*	API / Type
Java AWT	3319fcb	69,974	795	1,706	D / L
Apache CXF	4da7b71	48,655	3,403	7,625	D / F
JUnit	60aaf96	7,717	109	245	D / F
Maven	97c98ec	105,342	612	1,147	D / A
JFreeChart	1f6a91f	94,384	926	1,923	D / L
ArgoUML	d135342	134,367	776	1,959	D / A
Cucumber	323f724	42,662	238	282	A / F
Logbook	f0f36e7	16,210	96	162	A / L
Riptide	48b03a7	12,626	102	218	A / L
NetBeans	cade258	5,058,448	3,621	6,736	D / A

Evaluation

RQ_1: Does the identification of usage relationships have changed the variability visualization of a given system by symfinder-2?

Subject	Nodes	*symfinder*		*symfinder-2*	
		Graphs	Isolated	Graphs	Isolated
Java AWT	431	55	142	2	20
Apache CXF	3085	473	1149	105	500
JUnit	118	23	36	6	18
Maven	616	177	172	21	79
JFreeChart	578	54	167	5	51
ArgoUML	1270	123	460	38	183
Cucumber	331	45	122	14	50
Logbook	117	19	40	4	16
Riptide	89	20	37	8	19
NetBeans	3498	504	1666	195	836

- Same variability; denser places with variability; less isolated nodes

Evaluation

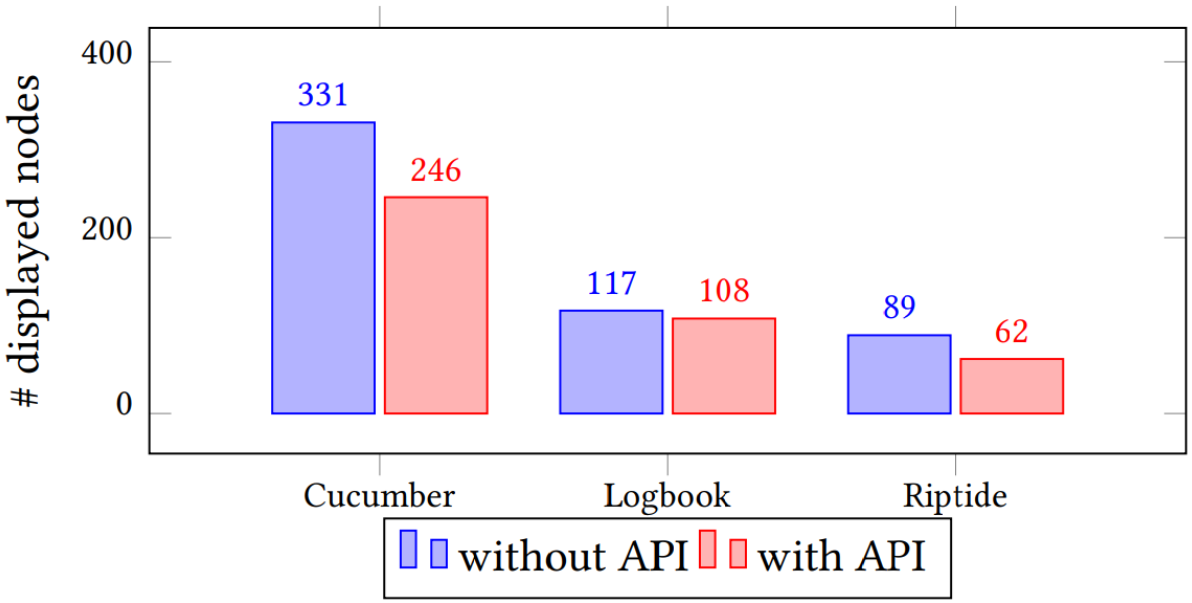
RQ_2: What is the starting density threshold to begin with the comprehension of the visualized variability by symfinder-2?

Project	*symfinder* nodes	*symfinder-2* ≥ 5 *v*-s ≤ 3 hops	*symfinder-2* ≥ 10 *v*-s ≤ 3 hops	*symfinder-2* ≥ 30 *v*-s ≤ 2 hops
Java AWT	431	28	22	3
Apache CXF	3086	98	32	4
JUnit	118	5	0	0
Maven	616	8	1	0
JFreeChart	578	34	15	3
ArgoUML	1258	40	15	3
Cucumber	331	4	0	0
Logbook	117	0	0	0
Riptide	89	0	0	0
NetBeans	3494	58	22	2

- The threshold parameters depend on the studied system; First threshold can be used as a good starting point

Evaluation

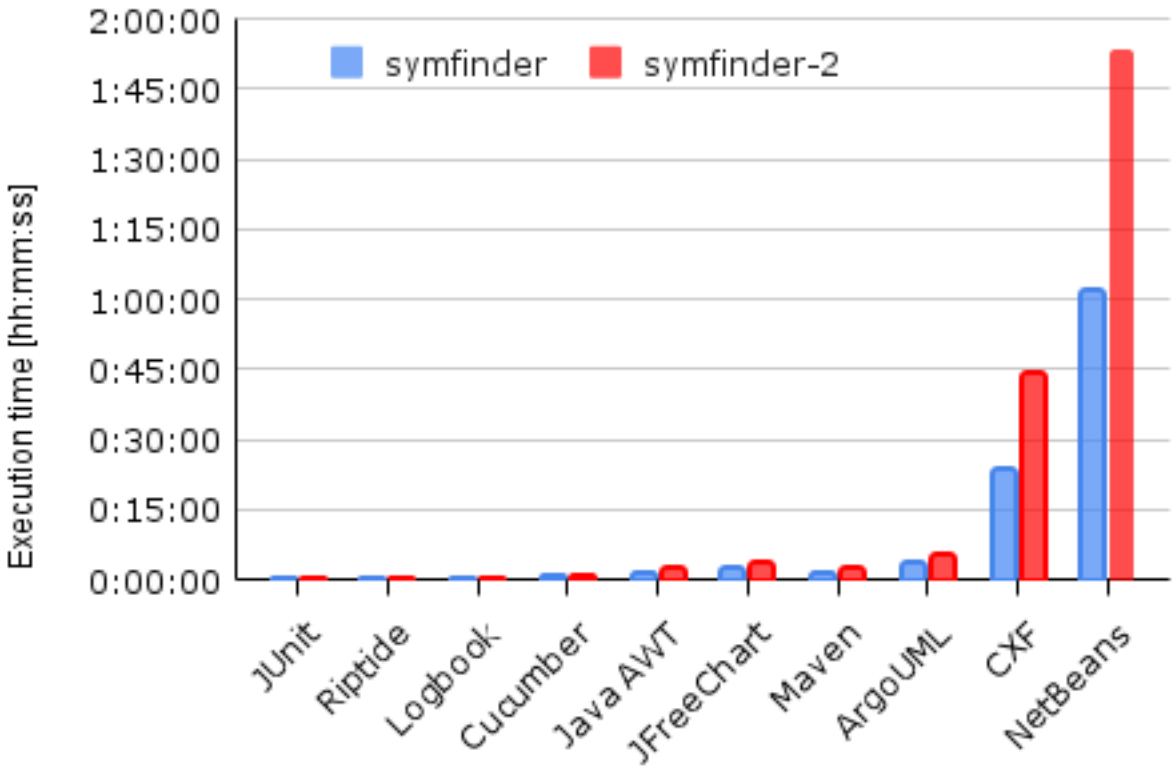
RQ_3: Is the API information of a given system useful to simplify its identified variability by symfinder-2?



- Potential VPs with variants ⇒ relevant variability places; Integrate different variability information sources

Evaluation

RQ_4: Does the identification of usage relationships impact the scalability of symfinder-2?



- Visualization: 700ms in Chrome, 850ms in Firefox for NetBeans
- The time difference is increased with the size of analysed system

Summary

- A tooled approach for I&V potential VPs with Variants in OO systems, with a single code base, implemented in Java
- Extended symfinder $\Rightarrow$ symfinder2
- It identifies the impl. variability by 8 traditional techniques
- It provides 5 entry points for variability comprehension
- Propose to integrate different variability sources

Availability:

https://deathstar3.github.io/symfinder2-demo/