Model-Driven Optimization:

Generating Smart Mutation Operators for Multi-Objective Problems

Niels van Harten
Radboud University Nijmegen
Nijmegen, The Netherlands
niels@vharten.com

CDN (**Diego**) Damasceno
Radboud University Nijmegen
Nijmegen, The Netherlands
https://damascenodiego.github.io/
d.damasceno@cs.ru.nl

Daniel Strüber
Chalmers | University of Gothenburg (SE)
Radboud University Nijmegen (NL)
https://www.danielstrueber.de/
danstru@chalmers.se

@damascenodiego

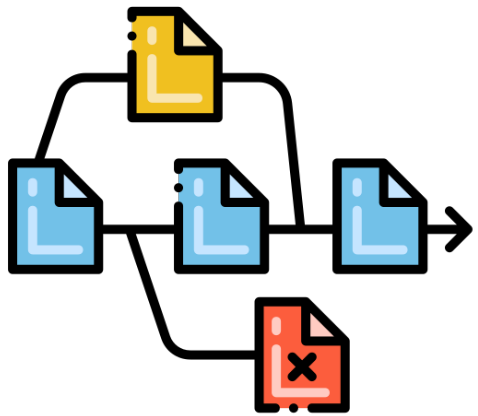
Euromicro DSD/SEAA 2022
Aug. 31 – Sep. 2, 2022

Radboud University

Optimization problems are at the heart of many software engineering tasks

Architecture refactoring

Components deployment

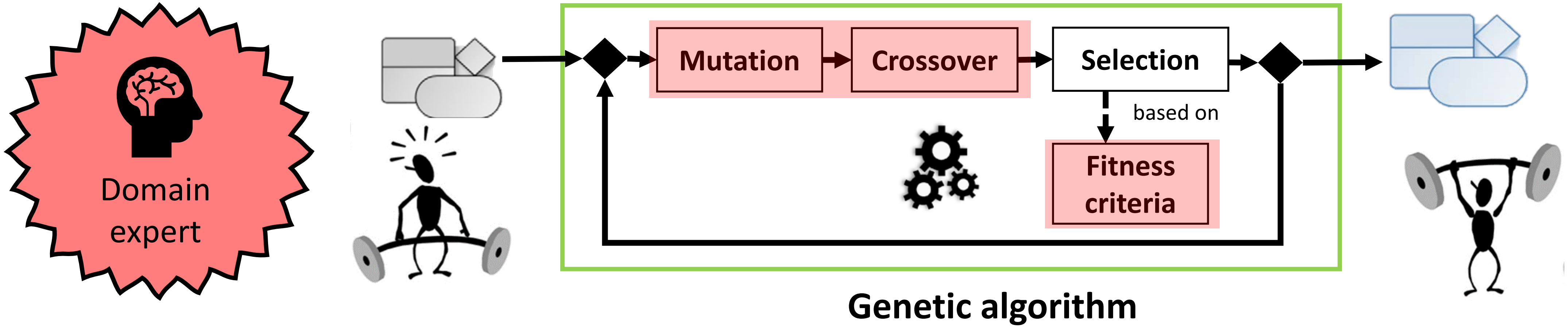


Release planning

Solutions	Assignment Classes → Packages	Assignment Components → Hosts	Assignment Next Release → Artifacts
Optimality	max. Cohesion min. Coupling	min. Price min. Overhead	min. Development Cost max. Customer Satisfaction

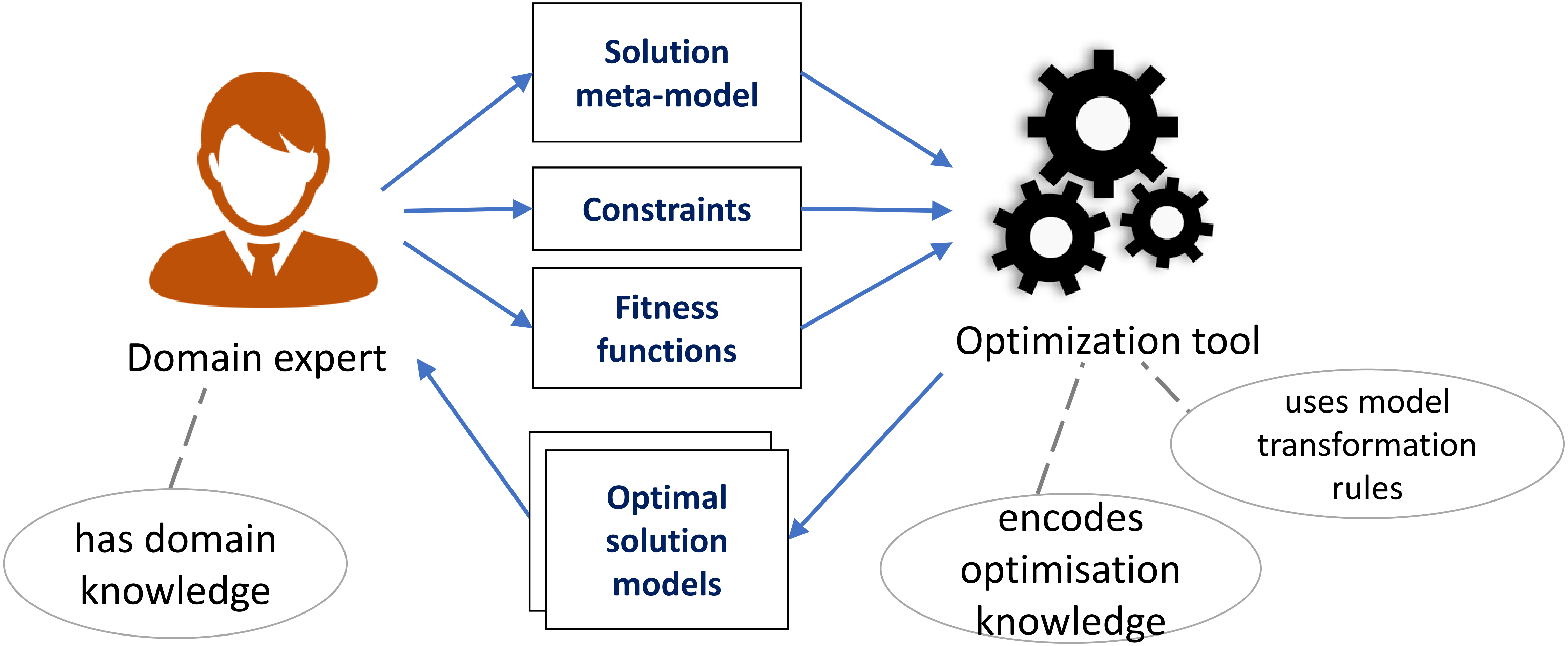
Search-based software engineering (SBSE)

- **Problem**: Search space usually too large to enumerate all solutions
- **Approach**: *Search guided by principles from the evolutionary theory*

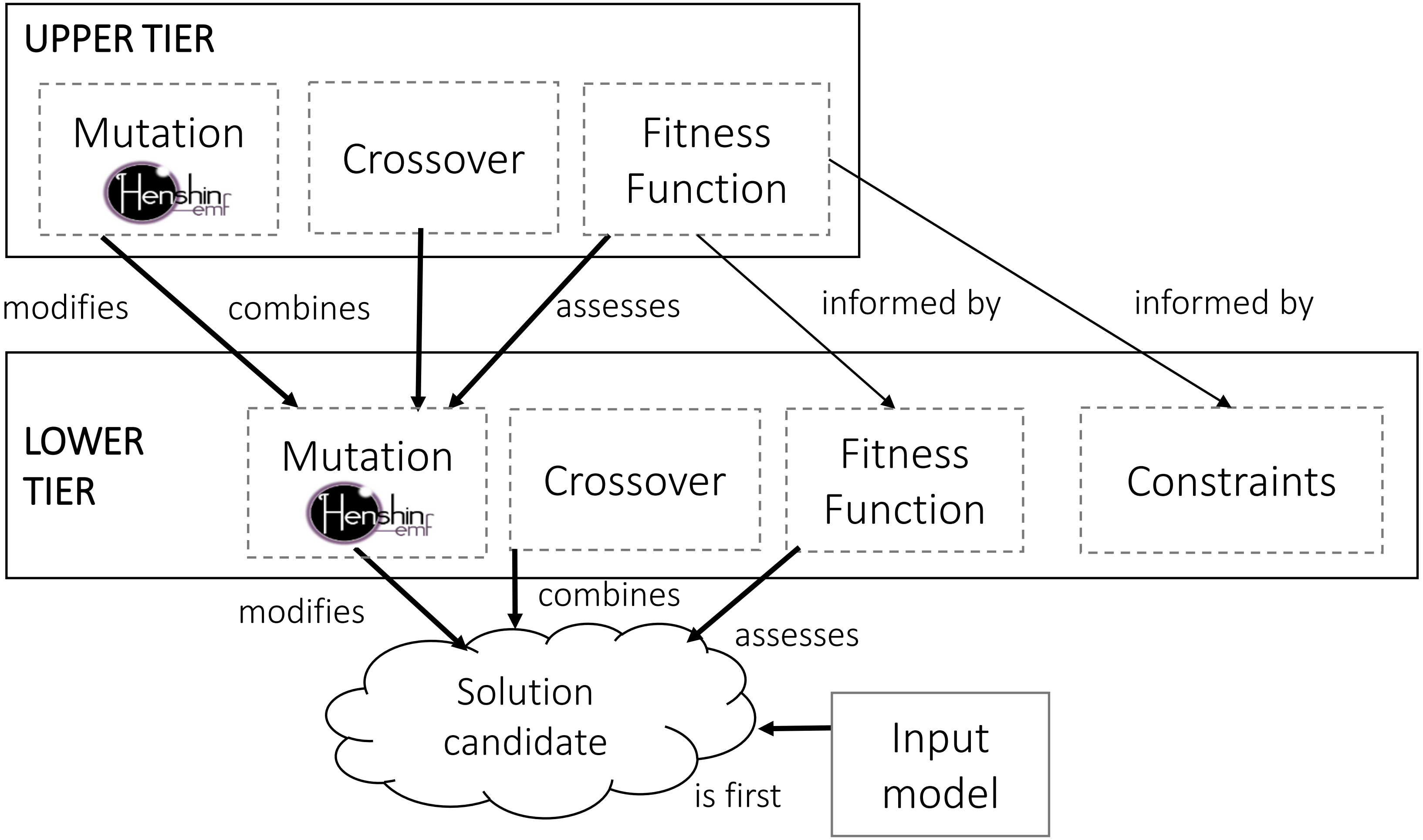


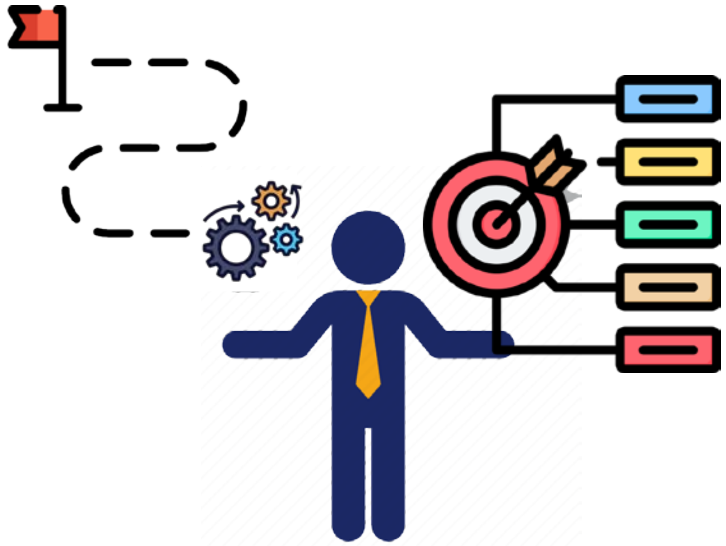
- **Challenge**: Search algorithms need to be custom-tailored to the problem at hand

Model-Driven Optimization



FitnessStudio: A two-tier framework of nested genetic algorithms





Limitations:

1. Relies on a **single-objective** genetic algorithm on both tiers
2. Mutation operators are generated: **from scratch** and **without additional user input**

Our Contribution

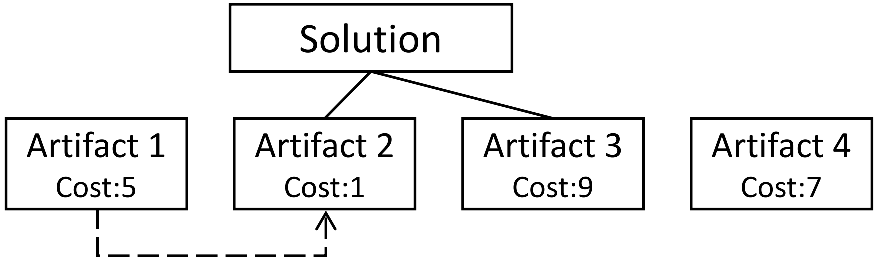
A model-driven optimization technique to **automatically generate** efficient **mutation operators** with support for:

1. **Multiple** optimization criteria
2. User-provided **domain knowledge**

Model Driven-Optimization

Model-Driven Optimization

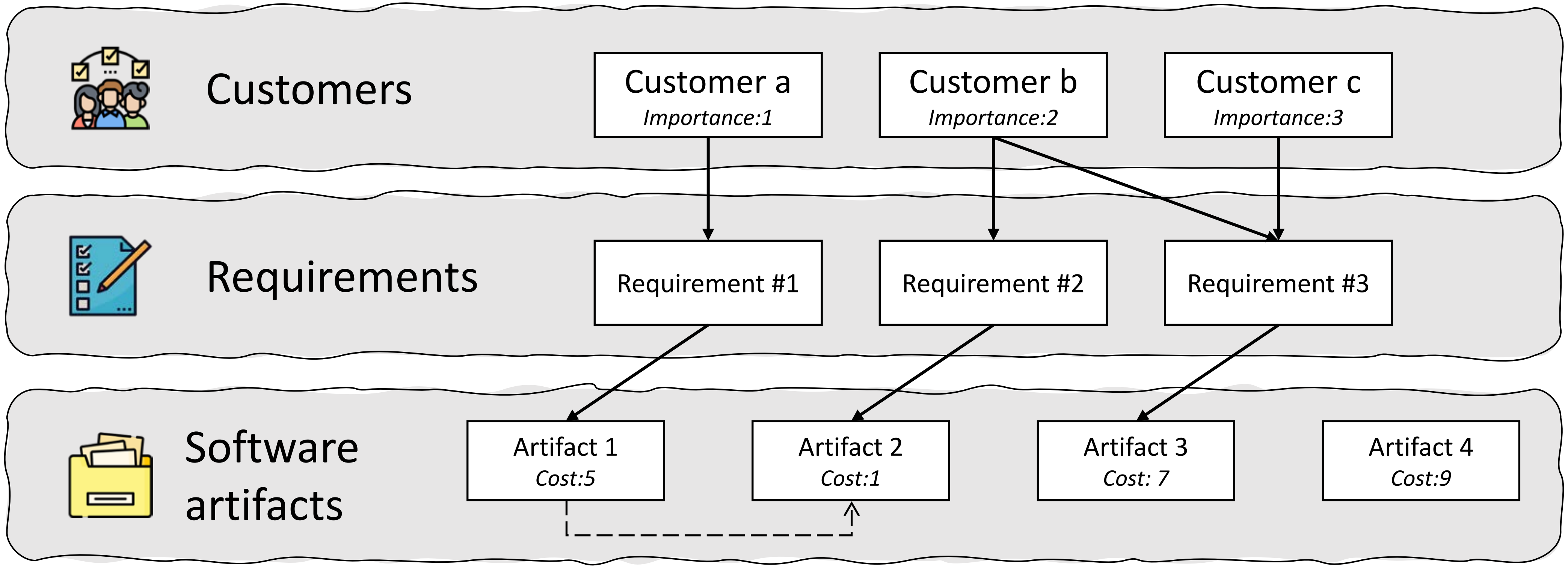
- Model-driven principles to bridge the abstraction gap between:
  - The low-level SBSE implementations
  - Declarative specifications of optimization problems (e.g., models)
    - Graph-transformations languages (e.g., Henshin)



Model-based encoding

Henshin: Japanese for *Transformation*
https://www.eclipse.org/henshin/

The Next Release Problem

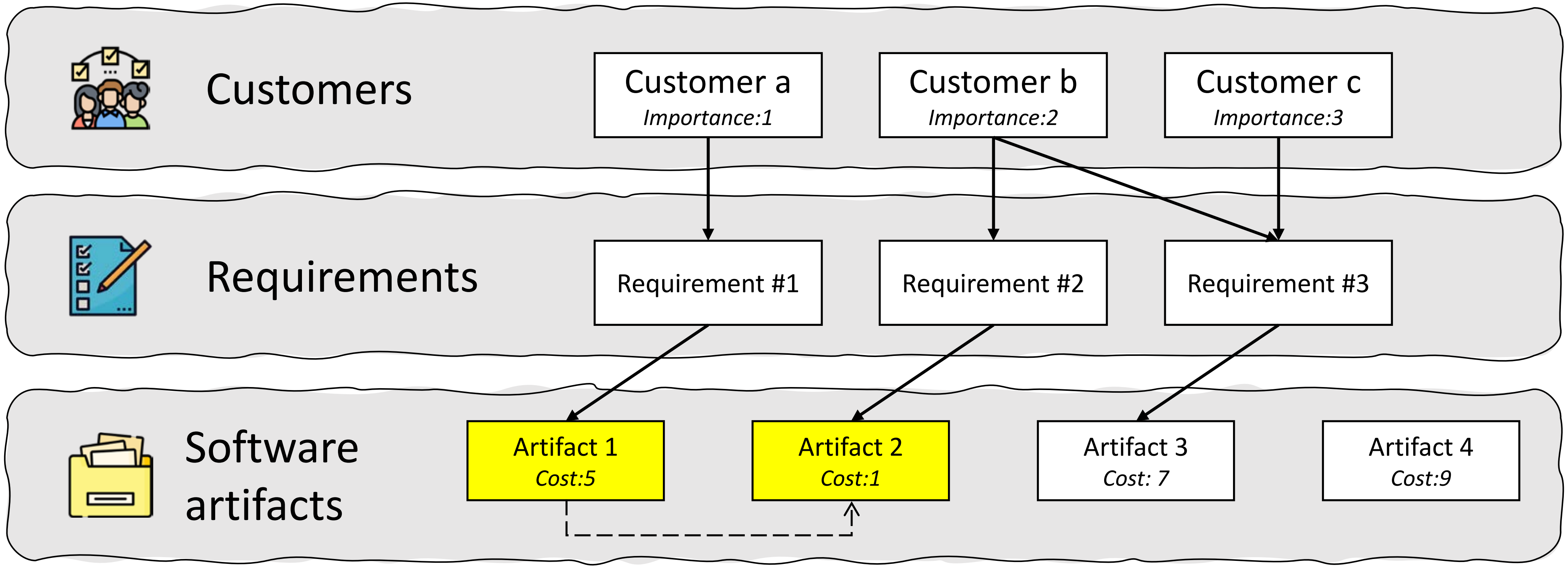


Find the “optimal” selection of artifacts w.r.t.

$$Cost = \sum_{sa \in SA'} cost(sa)$$

$$Satisfaction = \sum_{c \in C} importance(c) \cdot satisfaction(c)$$

The Next Release Problem

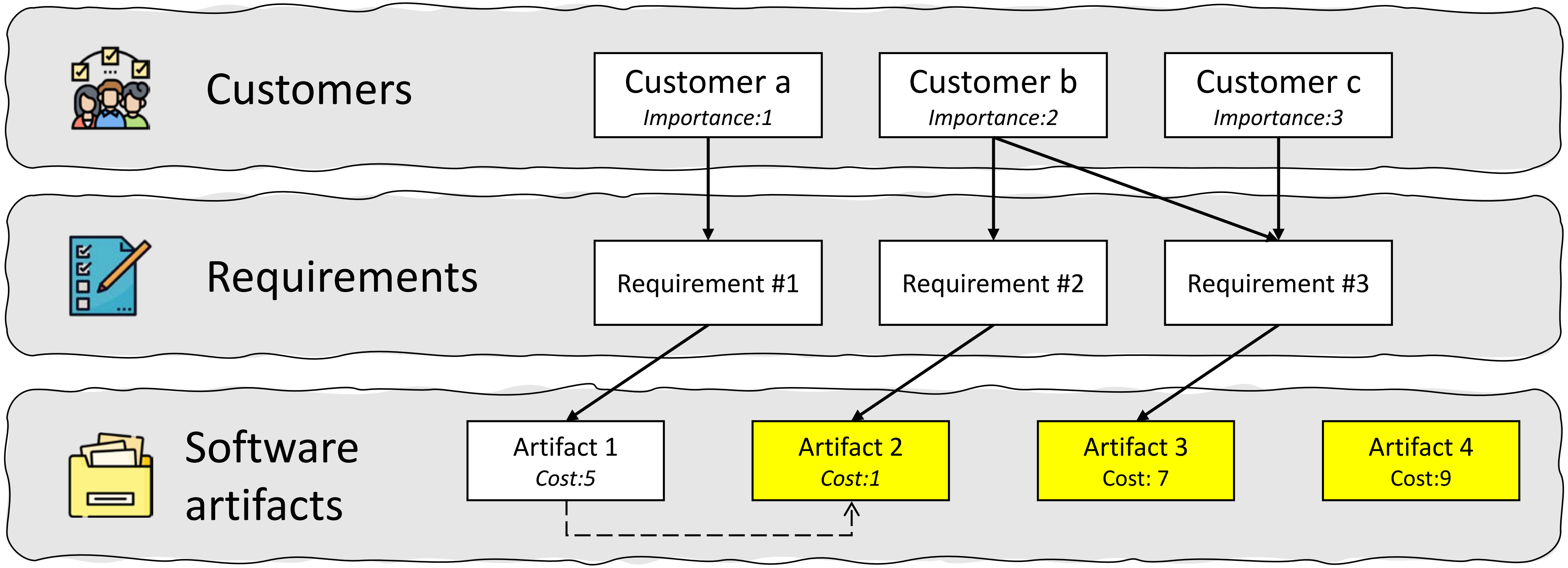


Example of solution: Release artifacts 1, 2

Cost = 5+1 = 6

Satisfaction = 1*100% + 2*50% + 3*0% = 2

The Next Release Problem



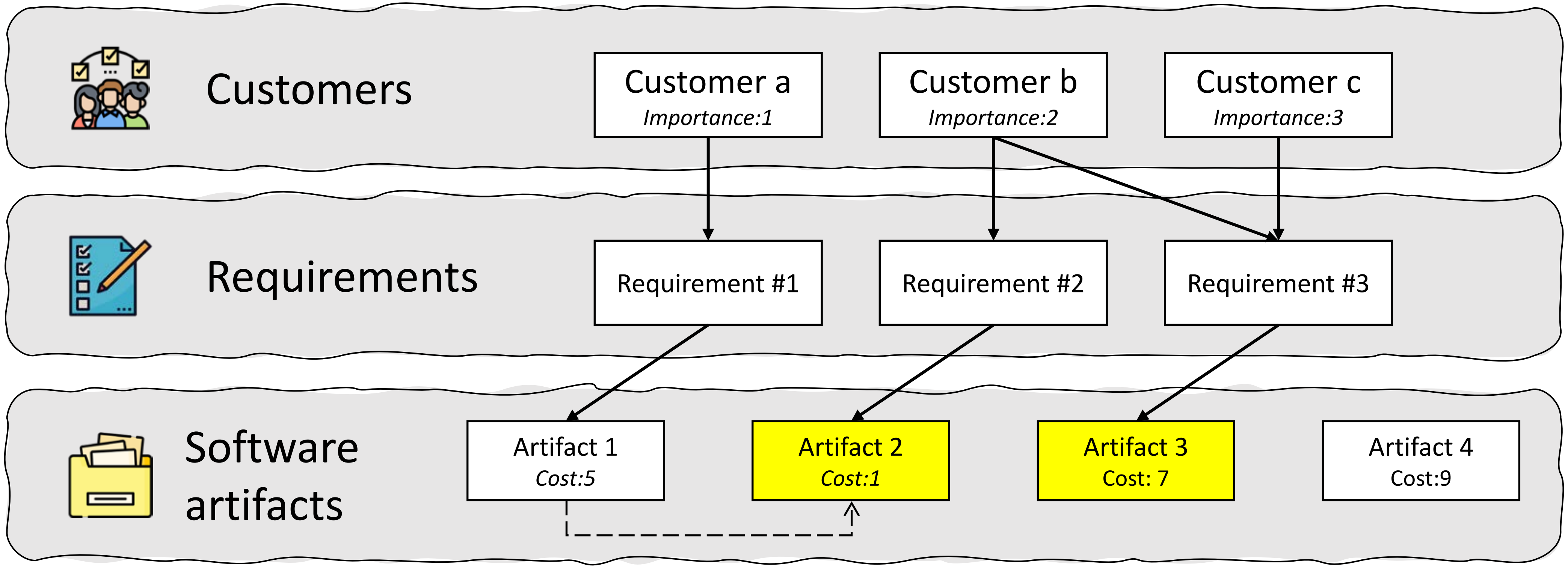
Example of solution: Release artifacts 2, 3, 4

Cost = 1+7+9 = 17

Satisfaction = 1*0% + 2*100% + 3*100% = 5

Can we do better?

The Next Release Problem



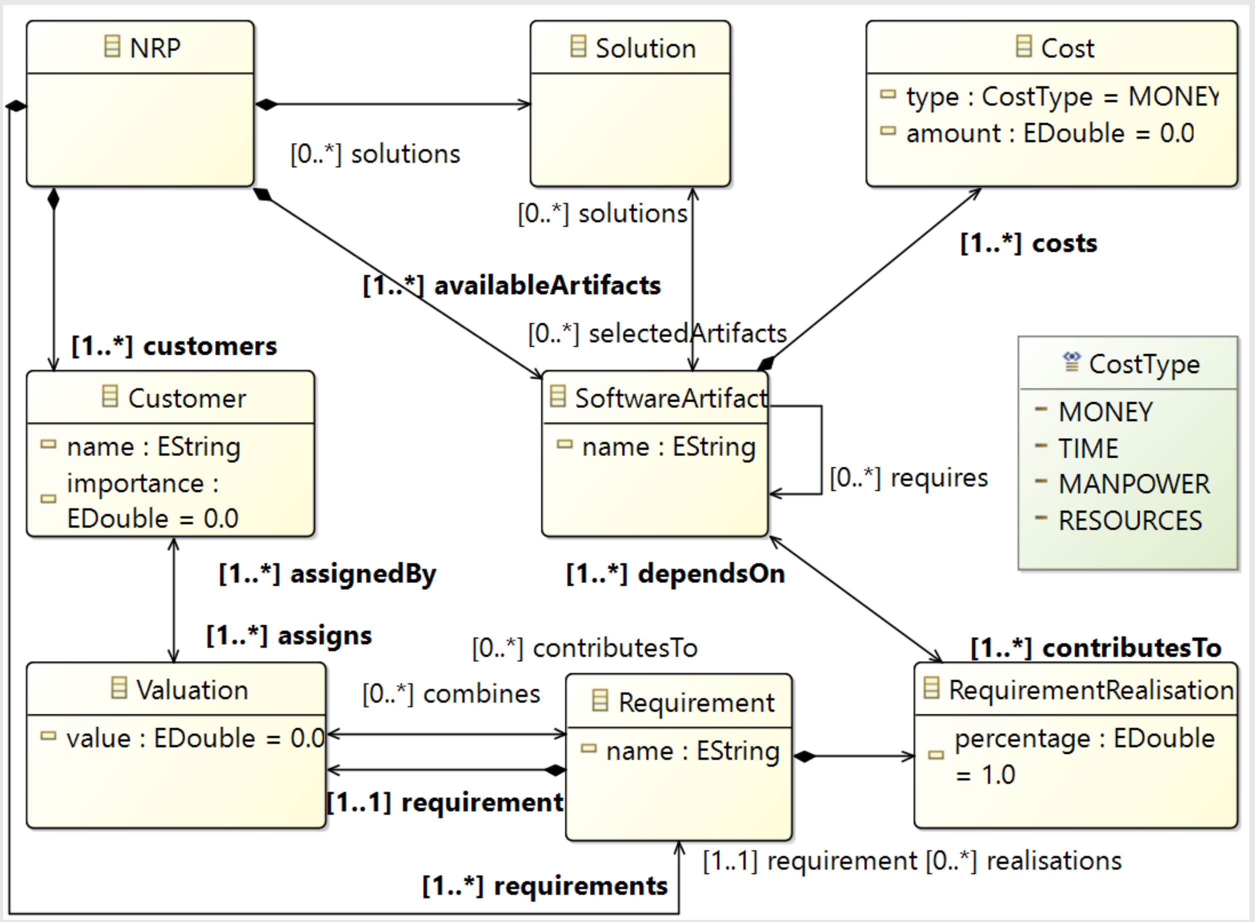
Example of solution: Release artifacts 2, 3

Cost = 1+7 = 8

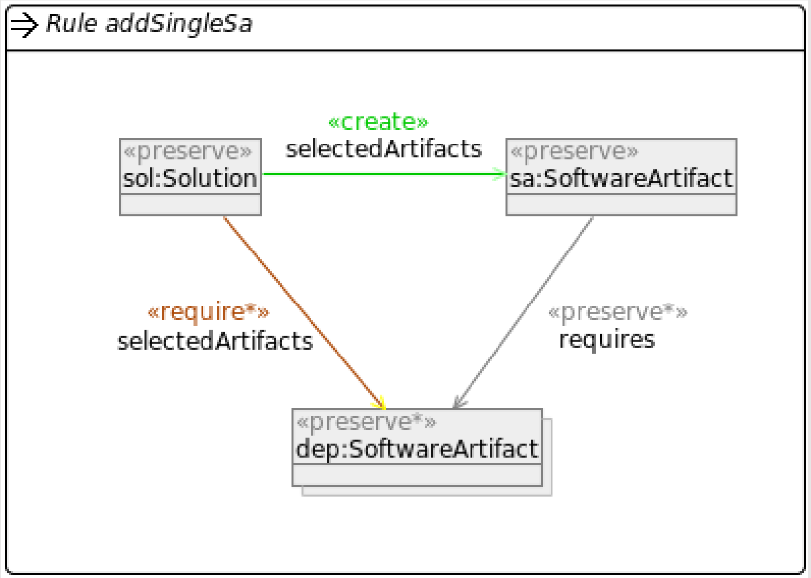
Satisfaction = 1*0% + 2*100% + 3*100% = 5

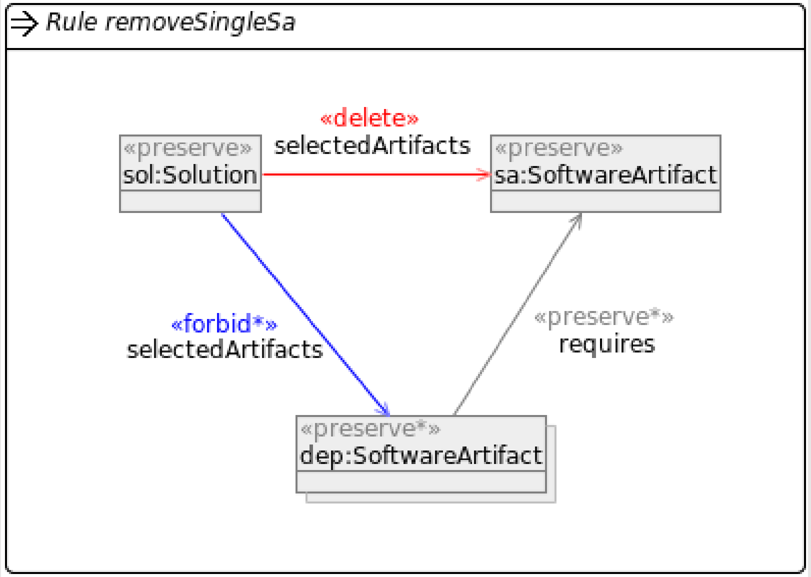
Can we do better?

The Next Release Problem



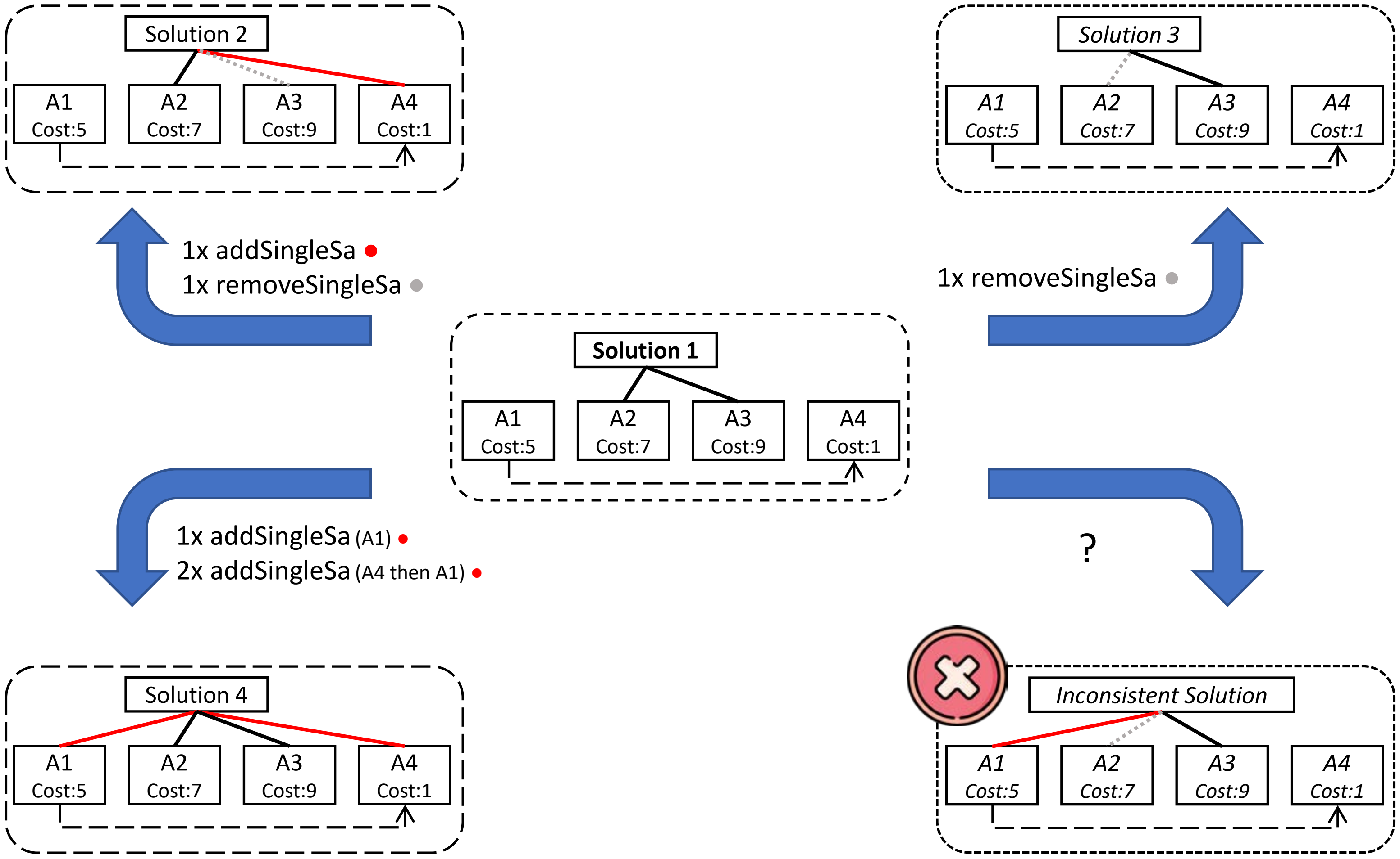
Metamodel for the next release problem (NRP)

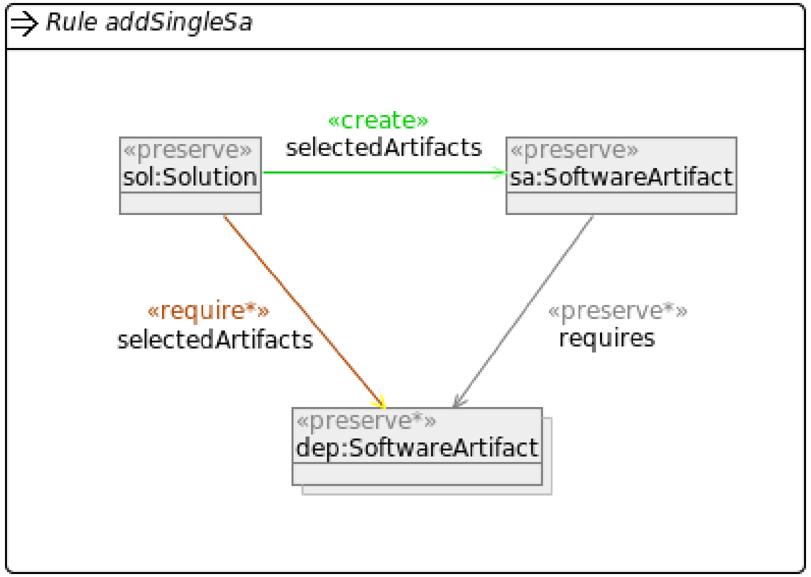


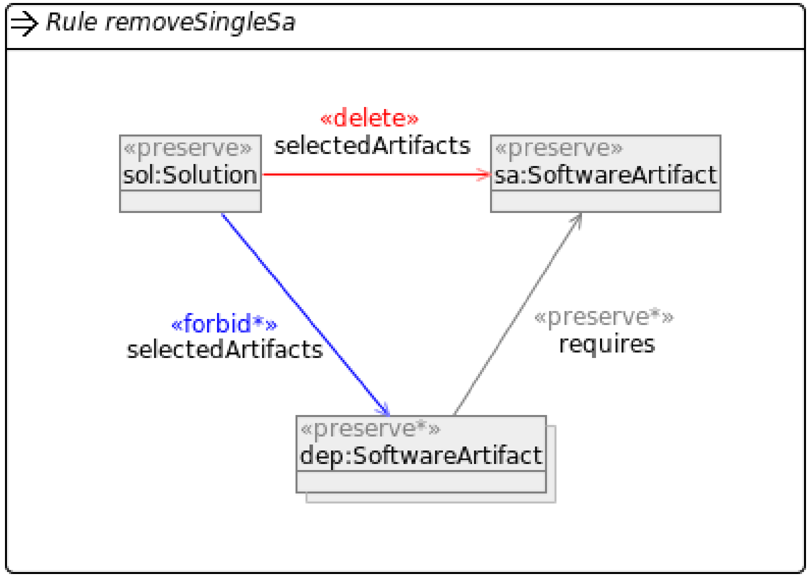


Transformation Rules

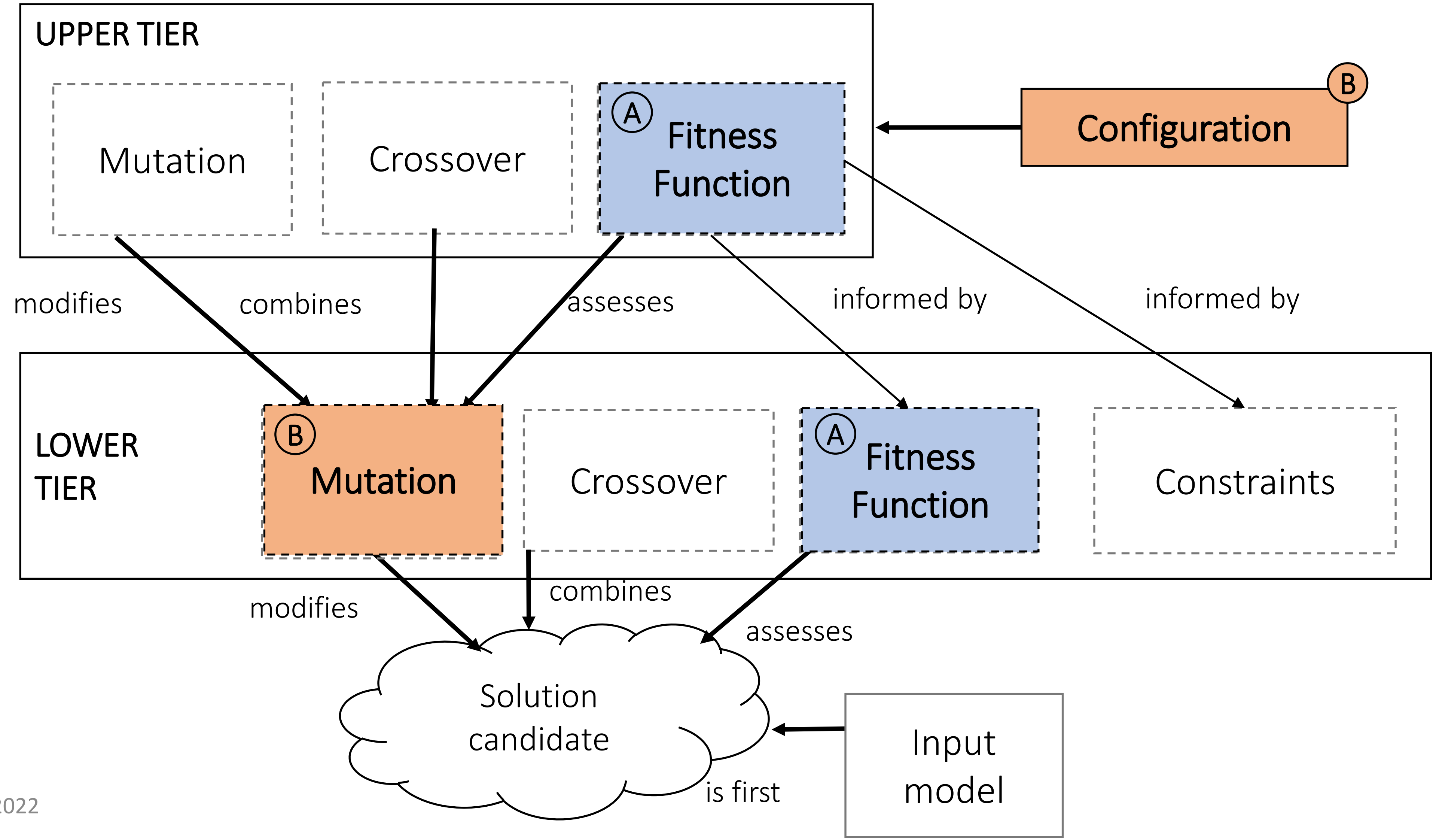
The Next Release Problem



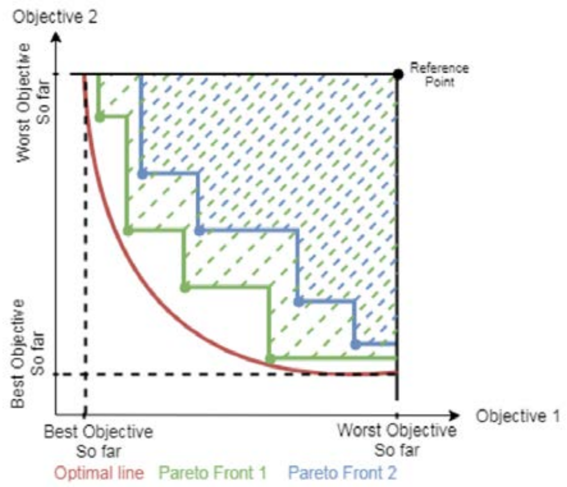




Our Contribution



(A) Support for multi-objective problems

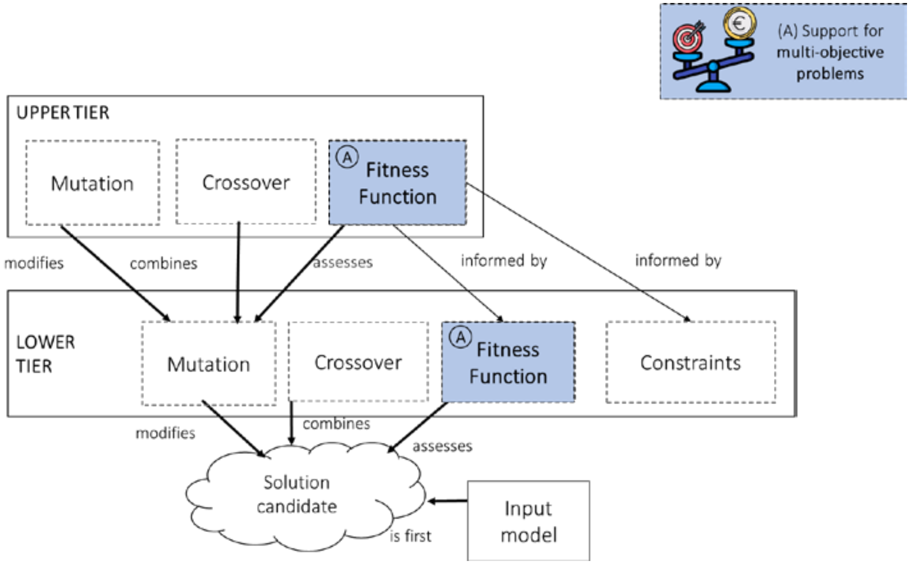


https://doi.org/10.1109/ACCESS.2019.2927418

- FitnessStudio aims at a *single objective*
  - Inapplicable to problems with multiple objectives (e.g., NRP)

$$Cost = \sum_{sa \in SA'} cost(sa)$$
$$Satisfaction = \sum_{c \in C} importance(c) \cdot satisfaction(c)$$

- Our contribution #1: Improved fitness functions
  - Lower tier: Relies on an **arbitrary number** of functions
  - Upper tier: **Hypervolume** for lower-tier solutions

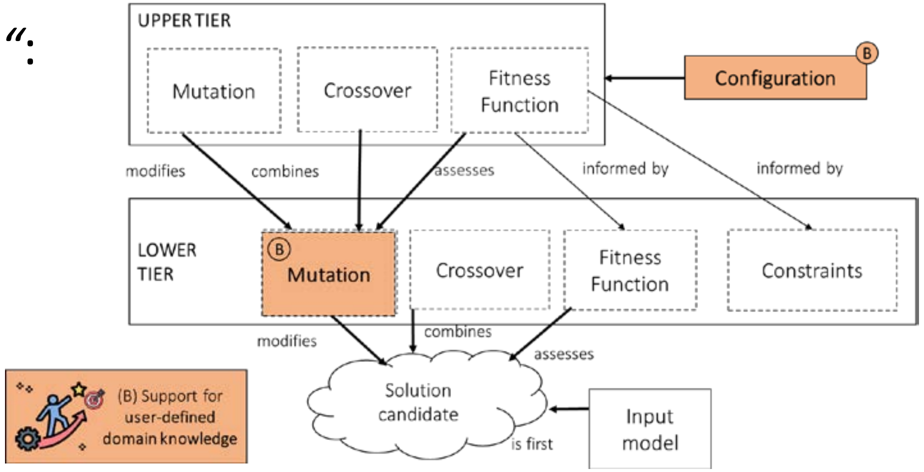


(B) Support for user-defined domain knowledge

- FitnessStudio generates mutation operators *only from scratch (a.k.a. full automation)*
- <u>Our contribution #2</u>: User-specified rule set
  - At **large scale problems**, we need *"the best of both worlds“*:
    - Useful, but not necessarily optimal mutation operators
    - Continuous improvement of mutation operators

```
if (Math.random() > 0.5)
    mutateWithFixedRules(graph);
else
    mutateWithGenRules(graph);
```

fixedXORgen: Combining generated with user specified fixed rules



- <u>Our contribution #3</u>: Configuration of upper-tier algorithm
  - Assign a weight to each higher-order mutation rule

Evaluation and Results

Evaluation

Input models	A	B	C	D	E
Customers	5	25	50	75	100
Requirements	25	50	75	100	120
Artifacts	63	203	319	425	602

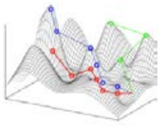
Table: *Instances of the next release problem*

- **Subject:** The Next Release Problem (NRP)
- **RQ1:** How does the mutation operator generated by our framework impact performance, compared to a manually specified operator?
  - Performance = Result quality and Execution time
  - **Focus:** multi-objective NRP

$$Cost = \sum_{sa \in SA'} cost(sa)$$
$$Satisfaction = \sum_{c \in C} importance(c) \cdot satisfaction(c)$$

- **RQ2:** To which extent does the customization with user-provided domain knowledge impact the performance?
  - Isolated user-provided domain knowledge
  - **Focus:** single-objective NRP

$$Fit(s) = \frac{sat(s)}{max\_sat} - \frac{cost(s)}{max\_cost}$$



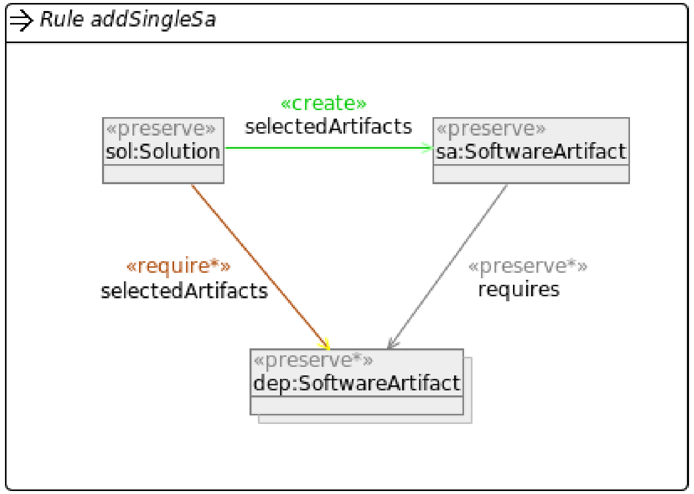
https://jmetal.github.io/jMetal/

Results for RQ1 (Performance Compared to Manually Specified Operators)

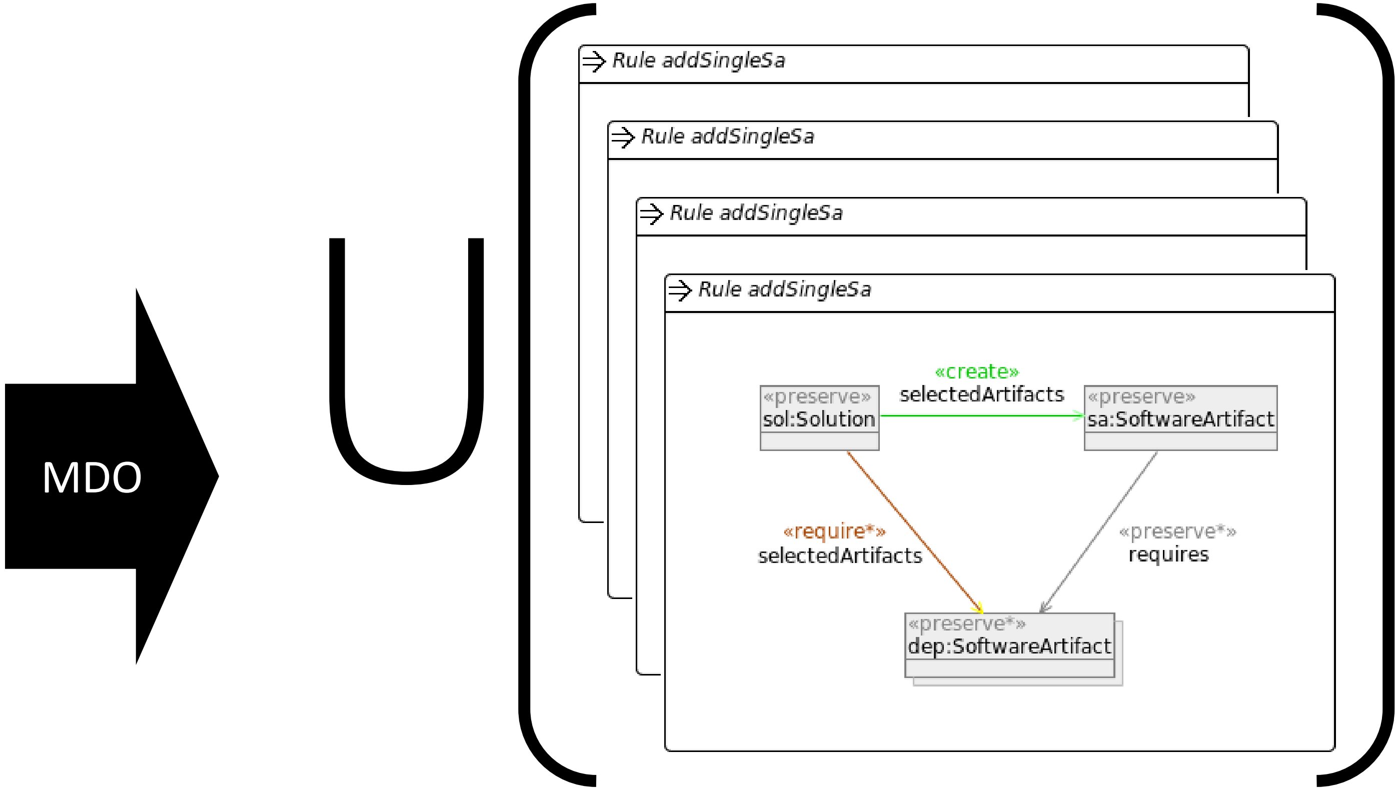
TABLE II: RQ1: $random$ and $rand + x$ results (mean, (stdev)), times denoted as mm:ss:xxx.

Results	*random*						*rand + x*					
Input model	Baseline (fixed)			Contribution (fixedXORgen)			Baseline (fixed)			Contribution (fixedXORgen)		
	HV	**Spread**	**Runtime**	**HV**	**Spread**	**Runtime**	**HV**	**Spread**	**Runtime**	**HV**	**Spread**	**Runtime**
A	**0.0911** **(0.0198)**	**0.5795** **(0.0824)**	00:08.817 (00:00.335)	0.1293 (0.0425)	0.6642 (0.059)	**00:07.284** **(00:00.486)**	0.103 (0.0134)	0.5079 (0.0559)	00:08.986 (00:00.363)	**0.0704** **(0.0116)**	**0.501** **(0.0544)**	**00:06.342** **(00:00.282)**
B	0.267 (0.0267)	0.7146 (0.0737)	00:28.057 (00:00.399)	**0.178** **(0.0255)**	**0.5283** **(0.0488)**	**00:23.433** **(00:02.271)**	0.222 (0.0193)	**0.4697** **(0.04)**	00:29.022 (00:00.339)	**0.2102** **(0.0179)**	0.4708 (0.0428)	**00:20.806** **(00:00.401)**
C	0.3512 (0.024)	0.8471 (0.0479)	00:45.160 (00:00.830)	**0.2622** **(0.0249)**	**0.7381** **(0.0584)**	**00:33.351** **(00:01.130)**	0.2629 (0.0163)	0.4205 (0.0341)	00:45.694 (00:00.529)	**0.2304** **(0.021)**	**0.4297** **(0.0387)**	**00:31.953** **(00:00.505)**
D	0.3985 (0.0209)	0.8714 (0.0494)	01:01.851 (00:01.004)	**0.3351** **(0.0257)**	**0.844** **(0.045)**	**00:46.192** **(00:01.139)**	0.2415 (0.0131)	0.4439 (0.0278)	01:03.647 (00:01.143)	**0.2283** **(0.0126)**	**0.4314** **(0.037)**	**00:43.879** **(00:00.487)**
E	0.4778 (0.0207)	0.9129 (0.0303)	01:34.769 (00:01.455)	**0.4672** **(0.0226)**	**0.9028** **(0.0324)**	**01:08.612** **(00:01.396)**	**0.3073** **(0.0151)**	**0.3698** **(0.047)**	01:34.197 (00:01.501)	0.3277 (0.0178)	0.3819 (0.0574)	**01:06.508** **(00:01.770)**

Results for RQ1 (Performance Compared to Manually Specified Operators)



Creation/Deletion of a <u>single</u> selected edge



Creation/Deletion of **<u>multiple</u>** selected edges
(aka. larger "steps" in the search space)

Results for RQ2 (Impact of Domain Knowledge on Performance)

TABLE III: RQ2: Results using *complete* and *random* initialization, times denoted as mm:ss:x.

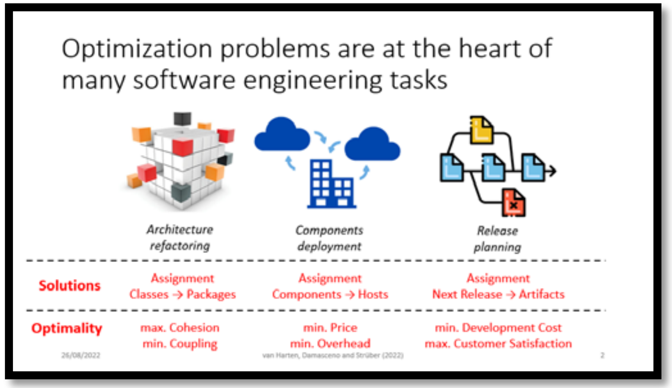
Init.	*complete* initialization						*random* initialization					
Results	Baseline (fixed)			Contribution (fixedXORgen)			Baseline (fixed)			Contribution (fixedXORgen)		
Input	NRP		Time	NRP		Time	NRP		Time	NRP		Time
model	best	median	median	best	median	median	best	median	median	best	median	median
A	**0.457**	0.446	00:10.0	**0.457**	**0.457**	**00:08.2**	0.454	0.439	00:11.4	**0.461**	**0.440**	**00:08.6**
B	0.526	0.504	**00:35.3**	**0.582**	**0.556**	00:40.7	0.589	**0.554**	00:38.4	**0.602**	0.540	**00:30.0**
C	0.379	0.357	00:47.8	**0.459**	**0.435**	**00:43.3**	0.508	0.461	00:59.2	**0.539**	**0.491**	**00:46.4**
D	0.314	0.276	**01:04.3**	**0.427**	**0.405**	01:35.8	0.434	0.403	01:20.9	**0.473**	**0.443**	**01:06.8**
E	0.218	0.207	**01:48.6**	**0.357**	**0.336**	02:25.1	0.272	0.226	02:15.0	**0.330**	**0.290**	**01:39.1**

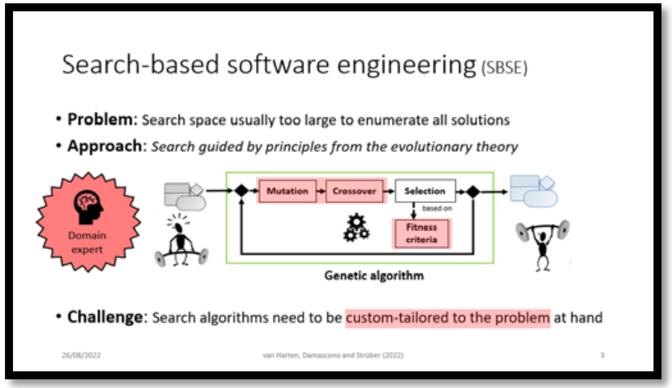
Final Remarks

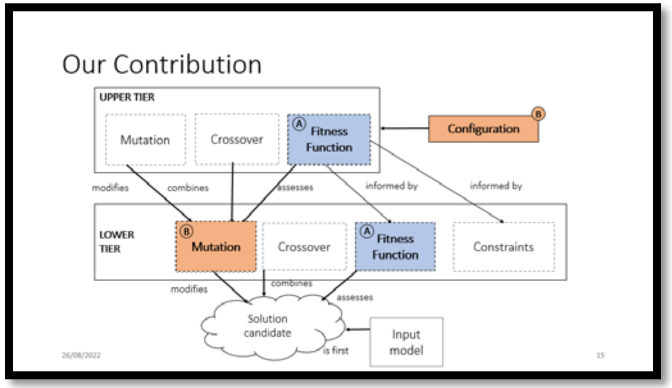
Final Remarks

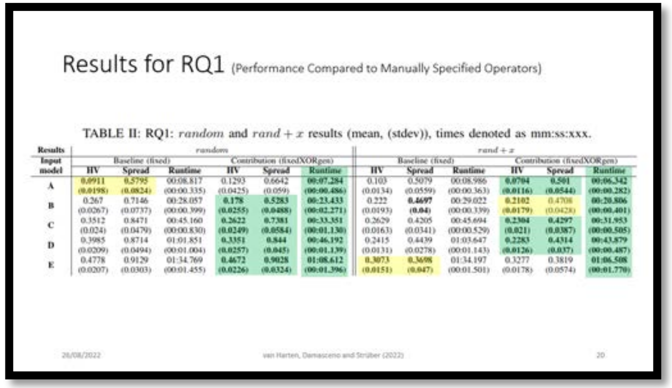
DOI: 10.5281/zenodo.6645808

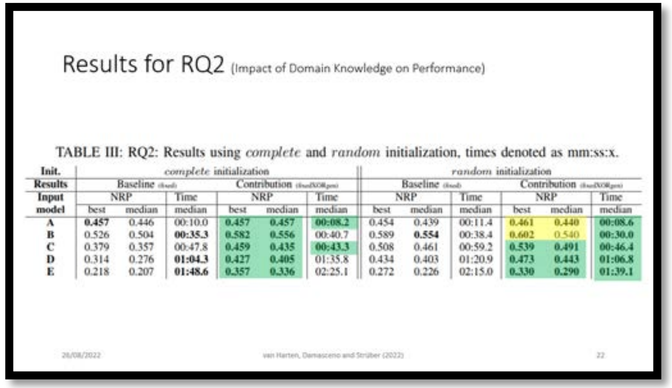
https://github.com/nielsvharten/fitnessstudio-nrp









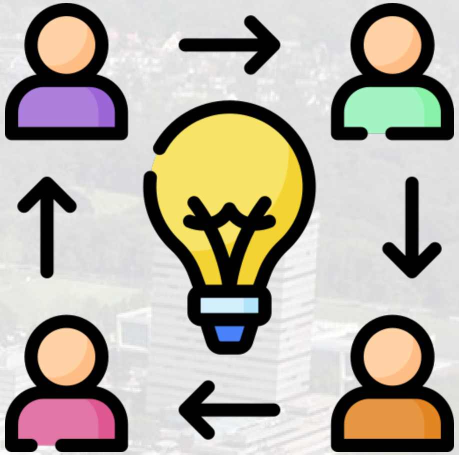


Niels van Harten
Radboud University Nijmegen
Nijmegen, The Netherlands
niels@vharten.com

CDN (**Diego**) Damasceno
Radboud University Nijmegen
Nijmegen, The Netherlands
https://damascenodiego.github.io/
d.damasceno@cs.ru.nl

Daniel Strüber
Chalmers | University of Gothenburg (SE)
Radboud University Nijmegen (NL)
https://www.danielstrueber.de/
danstru@chalmers.se

Thank you!



Questions?

@damascenodiego