Jan-Hendrik Telke
jan.hendrik.telke@rwth-aachen.de

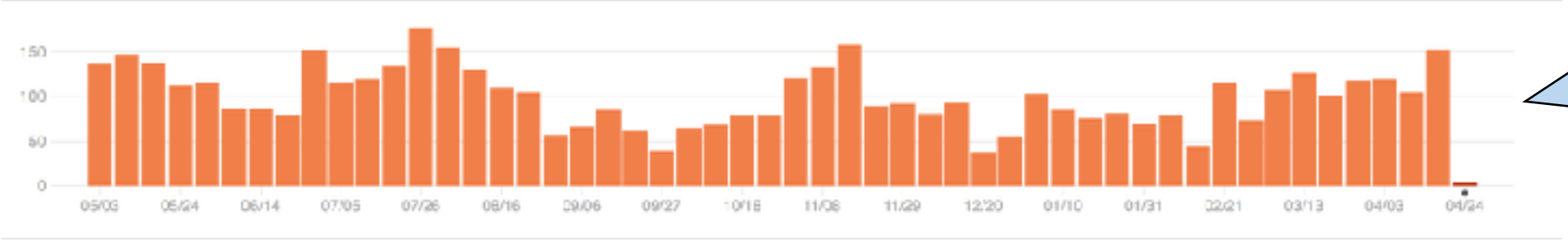
Enhancing Lazzer for Metric-based RTO Strategies

Bachelor Thesis Final Talk

SWC Software Construction | RWTH AACHEN UNIVERSITY

The Problem - Long Test Execution Times

- Apache Spark ~35 min



~ 100 c/w

https://github.com/apache/spark/graphs/commit-activity

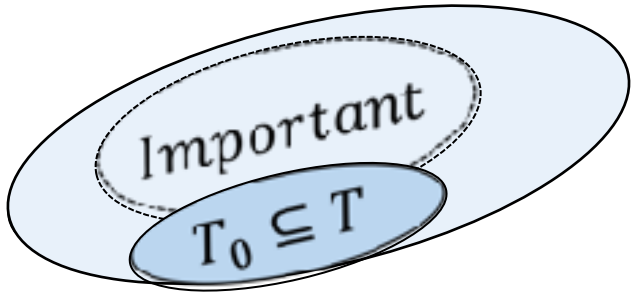
more than
~60 h per week

- Solution: Regression test optimisation

Regression Test Case Optimisation (RTO)

- Prioritisation
  - Run more important tests first
  - Does not necessarily save time

- Selection:
  - e.g. change relevant
  - Sacrifices safety



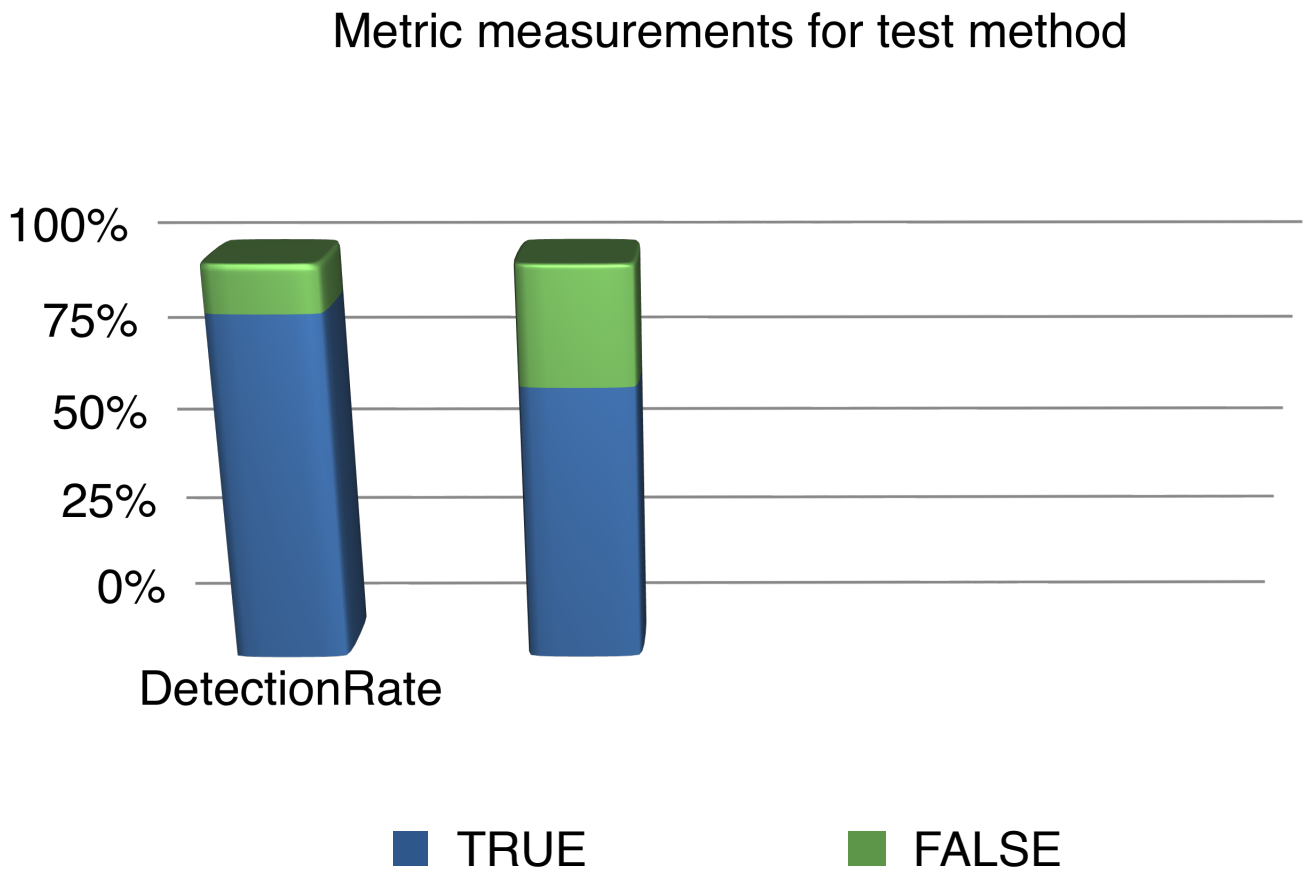
RTO - Advantages

- Faster Feedback → Faster Fix
  - Less developers involved
  - Time critical patches
- Reduces computation time
- No manual test case selection

Reduction of test executions by 40 % - 50 % with THEO

THEO - A self-adaptive approach

- Self-adaptive and metric-based algorithms
- For example THEO (Herzig et al, 2015)
  - Test selection
  - Uses metric measures:
    - Detection rate
    - False alarm rate
    - …



THEO - Cost Functions

- Calculate cost functions:

$$cost_{exec} = cost_{machine} + (P_{FP} * cost_{inspect})$$

$$cost_{skip} = P_{TP} * cost_{escaped} * Time_{delay} * \#engineers$$

if $cost_{exec} > cost_{skip}$
then skipTest()

THEO - Project Parameters

- Beside metric (self-adaptive) values

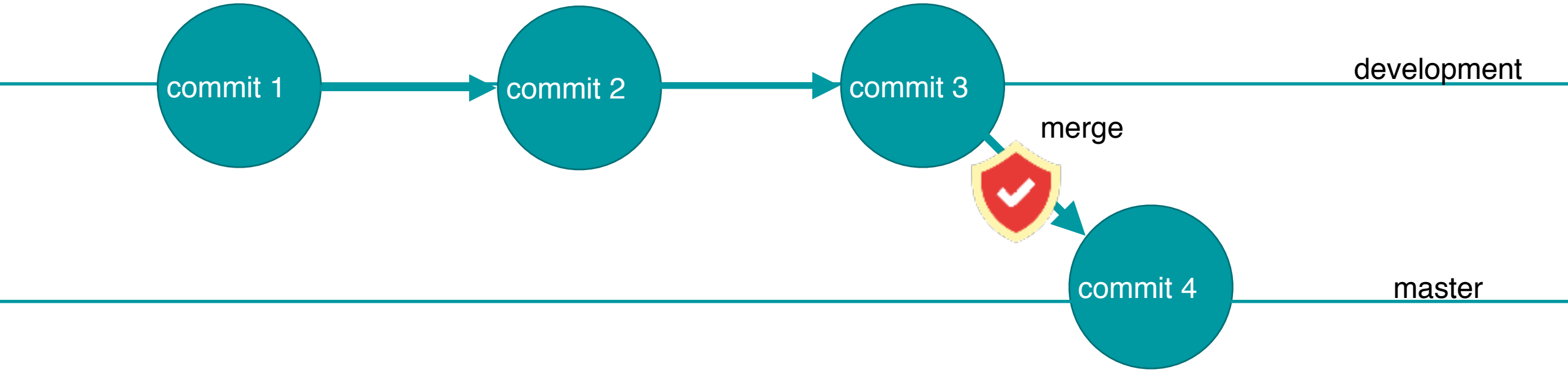
- Project specific parameters:

$$cost_{exec} = \underline{cost_{machine}} + (P_{FP} * \underline{cost_{inspect}})$$

$$cost_{skip} = P_{TP} * \underline{cost_{escaped}} * \underline{Time_{delay}} * \underline{\#engineers}$$

THEO - Project Parameters

- Quality Guards
  - Need certain branching schema
  - e.g. run all tests before merge

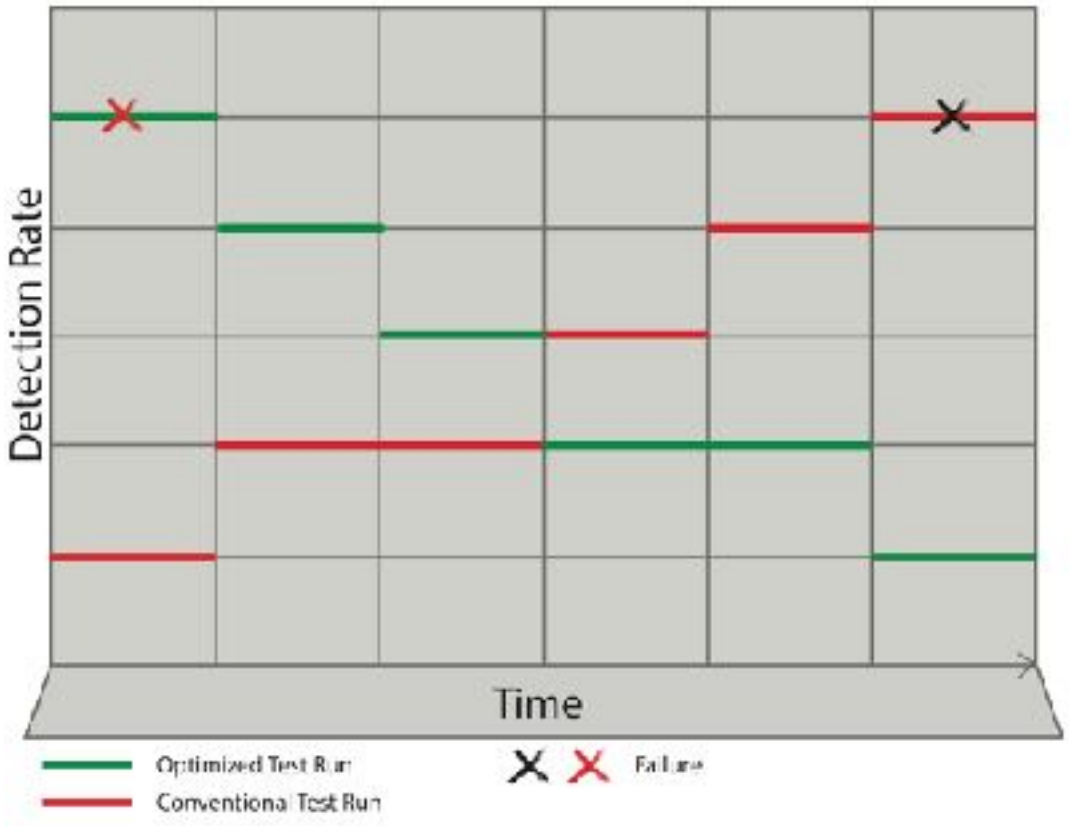


THEO - Additionally Prioritisation

- Prioritise subset
- e.g. detection rate

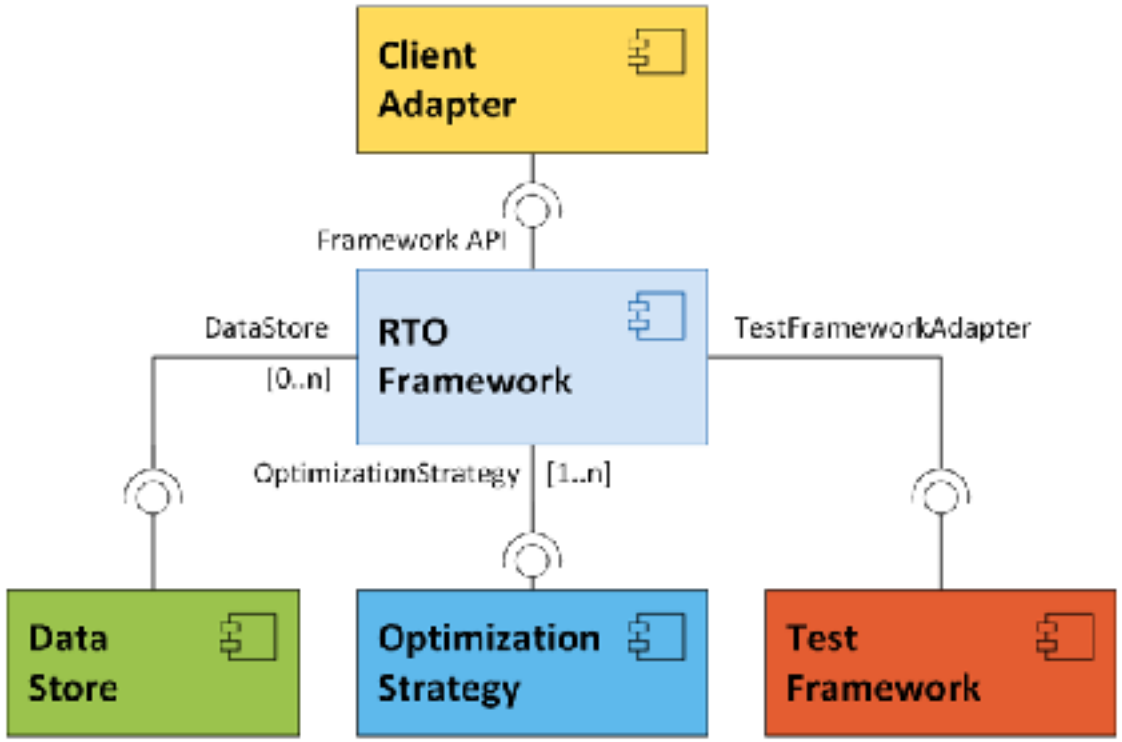
or

$$cost_{skip} - cost_{exec}$$



The Foundation - Lazzer Framework

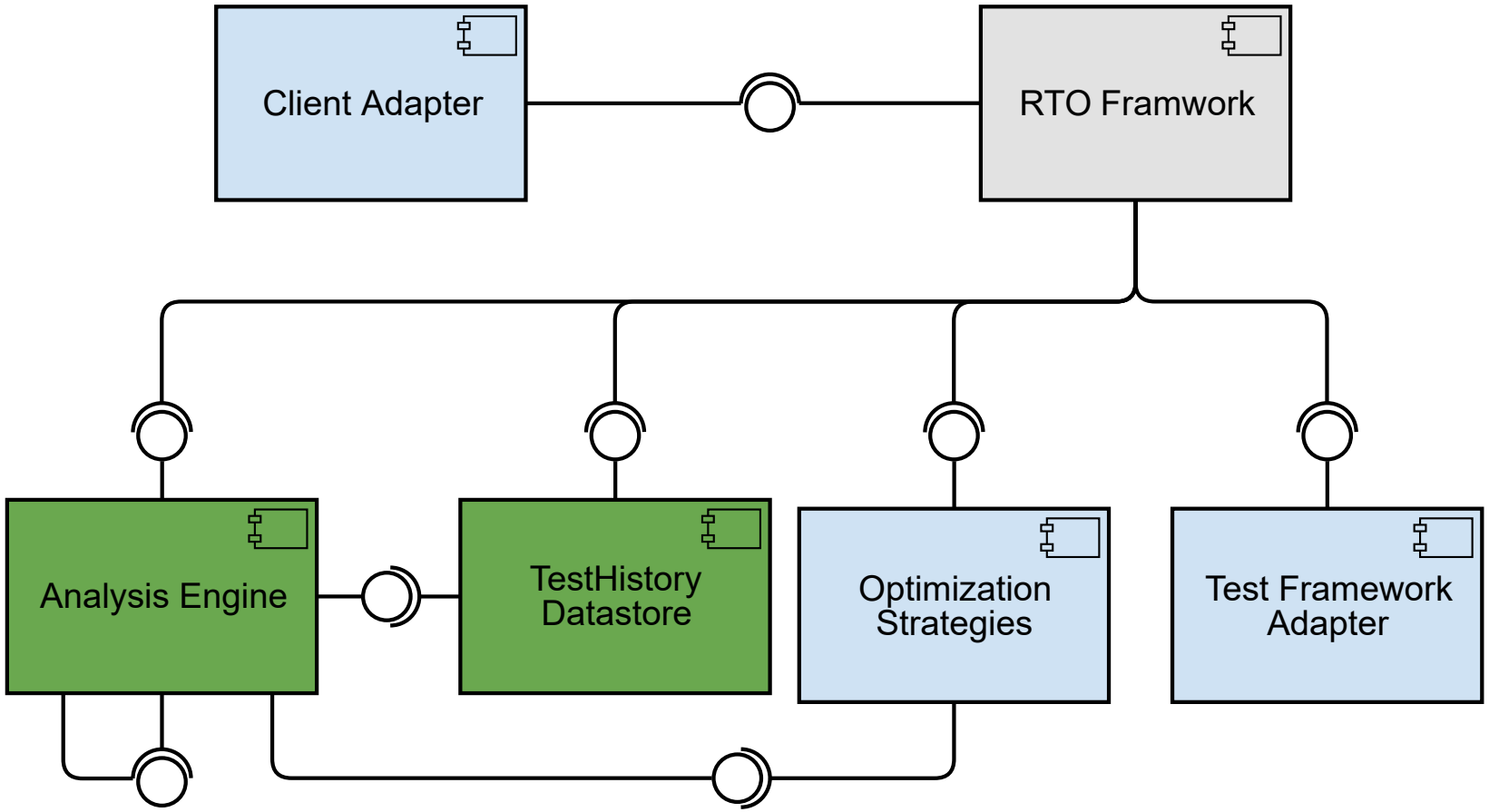
- RTO framework
- Functional prototype
- Modular structure



Requirements to Implement THEO

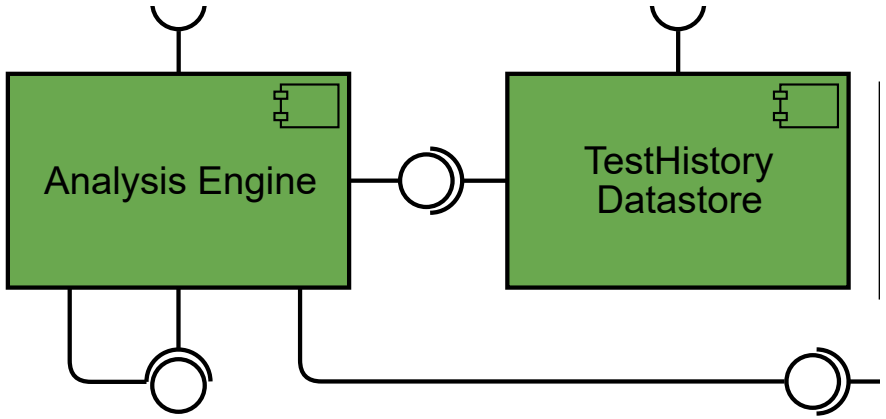
- Perform basic RTO tasks:
  - Discovery ✓
  - Selection ✓
  - Prioritisation ✓
  - Run ✓
- Handling metric data ✗
- Parameterisation ✗
- Store test history ✓✗

Concept - Refactoring and Extending Lazzer



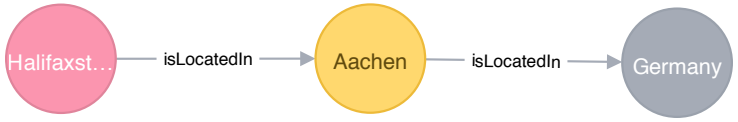
Concept - Refactoring and Extending Lazzer

- Refactoring: Split test history module



- Implement strategy parameters

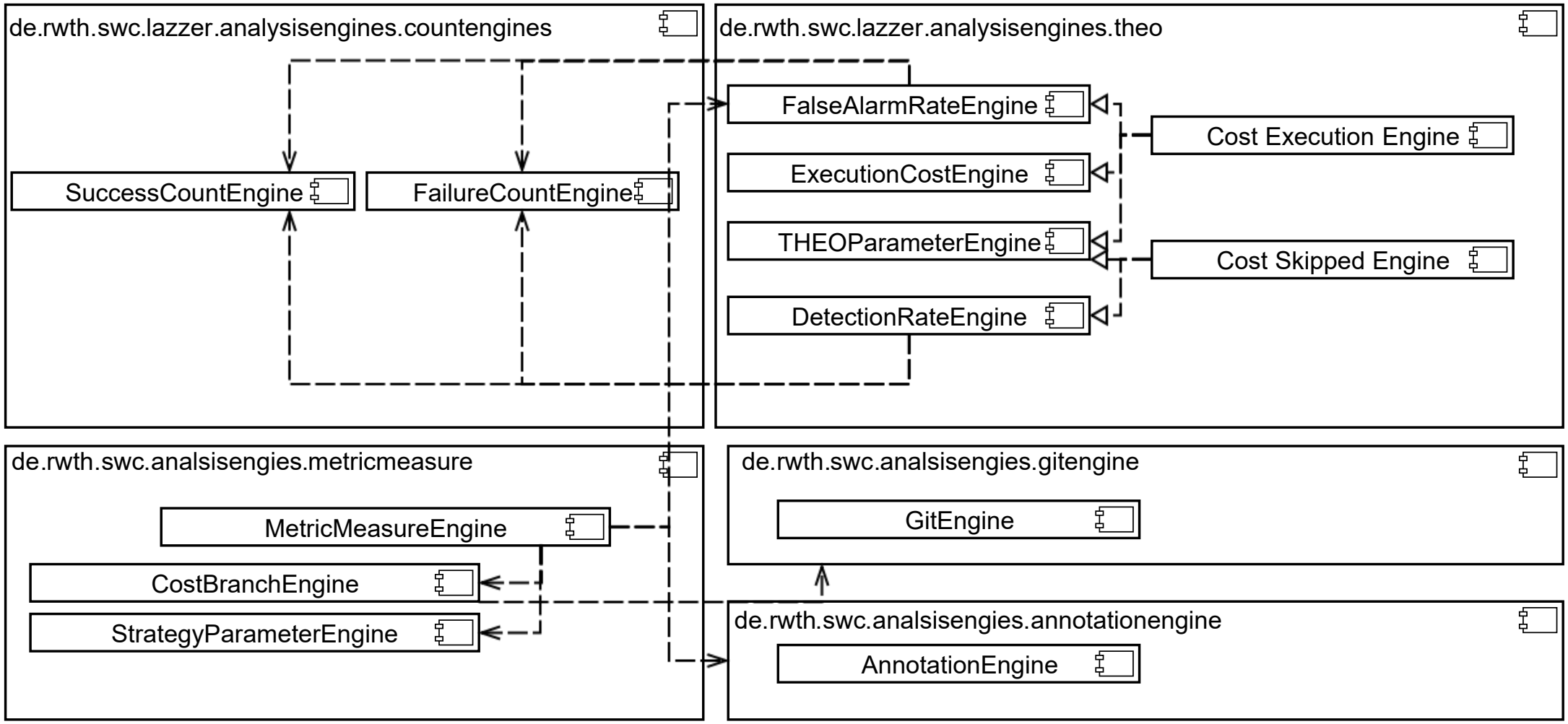
- Implement graph database



Realisation - Spring Framework

- **Spring Framework** provides a variety of helpful tools:
  - Dependency injection
  - XML reading
  - Spring Data

Realisation - Analysis Engines Overview

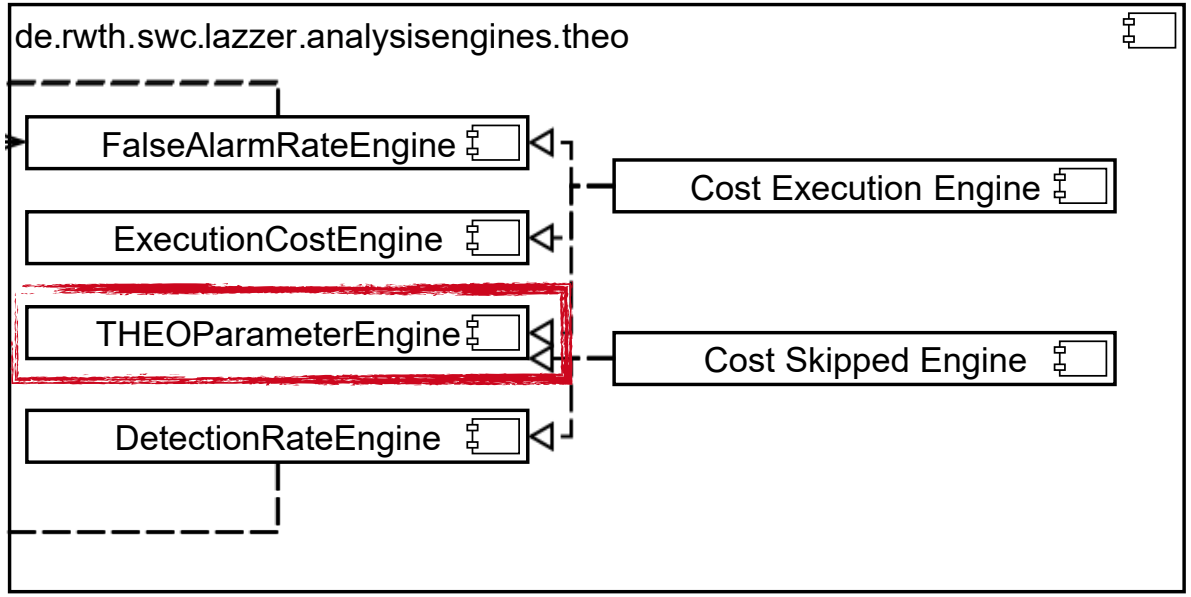


Realisation - Analysis Engines Detail

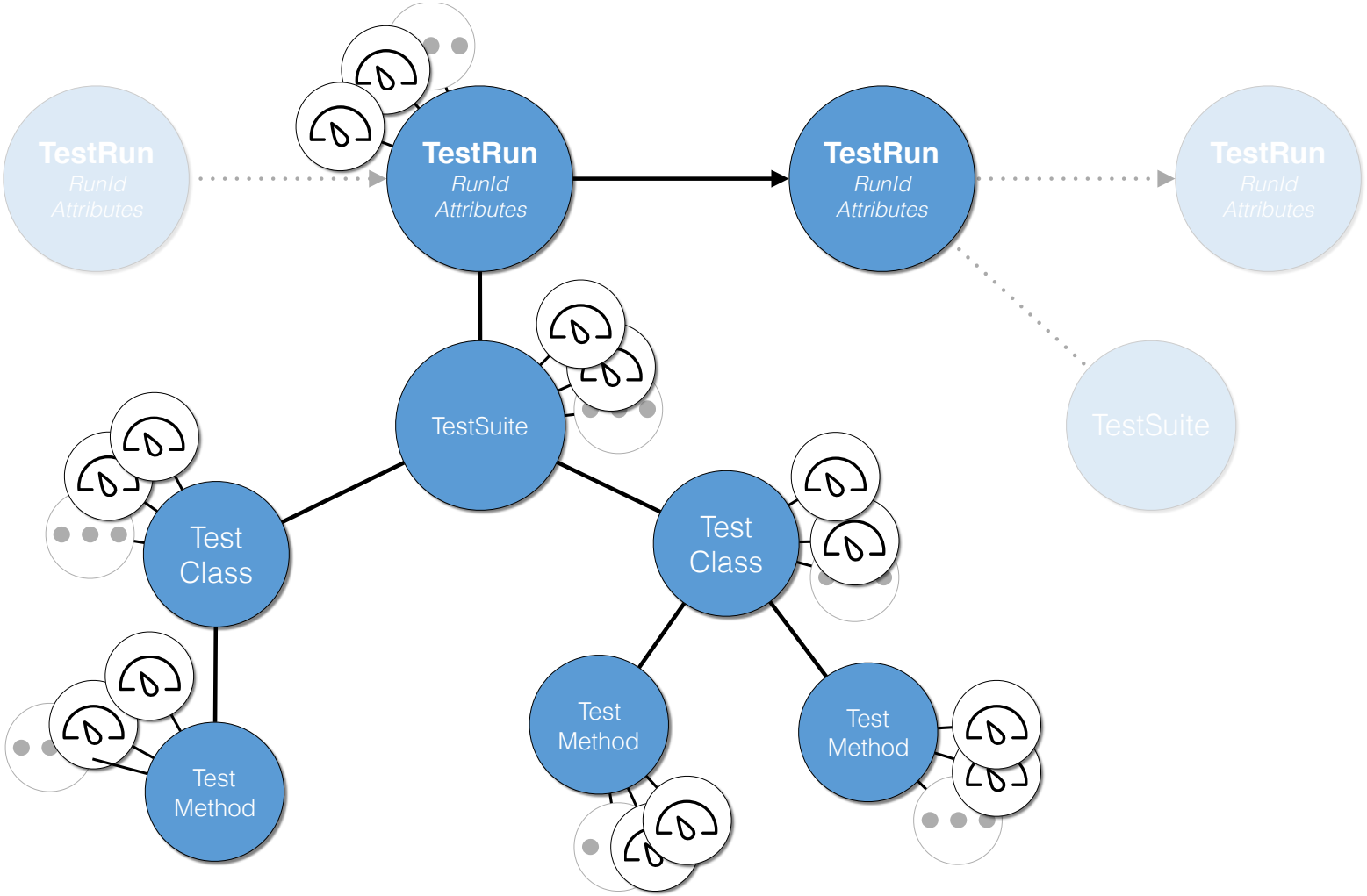
Realisation - Parameterisation

- THEO parameter engine

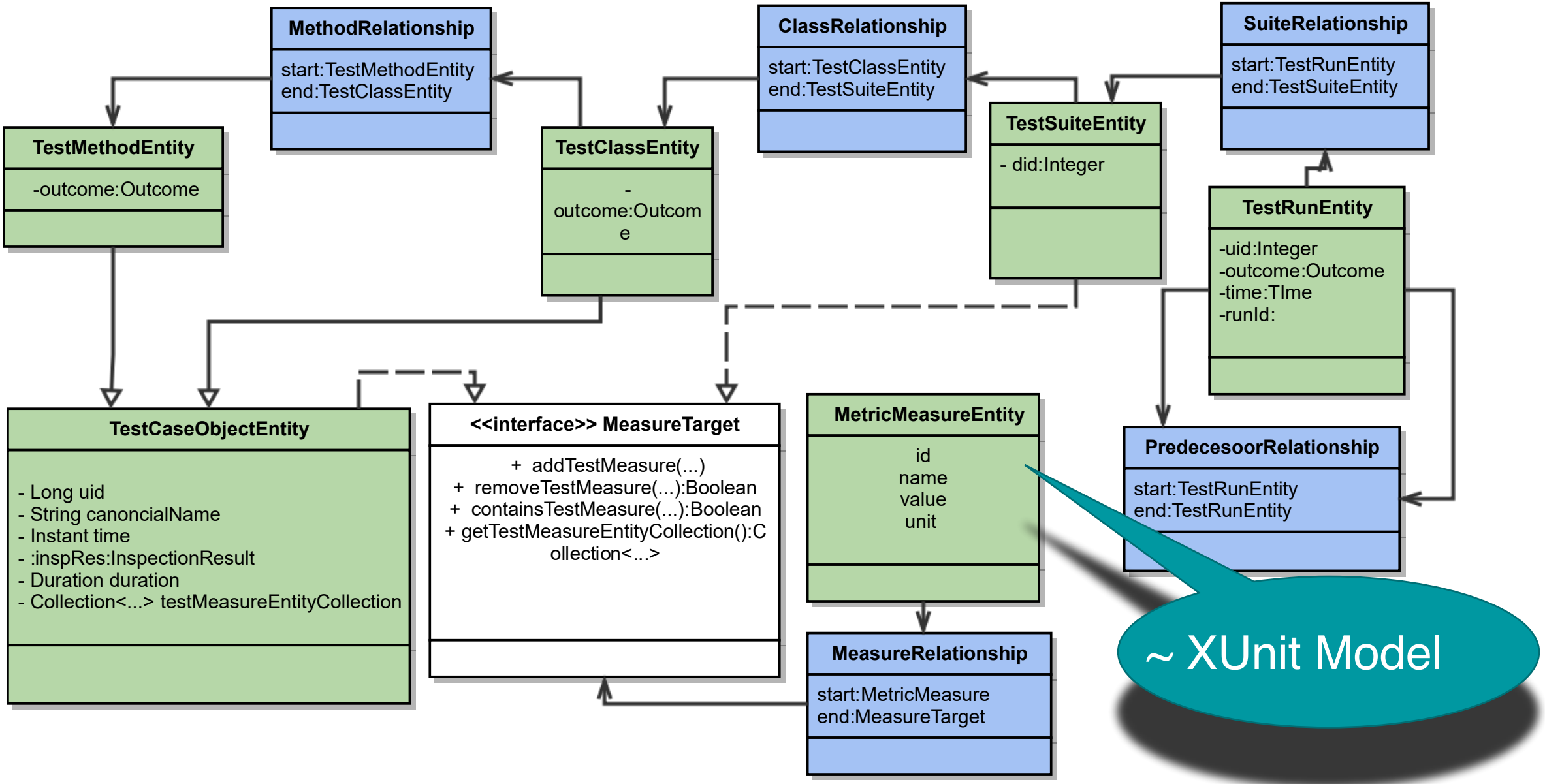
```
<theo>
    <hourly_cost>20</hourly_cost>
    <cost_escaped>5</cost_escaped>
    <cost_inspection>100</cost_inspection>
    <time_delay>2</time_delay>
    <number_of_engineers>3</number_of_engineers>
</theo>
```



Realisation - Graph Database



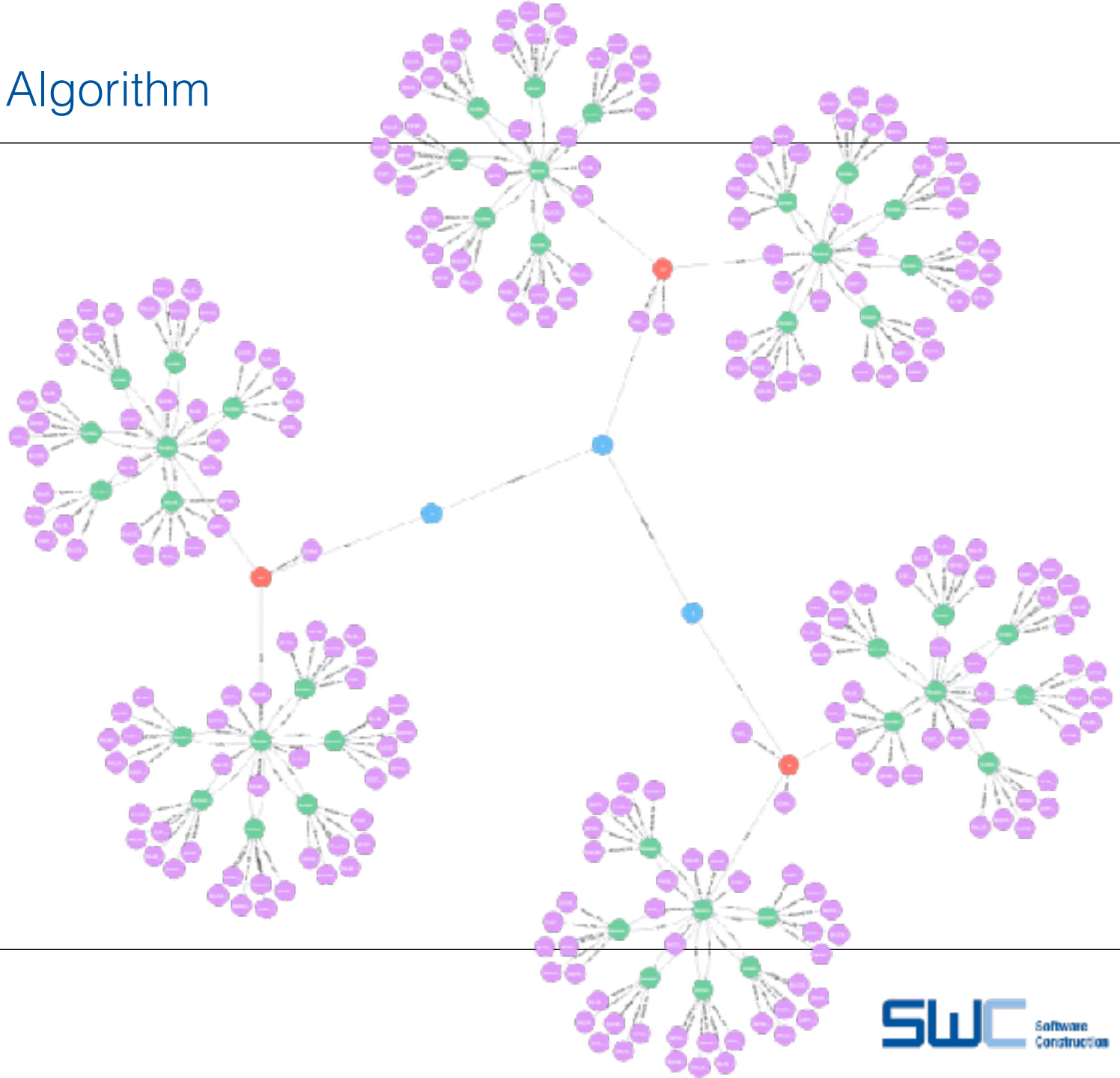
Realisation - Graph Database - Domain Model



Realisation - Graph Database - Advantages

- Advantages:
  - Resembles graph structure
  - No schema
  - Performance
  - (Graph operations, see future work)

Live Demo - THEO Algorithm



Realisation - THEO

- Access analysis engine

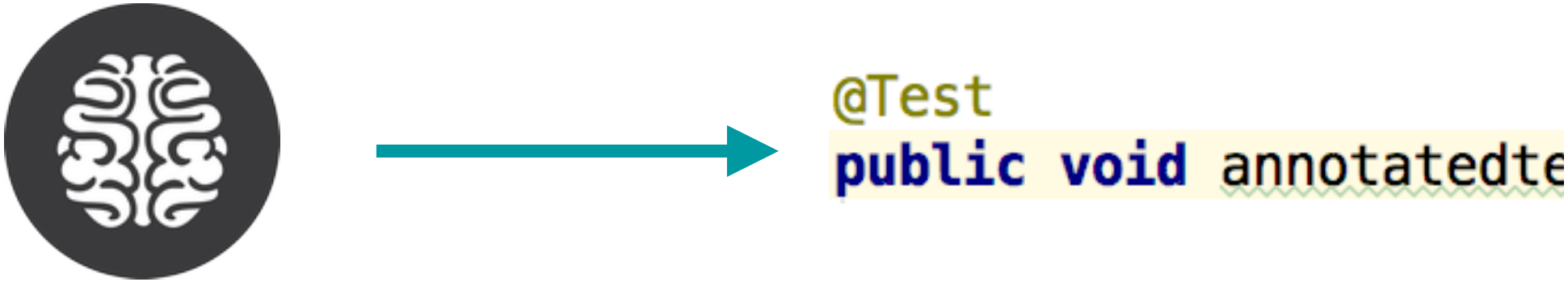
```
Optional<CostMeasure> costMeasure = Optional.empty();

return costValue.getCostExecution() < costValue.getCostSkip()
```

- Remember **Quality Guards**
  - *Simplification*: Run all tests on production branch

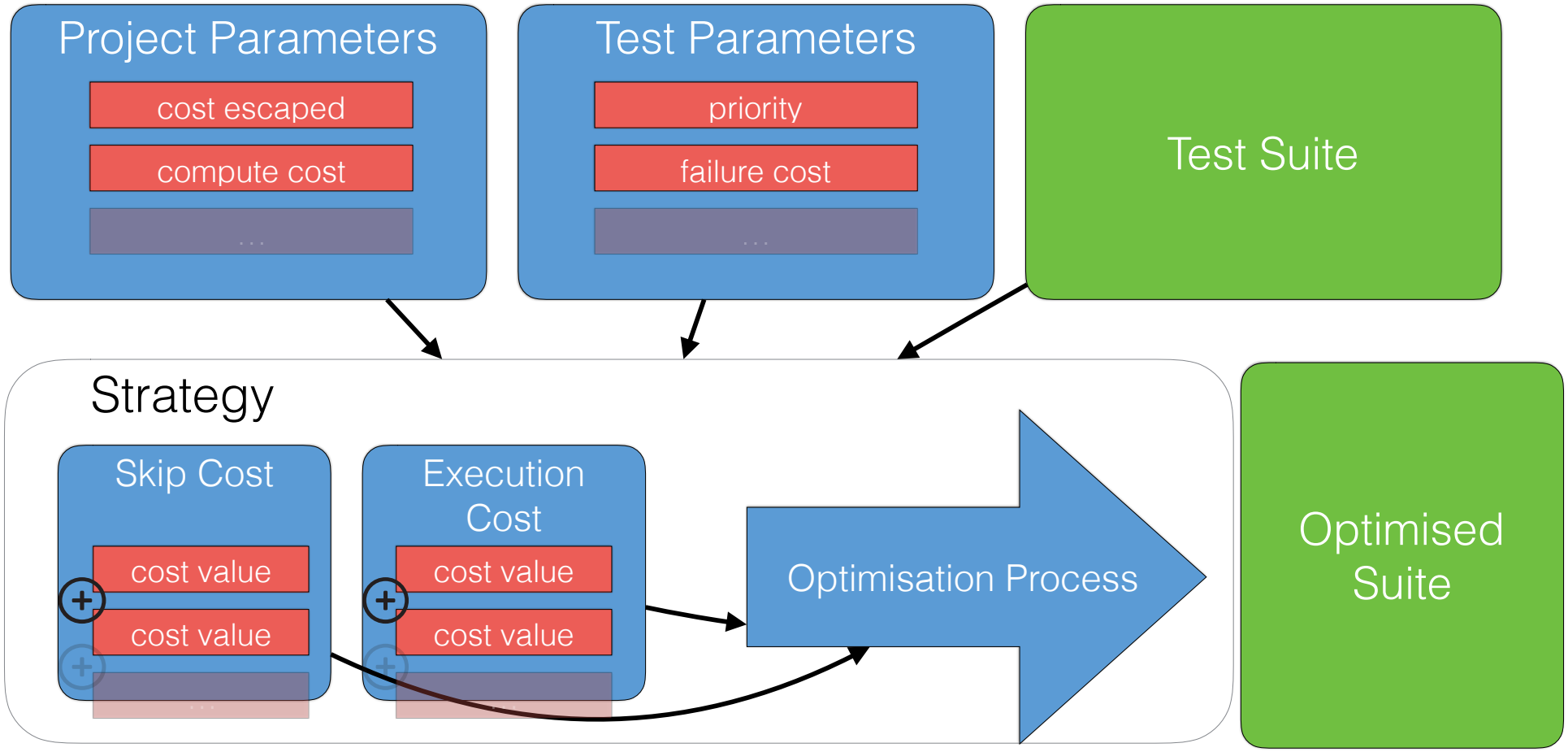
Metric Based Strategy

- Idea: Take parameterisation a step further:
  - Strategy parameterisation
  - Parametrisation of test methods

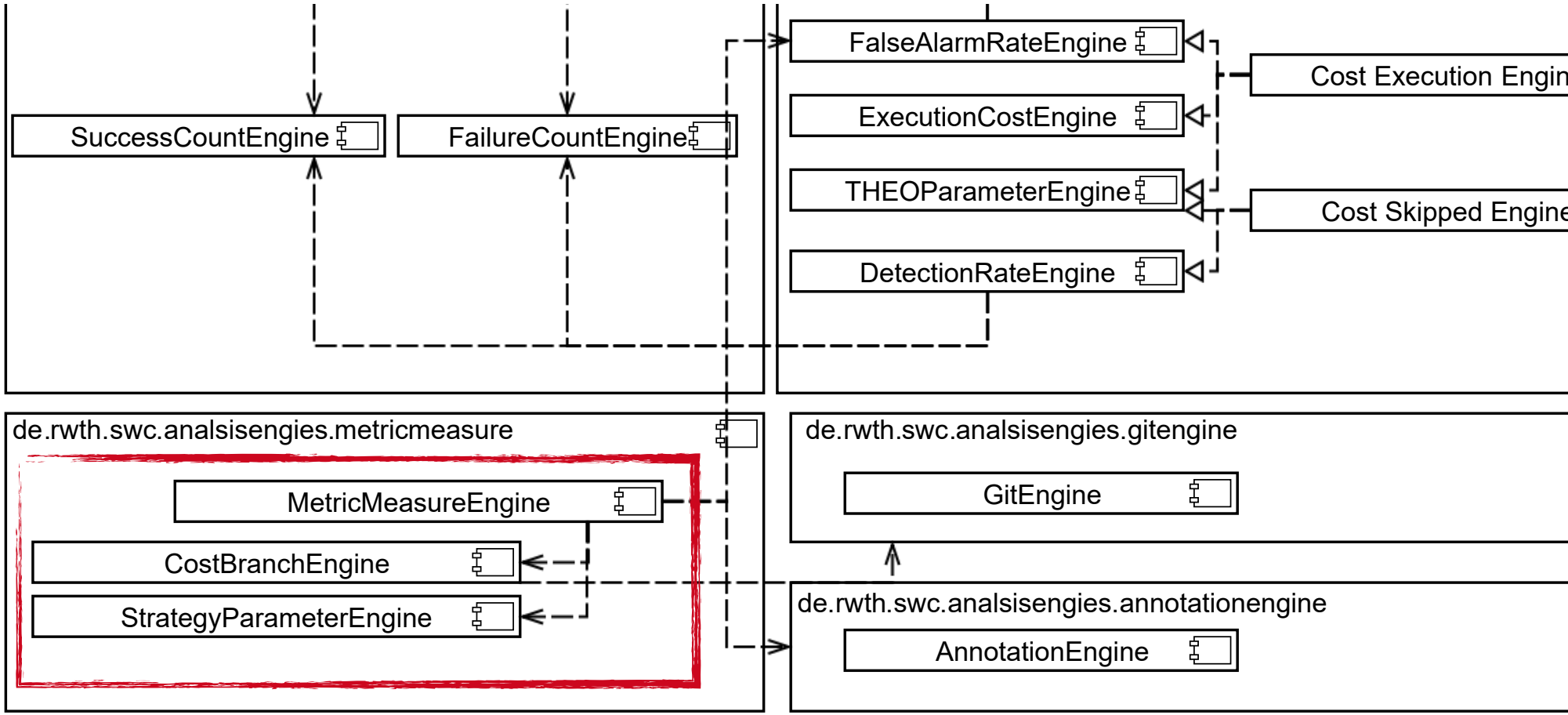
$$cost_{function} = PARAM$$



Metric Based Strategy



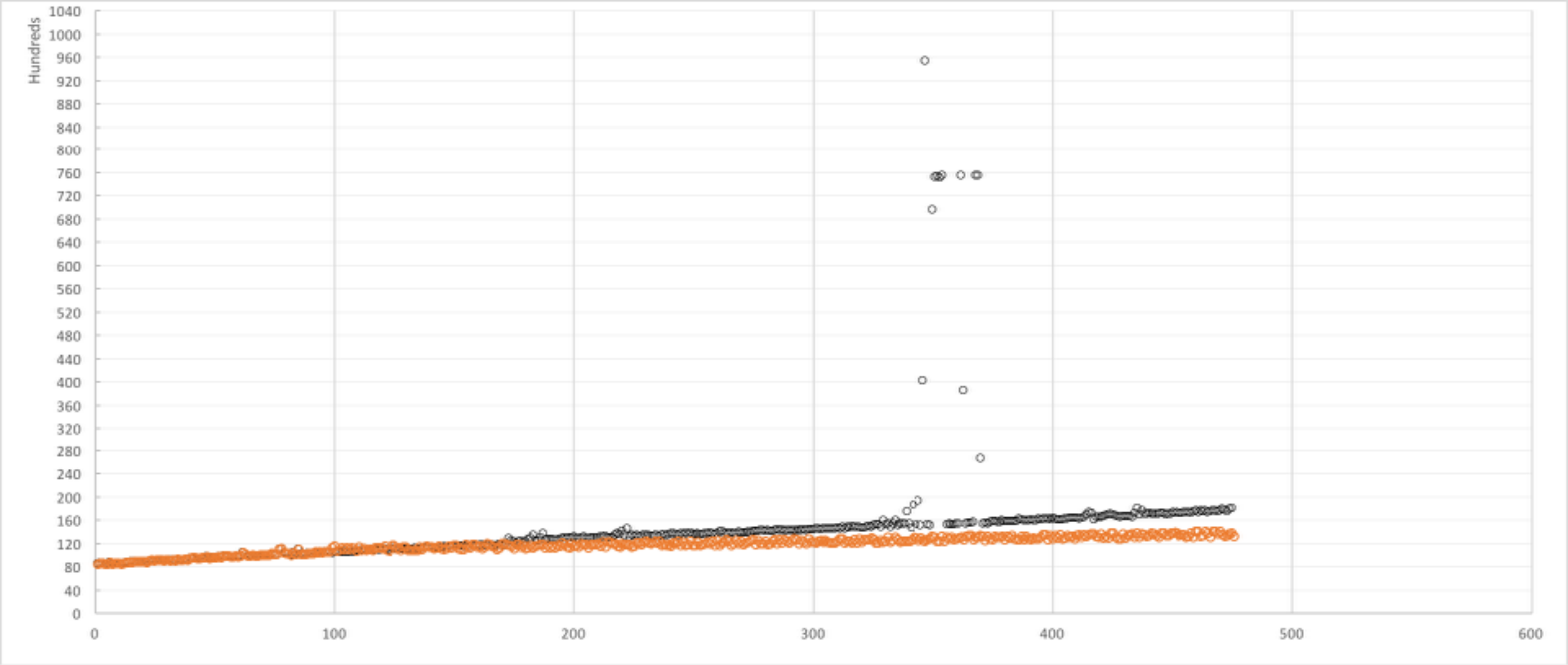
Implement metric-based strategies



Evaluation - Requirements

- Perform basic RTO tasks:
  - Discovery ✓
  - Selection ✓
  - Prioritisation ✓
  - Run ✓
- Handling metric data ✓
- Strategy parameterisation ✓
- Store test history ✓

Evaluation - Performance



Conclusion

- **Done:** Enhancing Lazzer for Metric-based RTO Strategies

- Future Work:
  - Real world evaluation and comparison
  - Other metric based strategies
  - Dependency graph strategies