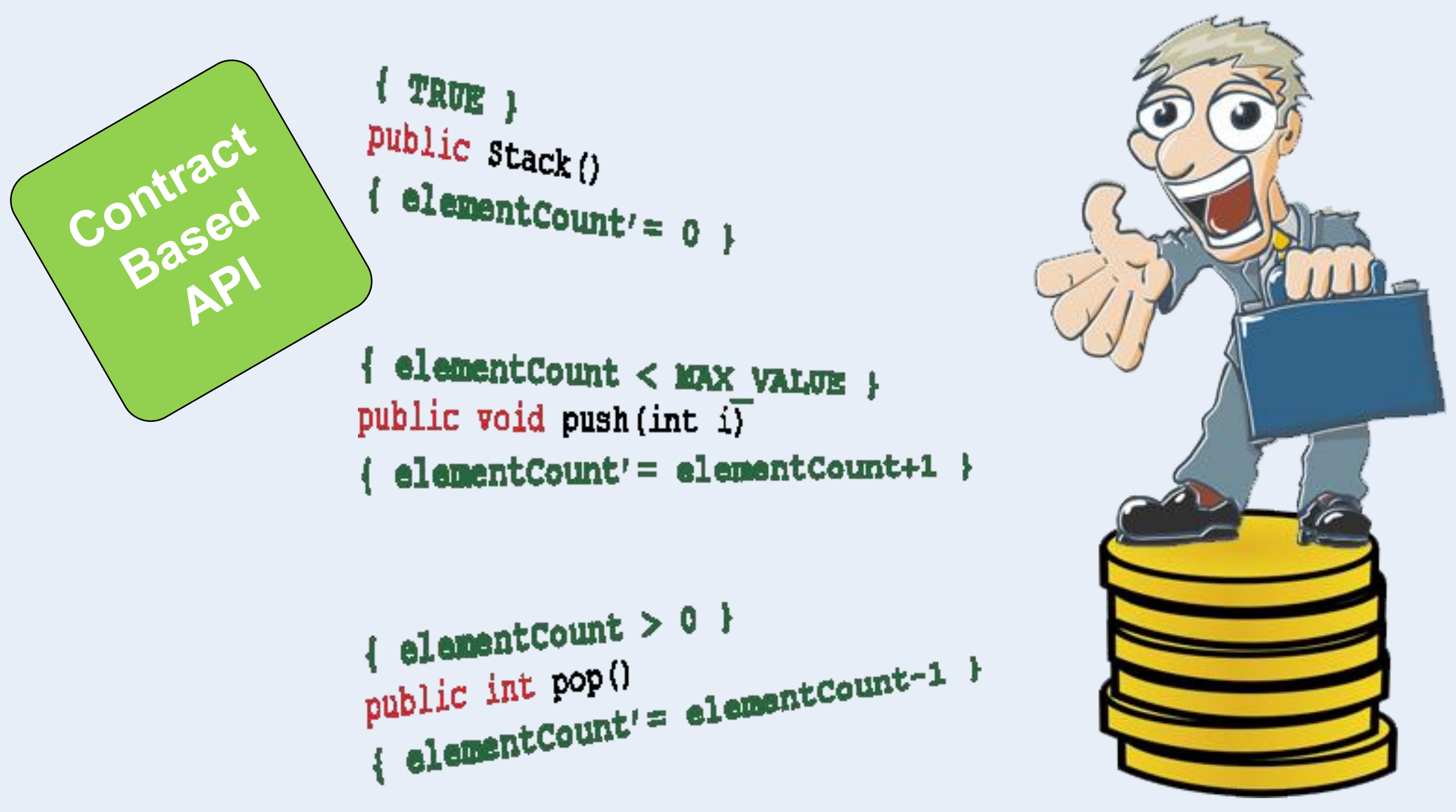
Behaviour Abstraction Adequacy Criteria for Protocol Conformance Testing

Hernán Czemerinski – hczemeri@dc.uba.ar
Departamento de Comptación, FCEyN, Universidad de Buenos Aires

Why Testing Matters?

Goal: To Find Bugs

Do Implementations Meet Specs?



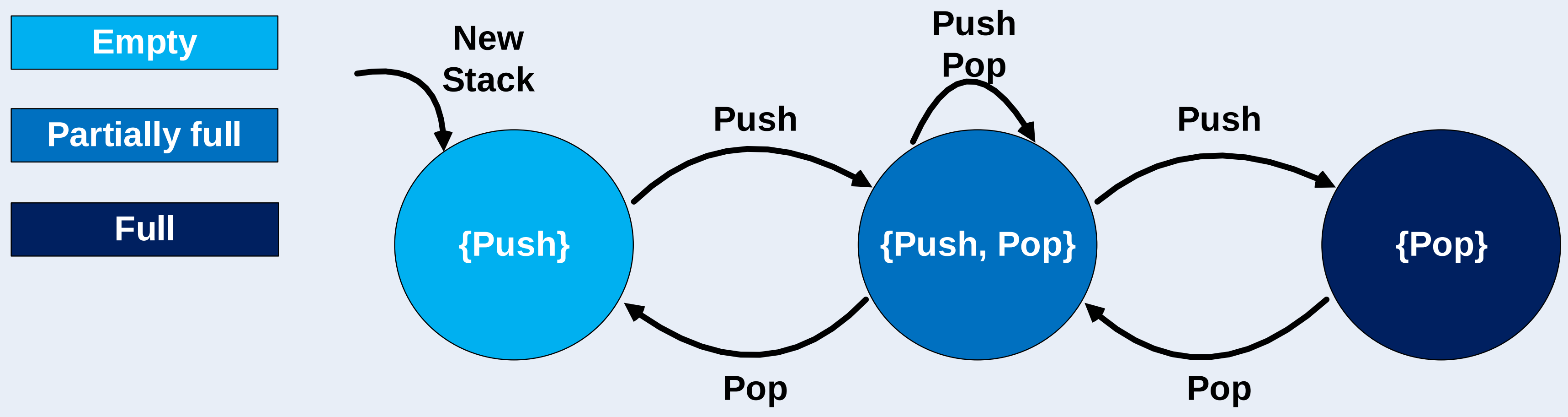
```
{ TRUE }
public Stack()
{ elementCount'= 0 }

{ elementCount < MAX_VALUE }
public void push(int i)
{ elementCount'= elementCount+1 }

{ elementCount > 0 }
public int pop()
{ elementCount'= elementCount-1 }
```

Enabledness Models

API specifications are often given in terms of contracts:

- ✓ they say a lot about what each function does, but...
- ✗ don't say much about how it should be used as a whole; i.e. its protocol

For having such understanding we use enabledness models

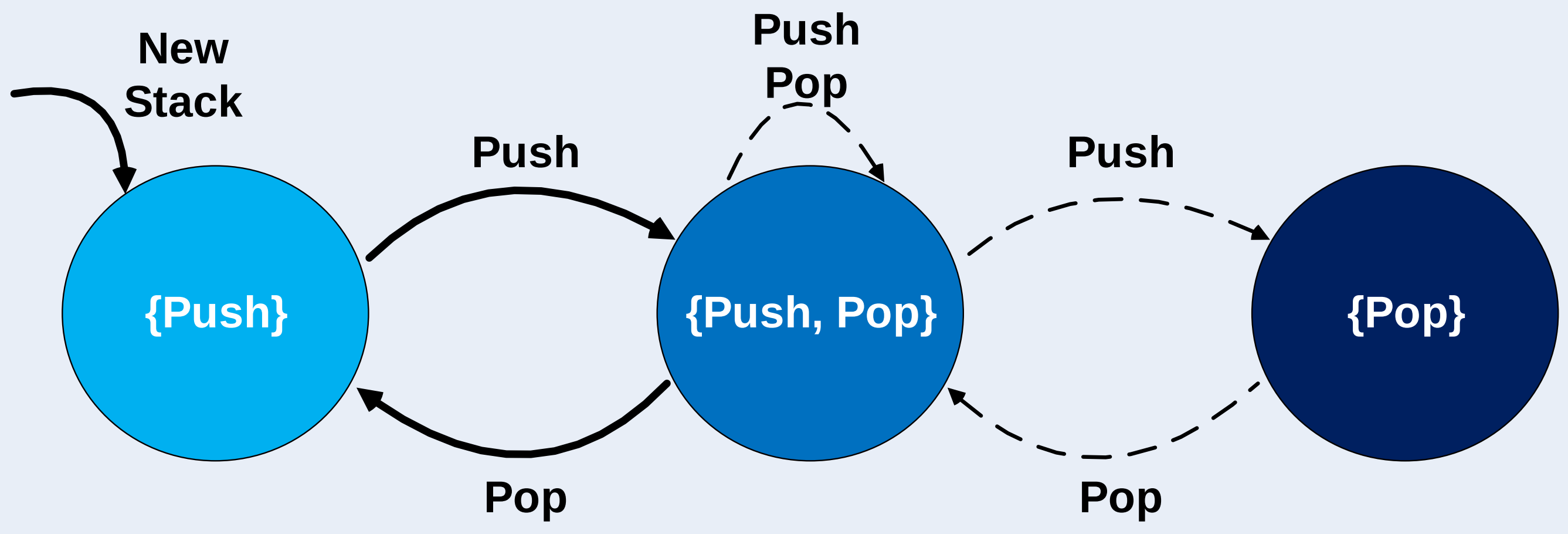


- each state of the model represents a particular set of enabled/disabled actions
- a transition from state *A* to state *B* labeled with *c* means that when the available actions of the API are those that *A* represents, after the execution of *c* the new set of enabled actions could eventually be those that *B* represents
- we build the enabledness model by using the Contractor tool, which takes either a contract-based API specification or its source code as input and produces the model as output

Coverage Criterion

We say that a test suite is EPA-k adequate if it covers at least k% of the transitions of the API Enabledness model

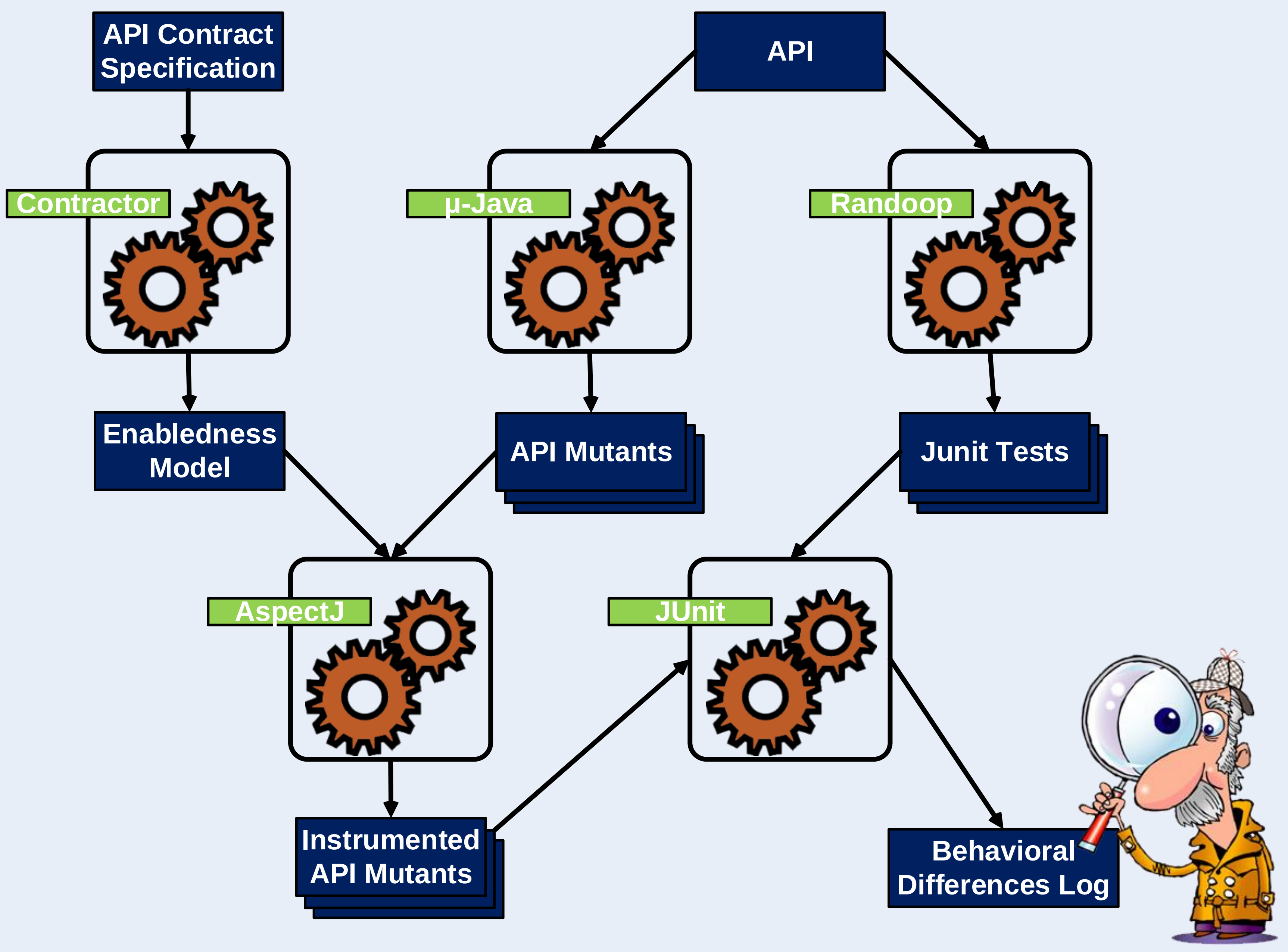
```
public void test() {
    BoundedStack bs = new BoundedStack();
    bs.push(´a´);
    bs.pop();
    assertEquals(0,size(bs));
}
```



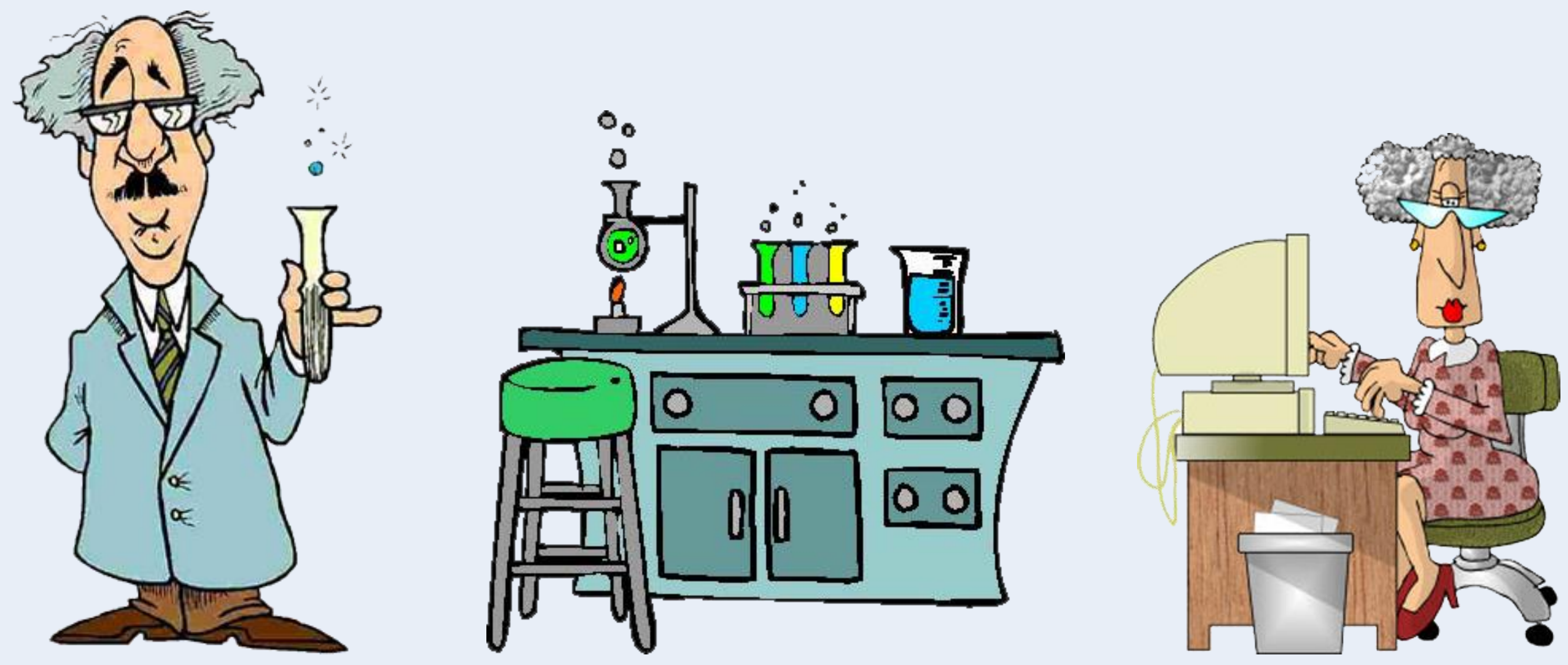
Experimental Setup

In order to obtain experimental data for contrasting our hypothesis we choose Java APIs and:

- generate enabledness models from a contract-based specification
- generate randomly JUnit Tests using Randoop tool for the API
- generate mutants of the API by executing the Mu-Java framework
- instrument API mutants for logging how they exercises the model
- execute unit tests on mutated versions of the API and detect behavioral differences found with respect to the oracle (API original version)



Findings



We experimented on fiver industrially relevant Java classes with rich protocols and observed the following:

- the criterion is a good predictor of mutant detection
- for fixed-size test suites, those with highest behavioral adequacy are statistically better in terms of fault finding
- the criterion is good predictor of structural coverage (statement and branch coverage)
- the domain partition implicitly dervied from the criterion is likely to produce subdomains that are dense in failures (i.e., with high failure rate)
- for each fault there is almost always on transition highly effective in detecting it, while nearly all the rest have poor effectivness
- transitions are much more effective than actions for exposing faults
- prioritzing tests according to the criterion produces test orders with high APFD values (even better that does using structural coverage criteria)