Support software certification by testing actual code against security requirements

Thomas Arts

QuviQ

Observations

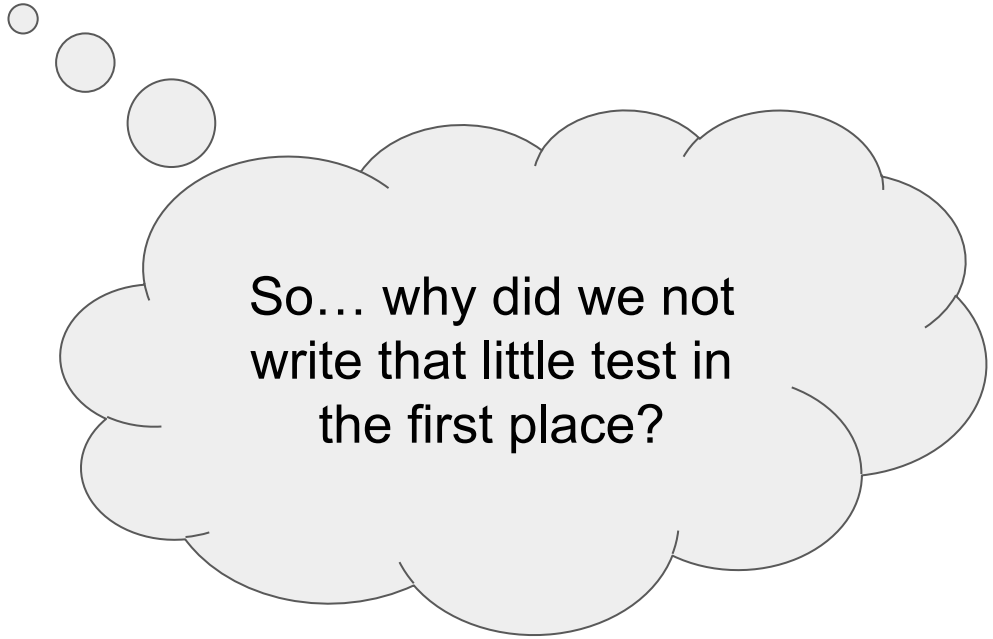
Application vulnerabilities one important cause of data breaches / attacks

Errors in your code that are not found or found, but not fixed

Observations

Application vulnerabilities one important cause of data breaches / attacks

When vulnerability is detected, a small test can show presence of this vulnerability



Why is testing hard?

O(n) test cases

3 - 4 tests per feature

features	tests	additional tests when adding 1 feature
20	80	4

Why is testing hard?

$O(n^2)$ test cases

pairs of features

features	tests	additional tests when adding 1 feature
20	80 + 190	4 + 20

$$\sum_{n=1}^{n=19} n$$

Why is testing hard?

$O(n^3)$ test cases

triples of features

features	tests	additional tests when adding 1 feature
20	80 + 190 + 1140	4 + 20 + 190

Finding tricky faults in software is difficult

Don't write tests!

Generate them

from a specification

A specification describes how the software should behave

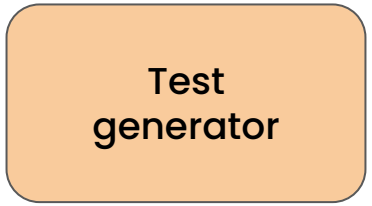
It is a linear description…
adding a feature makes it only a bit longer

Use this to automatically generate and execute tests from

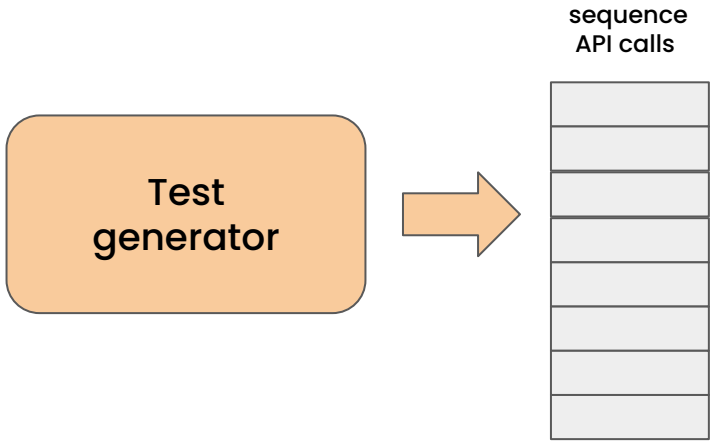
Koen Claessen and John Hughes. 2000. QuickCheck: a lightweight tool for random testing of Haskell programs. **SIGPLAN Not. 35**, *9 (Sept. 2000), 268–279.* https://doi.org/10.1145/357766.351266

Thomas Arts, John Hughes, Joakim Johansson, and Ulf Wiger. 2006. Testing telecoms software with quviq QuickCheck. In Proceedings of the 2006 ACM SIGPLAN workshop on Erlang (ERLANG '06). Association for Computing Machinery, New York, NY, USA, 2–10. https://doi.org/10.1145/1159789.1159792

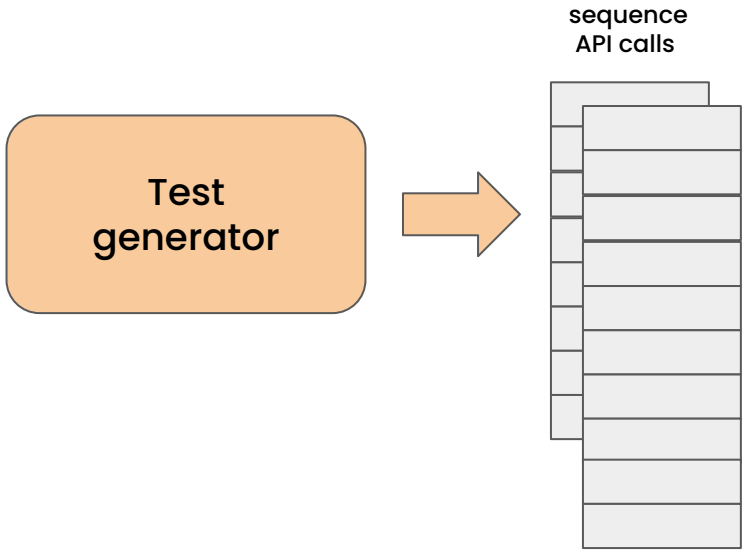
Generation of test sequences



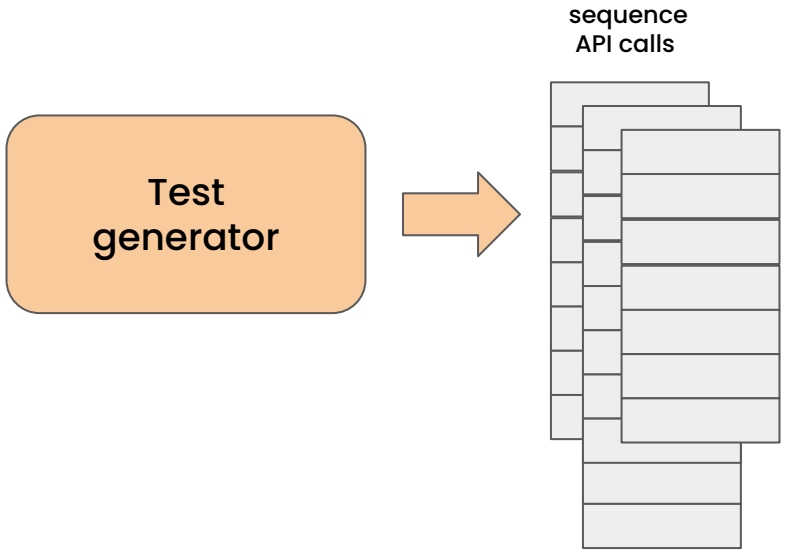
Generation of test sequences



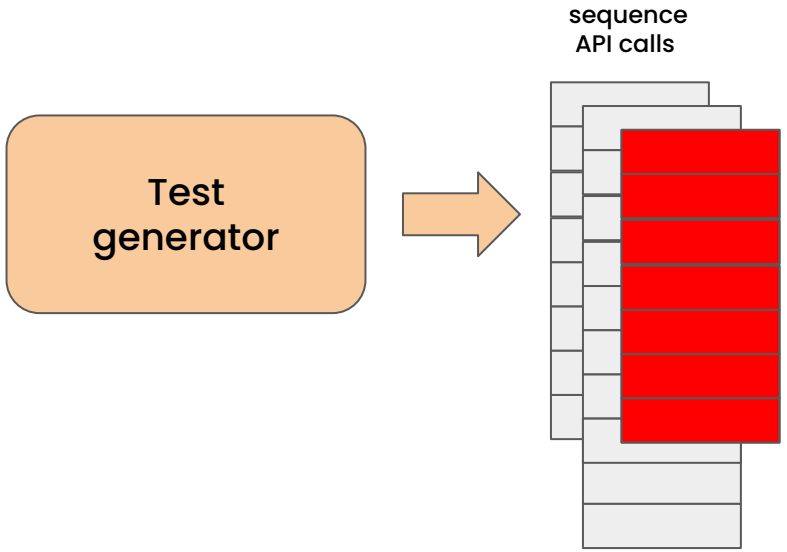
Generation of test sequences



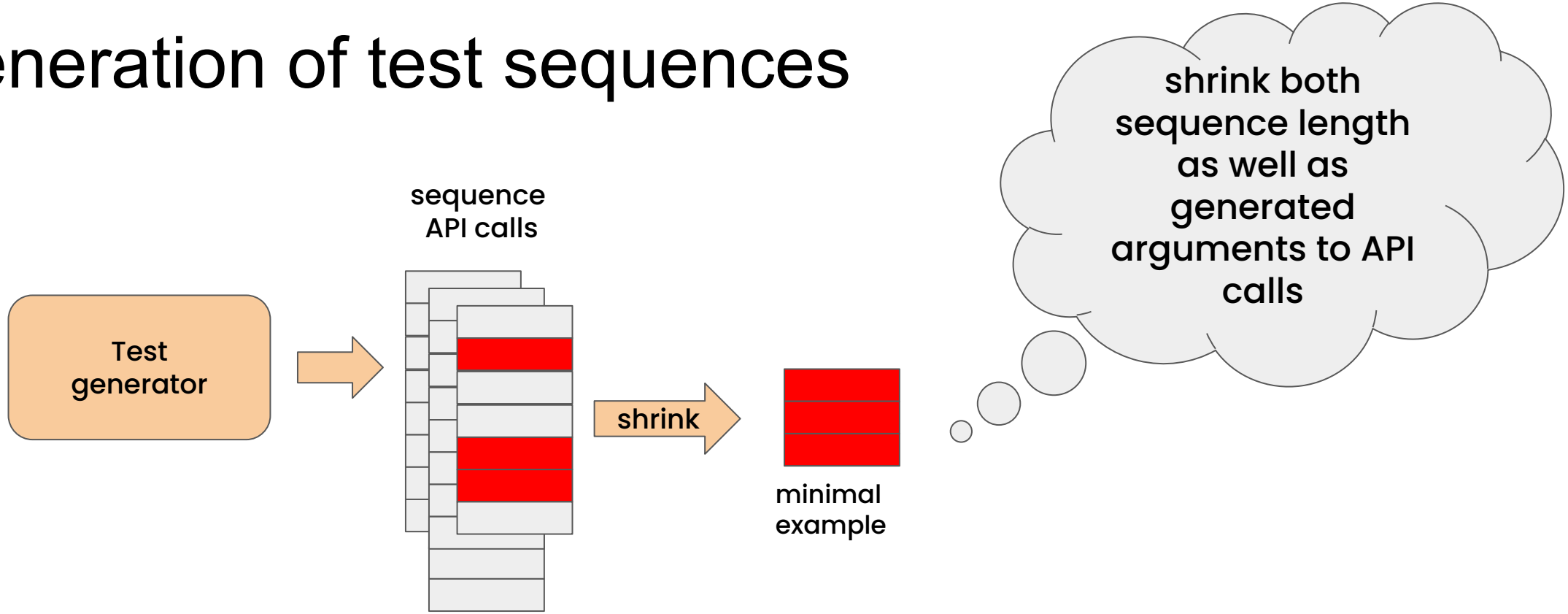
Generation of test sequences



Generation of test sequences



Generation of test sequences



Scaled to industrial examples

More than 10 years of R&D to adapt to industrial needs

protocols, base stations, switches, first response systems, distributed databases, video on demand servers, video conferencing, file synchronization (e.g. dropbox), messaging, automotive software, financial software, web services, railway applications, smart contracts, factory automation, ….

Scaled to industrial examples

More than 10 years of R&D to adapt to industrial needs

3,000 pages of specifications

20,000 lines of QuickCheck

1,000,000 LOC, 6 suppliers

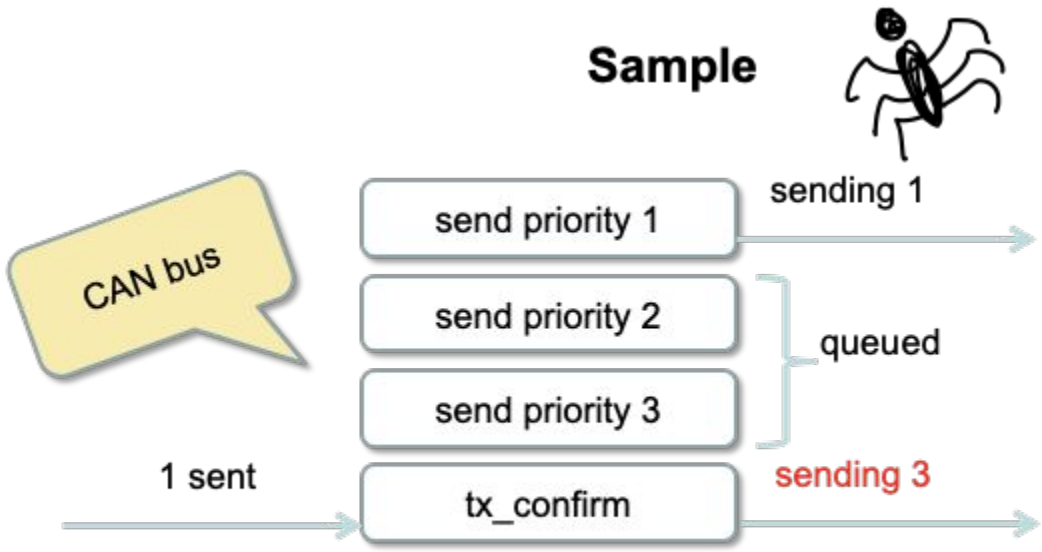
200 problems

100 problems in the standard

10x shorter test code

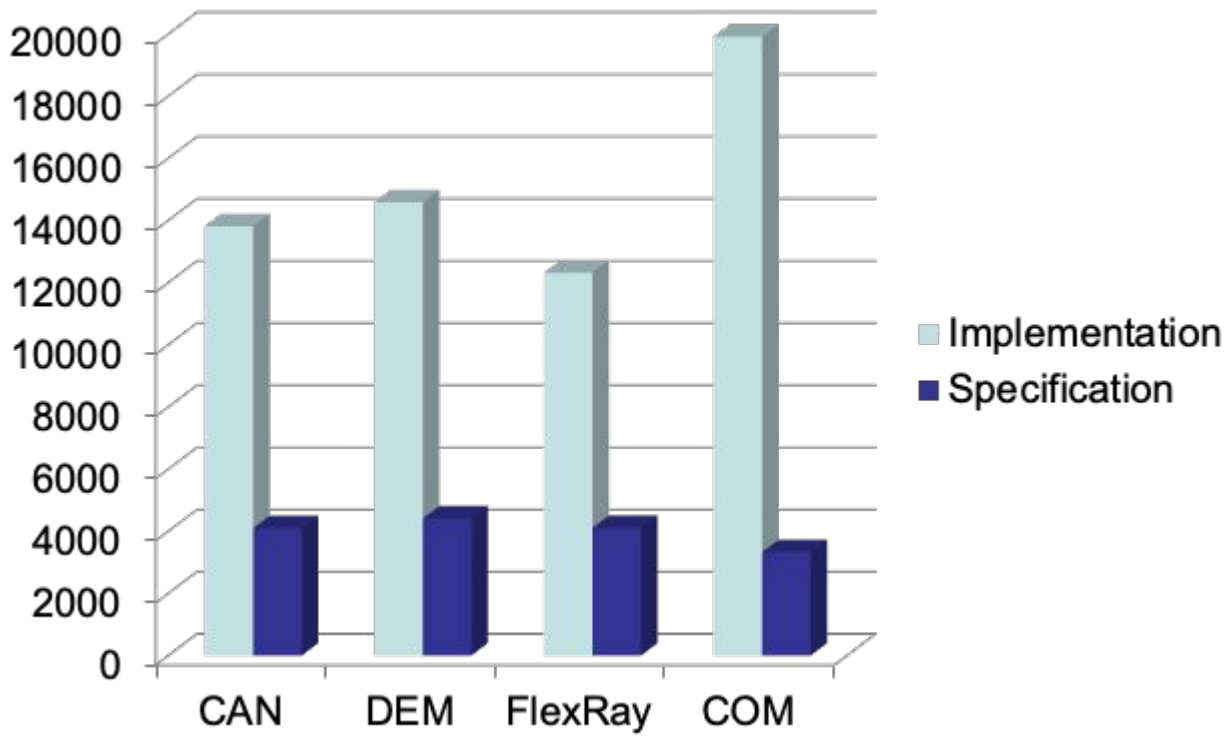
Scaled to industrial examples

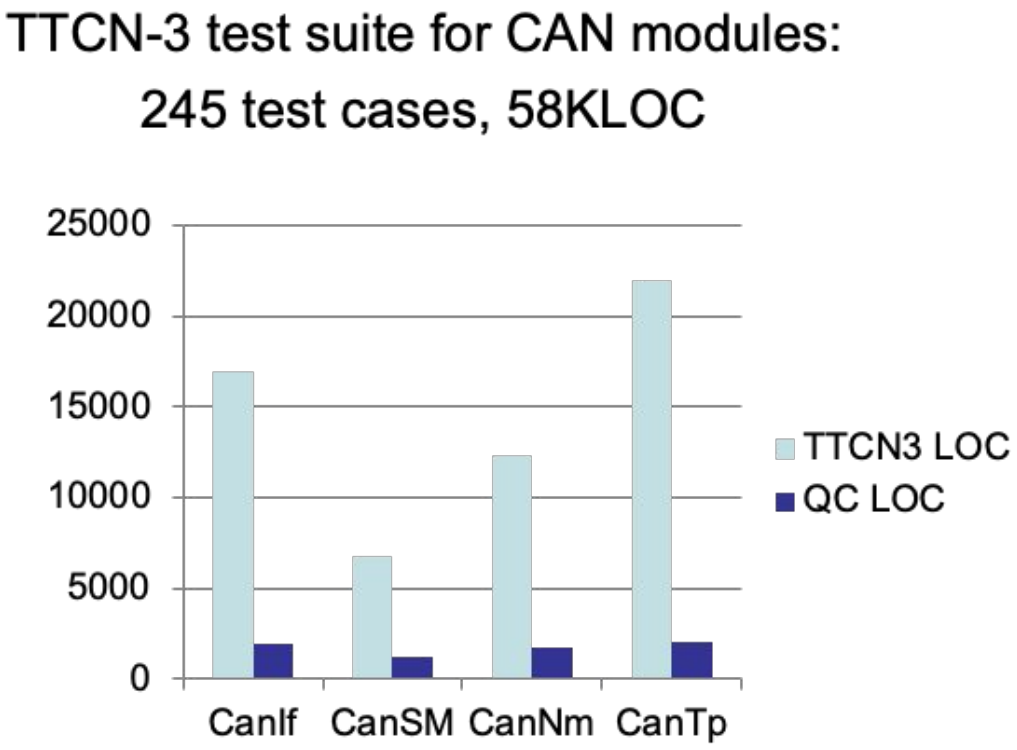
Sequences reveal faults



Cause: failure to mask a bit off an extended CAN-identifier

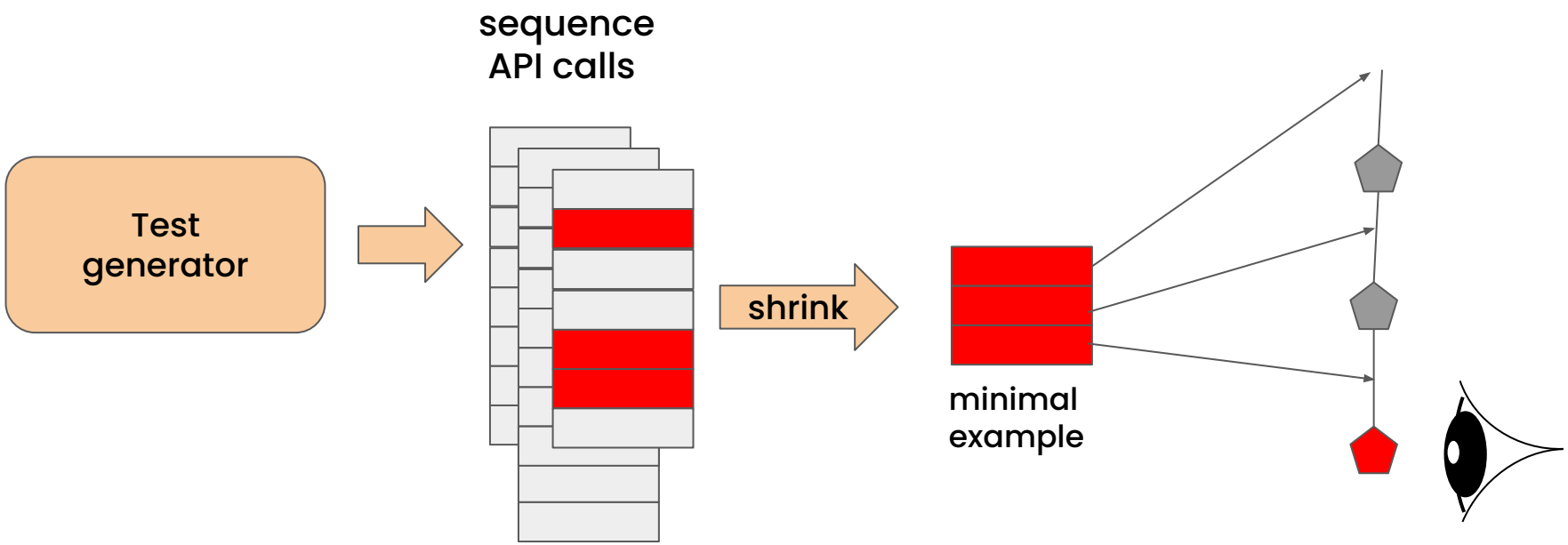
Scaled to industrial examples



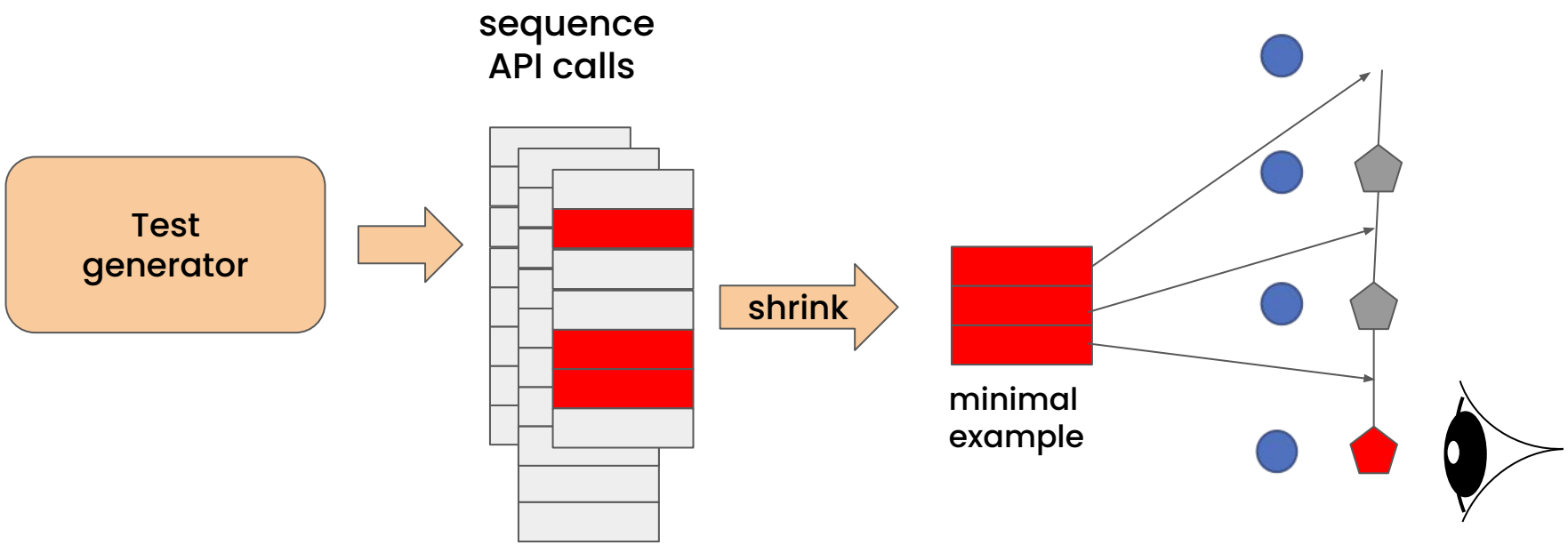


T. Arts, J. Hughes, U. Norell and H. Svensson, "Testing AUTOSAR software with QuickCheck," *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Graz, Austria, 2015, pp. 1-4,

Generation of test sequences



Generation of test sequences



Generation of test sequences

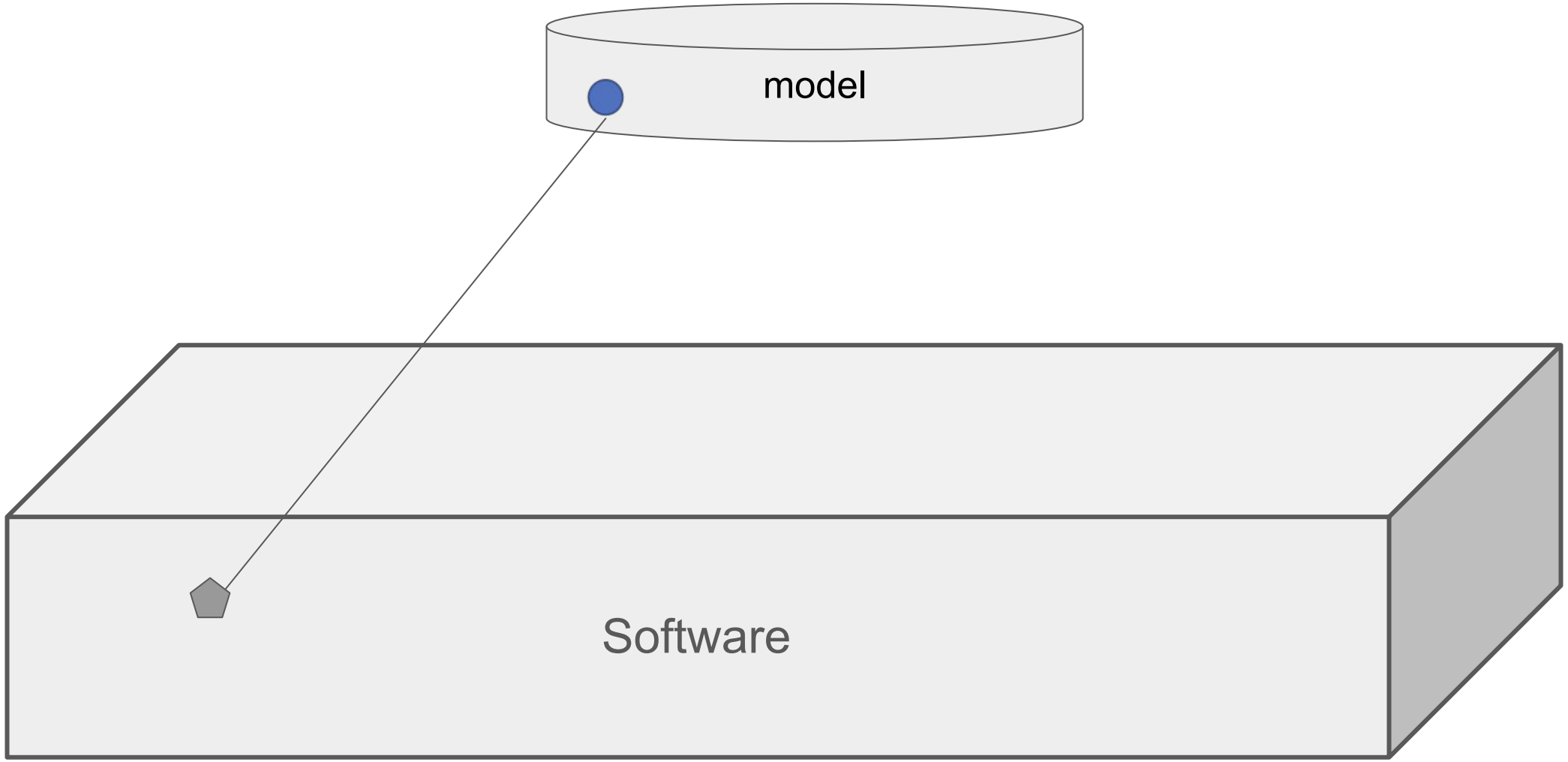
Specification is stateful model for API

initial state
for each API
precondition: possible in this state?
generate arguments for the API call
next state: update the model state given the call
postcondition: is SUT result comptable with model state

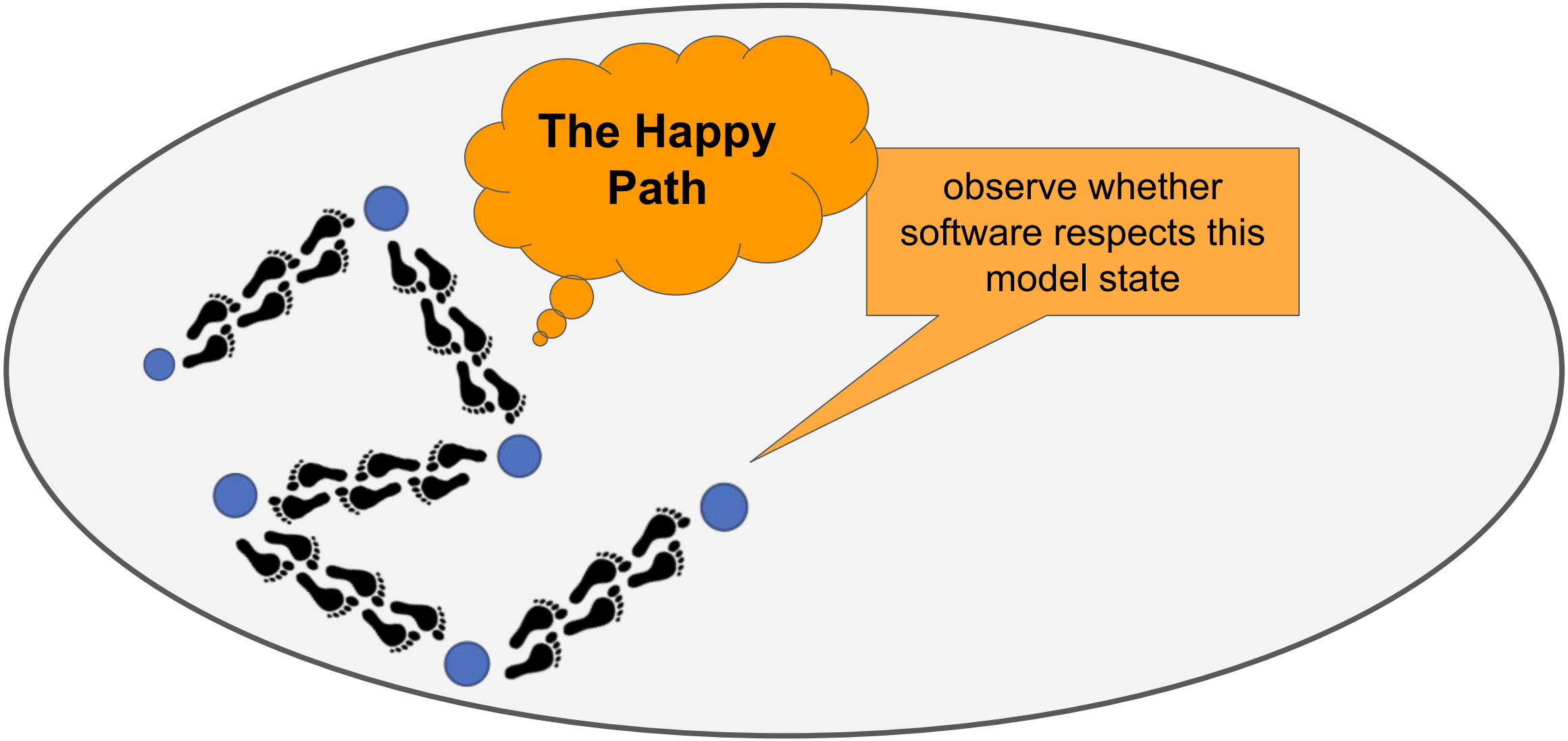
Important:
choose right level
of abstraction for
model

specification
linear in
number of API
calls!

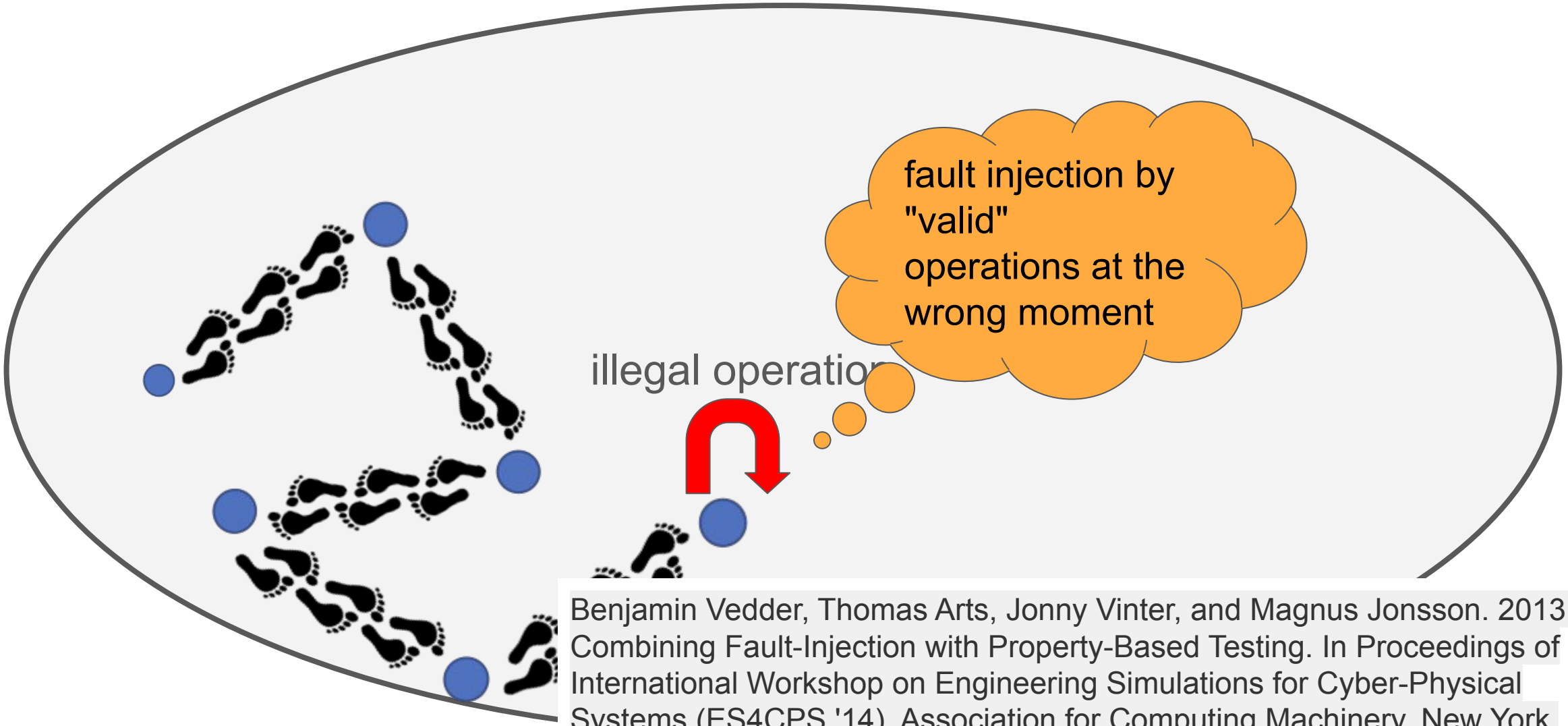
Specifications: a model of the software



Visit random states the software can be in

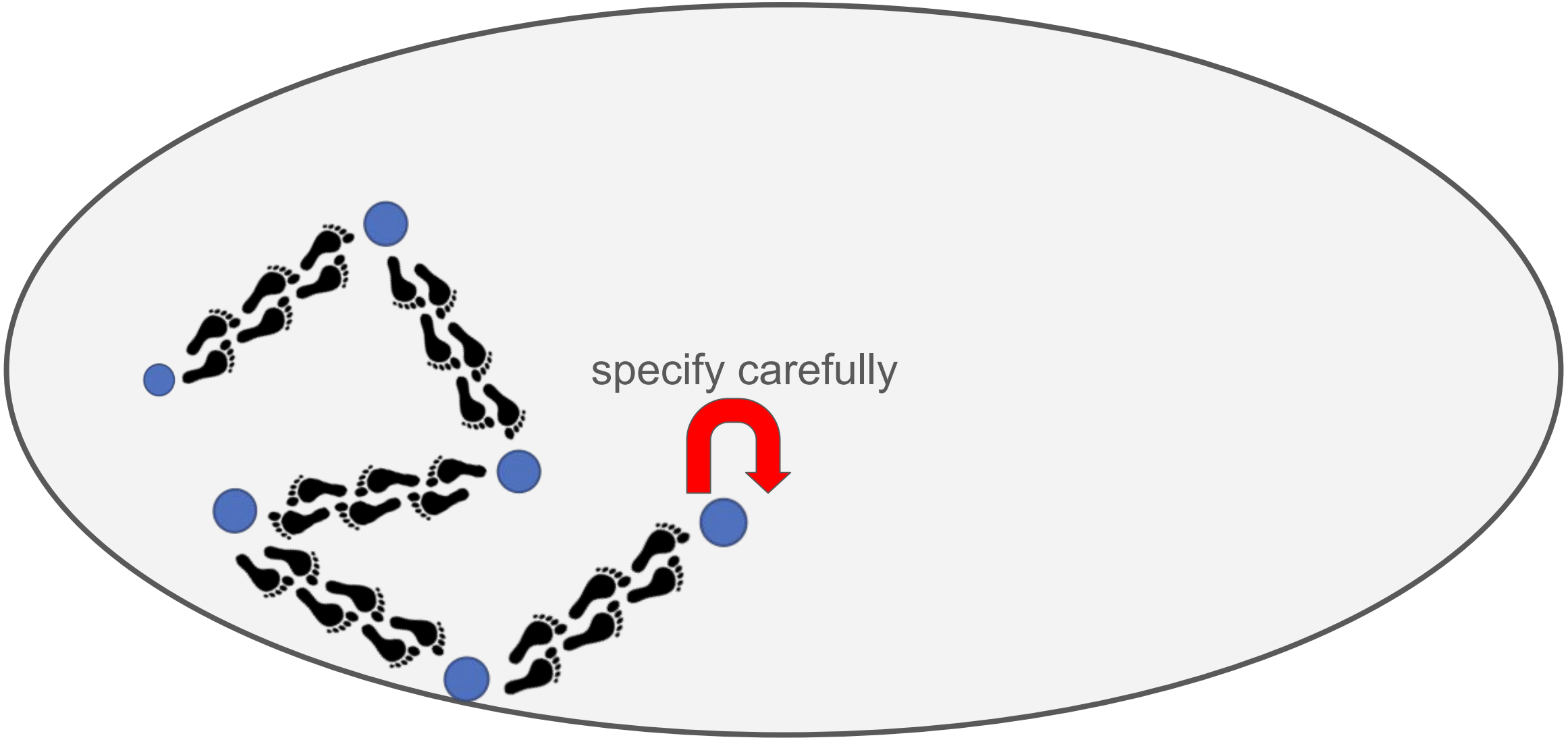


Negative testing for free!

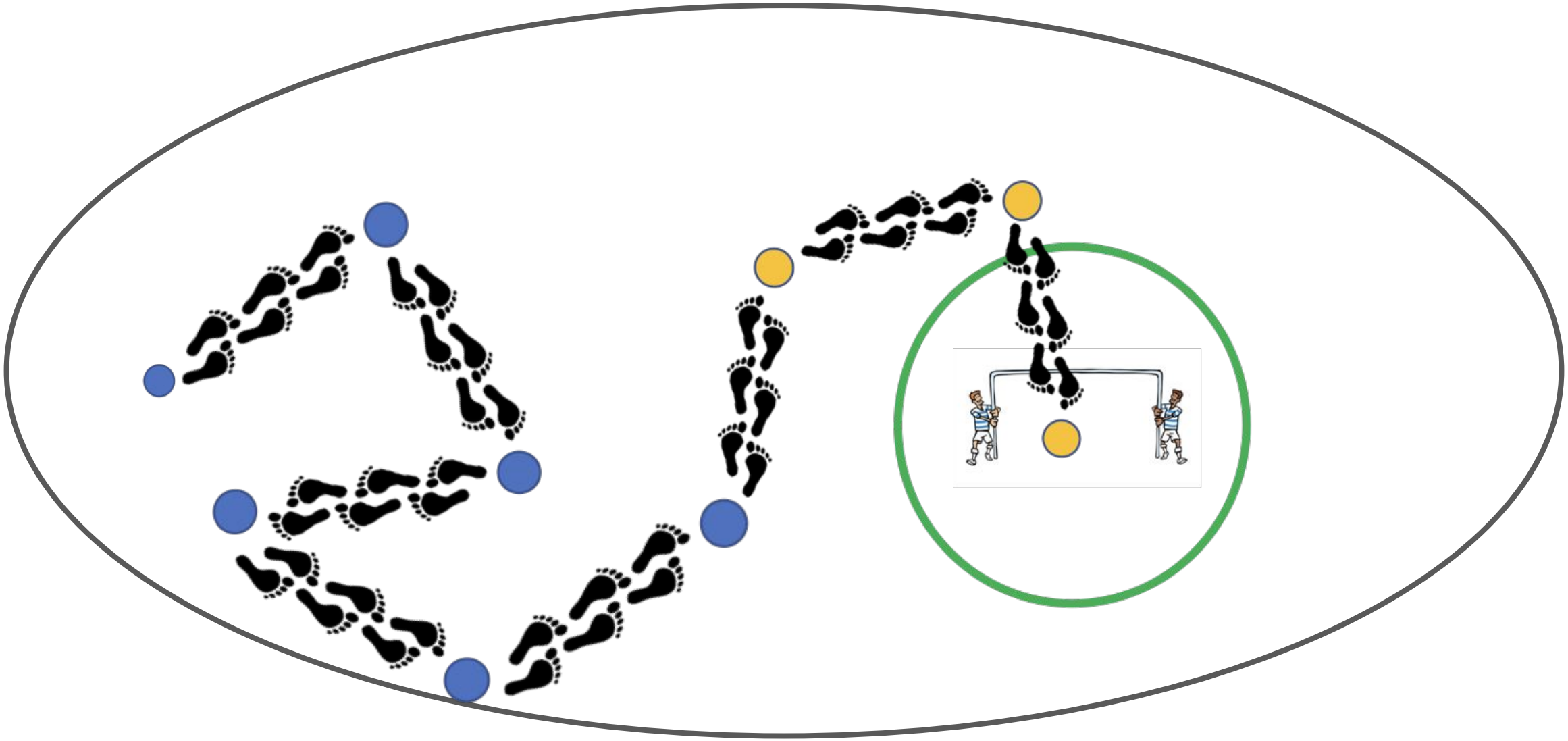


Benjamin Vedder, Thomas Arts, Jonny Vinter, and Magnus Jonsson. 2013. Combining Fault-Injection with Property-Based Testing. In Proceedings of International Workshop on Engineering Simulations for Cyber-Physical Systems (ES4CPS '14). Association for Computing Machinery, New York, NY, USA, 1–8.

Threat model: subtle modifiers of actions



Security requires more…



Strategies for eventuality properties

Model language to express **strategies**:

From any state we are in… this is how we get to the goal

This forces developers to describe they covered all the cases… and it can be tested that so is the case

Software certification

CARDANO

- should specify model to cover functional behaviour
  - covers both positive and negative test cases
- should specify threats using threat model
- should specify necessary eventuality properties

Model is inspected by certifiers, thousands of tests are automatically generated to verify that the software respects the model. Coverage used to double check that there is no bias in tests.