

Taming the dragon: bootstrapping business processes DSL for testing business processes DSL

Ulyana Tikhonova



The Xlingq DSL ecosystem



Query



Initialization



Permissions



Data persistence



Presentation of data



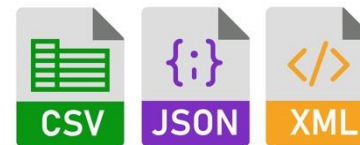
UI Testing



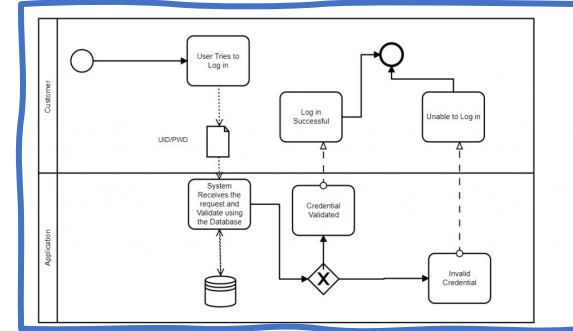
Configuration



External APIs



Data mappings



Business processes

The Xlingq DSL ecosystem

```
module tcs.cosmo
specification Galaxy of kind businessactor from template BusinessActor
  fields
    identifier id : String autogen
    name : String
    dominatingCivilization : String
    involvedParty : InvolvedParty
    ...
  states
    created
    final singularity
  transitions
    transition behold
      parameters
        set name : String maxlength 20
        set serviceProvider : Universe
      preconditions
        expect serviceProvider is initialized
        fail with error DEF0000
      postconditions
        set galaxy = this
        set specName = "Galaxy"
        set uniqueField = call uuid2string(Identifier())
```



```
composablePage galaxiesCloneDetail {
  type: detail;
  variant: horizontal;
  datapoint: Galaxy;
  ...
  component headerActions {
    type: text
    variant: rightContent
    datapoint: Galaxy
    title: ""
    actions {
      setGalaxyAgreements @Icon("control_point")
```



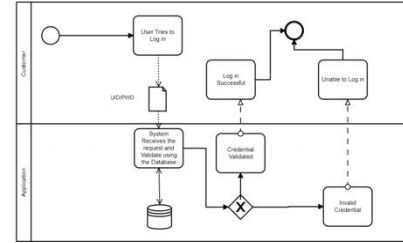
```
test GivenLoginThenViewPageWithMandatorySearch
  given login darthSidious
  given view page moonsDashboard
  then check rowCount is 0

  when enter "Fifth" in search field Moon.name
  when click applyfilter
  then check rowCount is 1
```



```
    con("thumb_up_outlined"),
    ("check_outlined"),
    on @Icon("check_outlined")
```

DSL for business processes



```
orchestration tcs.octo.LongRunningExpansion;  
import tcs.cosmo.Galaxy;  
import tcs.cosmo.GalaxyAgreement;
```

```
external processes {  
  tcs.octo.SubProcess;  
}  
mappers {  
  galaxiesCsv: GalaxiesCSVMapper;  
}
```

```
inputfields {  
  id: Galaxy;  
  file: File;  
}  
outputfield {  
  out: Boolean;  
}
```

```
process {  
  val galaxy : Galaxy == Galaxy[id.id];  
  if (galaxy is created) then  
    val agreement == GalaxyAgreement.create(galaxy);  
  endif  
  
  foreach(row in galaxiesCsv.getRows(file)) {  
    send galaxy.behold(row);  
  }  
  
  call wait(10000);  
  call SubProcess(Galaxy["NGC4414"], 100);  
  terminateProcess(wait(20000));  
}
```

DSL for k

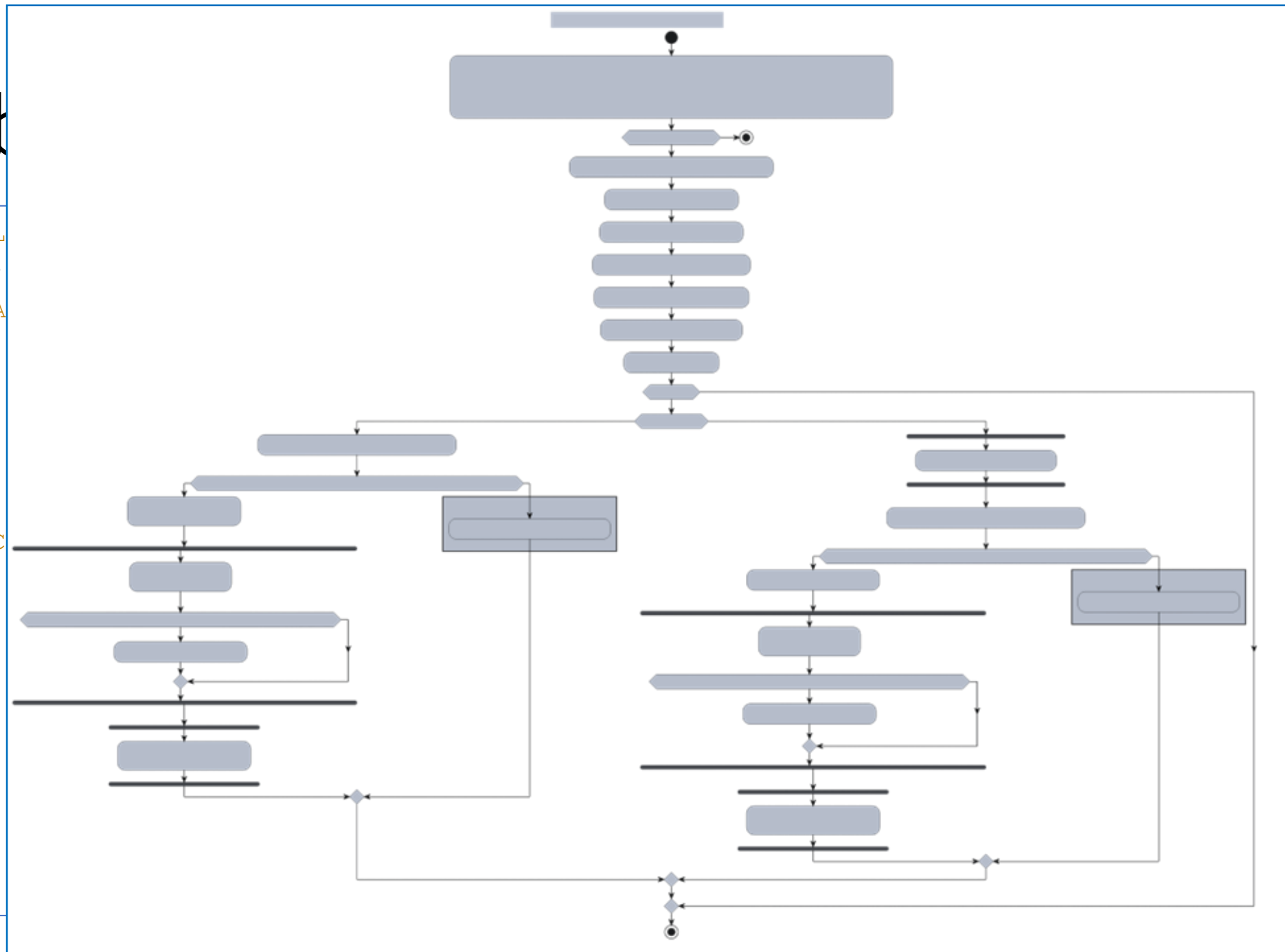
```
orchestration tcs.octo.L
import tcs.cosmo.Galaxy;
import tcs.cosmo.GalaxyA

external processes {
  tcs.octo.SubProcess;
}

mappers {
  galaxiesCsv: GalaxiesC
}

inputfields {
  id: Galaxy;
  file: File;
}

outputfield {
  out: Boolean;
}
```



Functional test of business processes

- To validate behaviour without the browser/UI
 - File-based flows
 - Queries and data state
 - API calls

“Receive data from three different files/sources, persist it as 40 different entities in the data base”

“Process/translate the received data; update data base tables and generate the resulting files”

- We need it working in two months!

What do we need to do functional testing?

“Receive data from three different files/sources, persist it as 40 different entities in the data base”

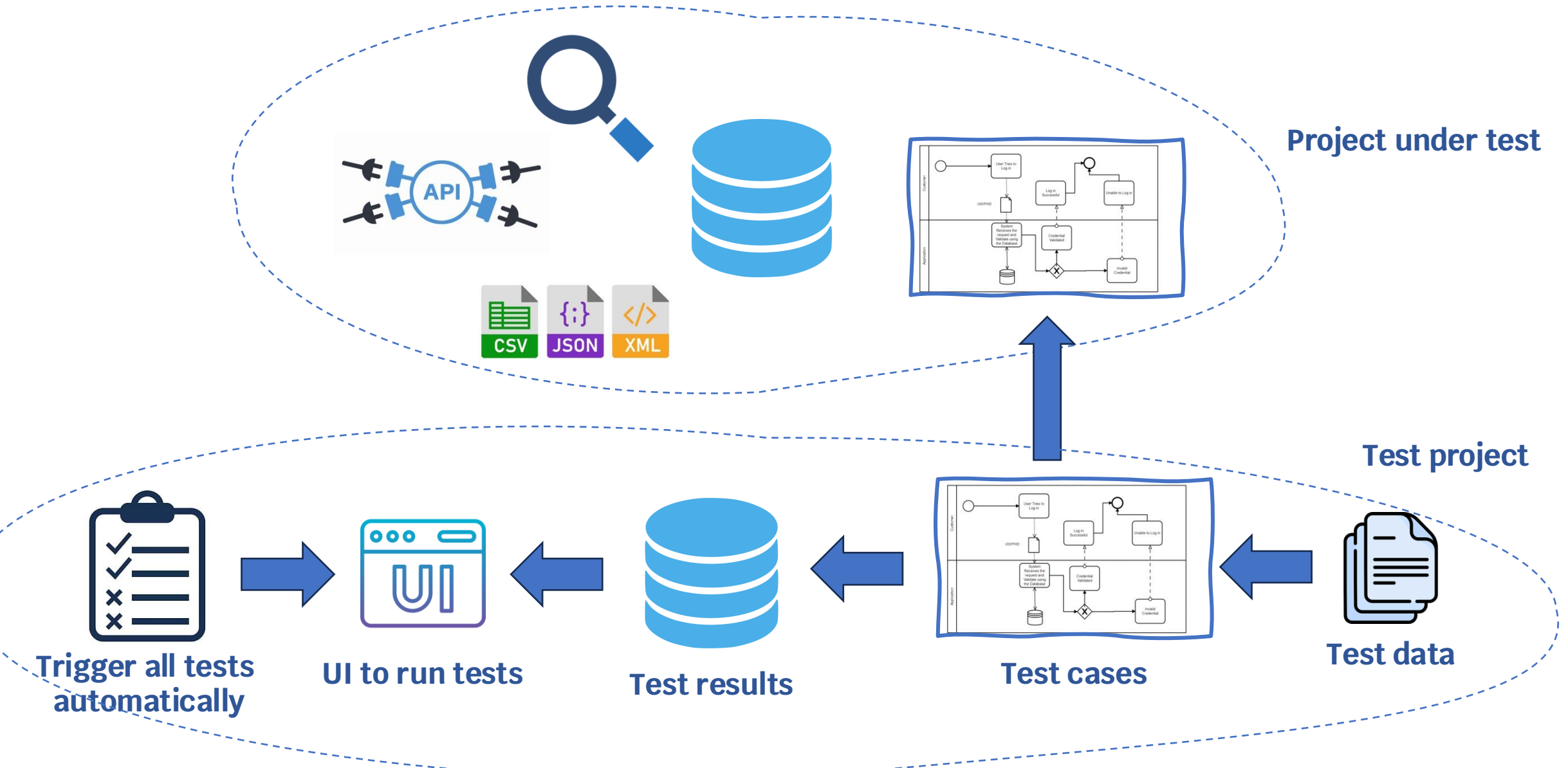
“Process/translate the received data; update data base entities and generate the resulting files”

- Read and write files
- Connect to external APIs
- Invoke a business process
- Query data base
- Send transitions to data base
- Assert values



**We already have it in our
business processes DSL!**

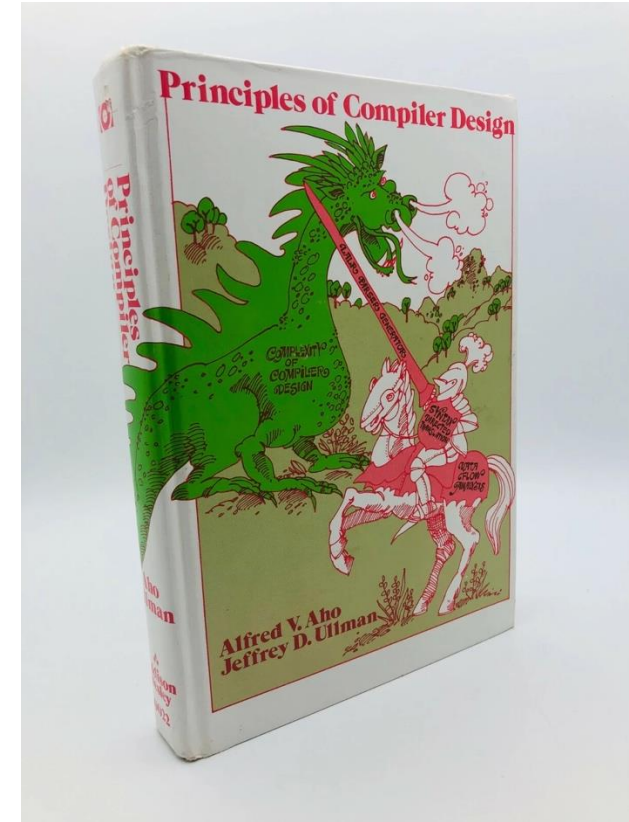
Bootstrapping set-up



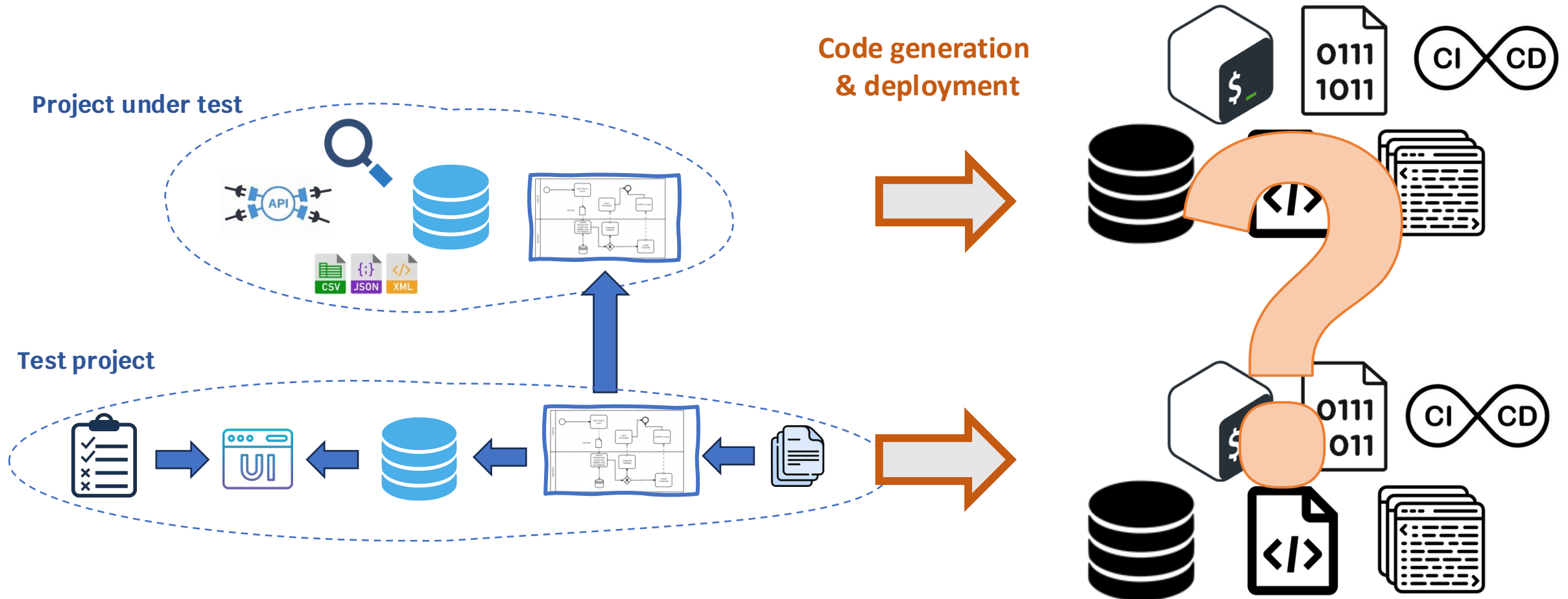
Demo

What it takes to tame the dragon?

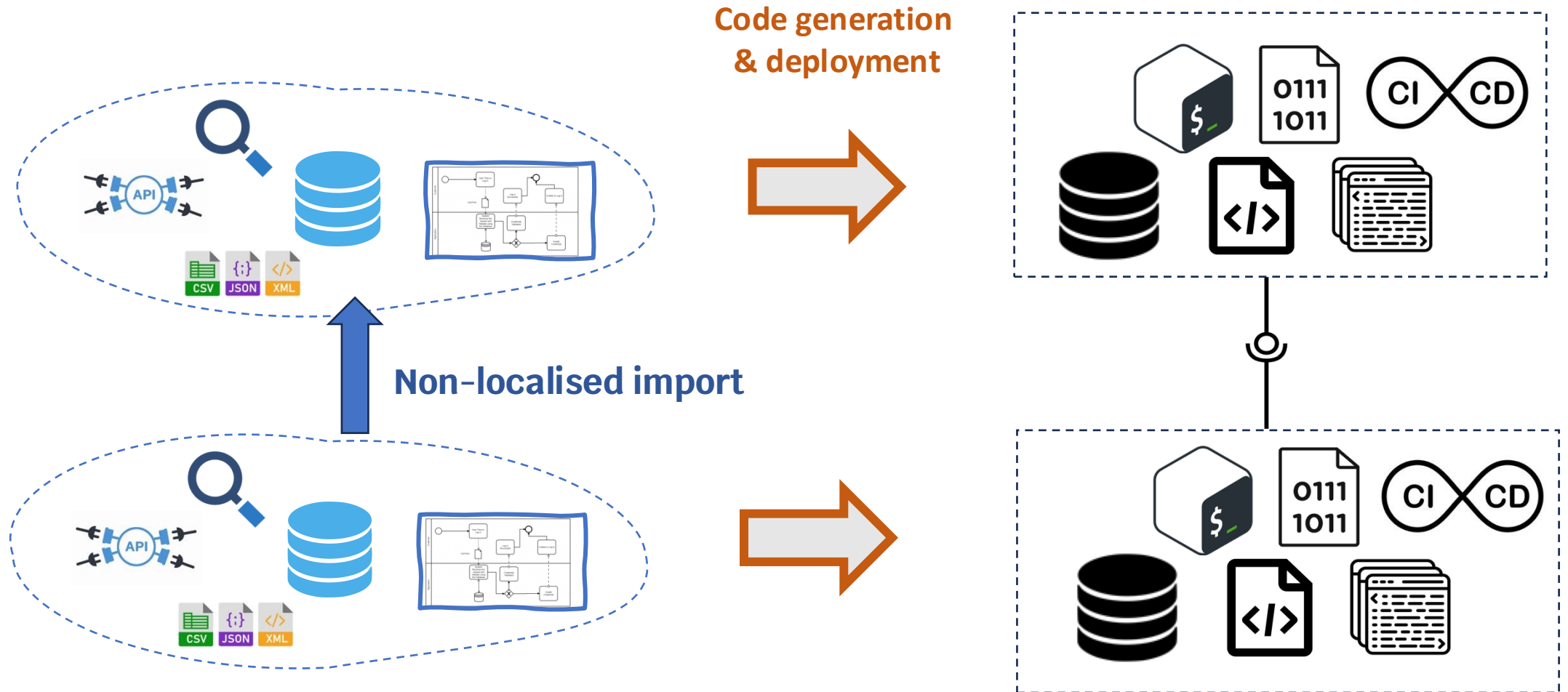
- Technical challenges
 - How modular are our DSLs?
- Expressiveness
 - Eat your own dogfood



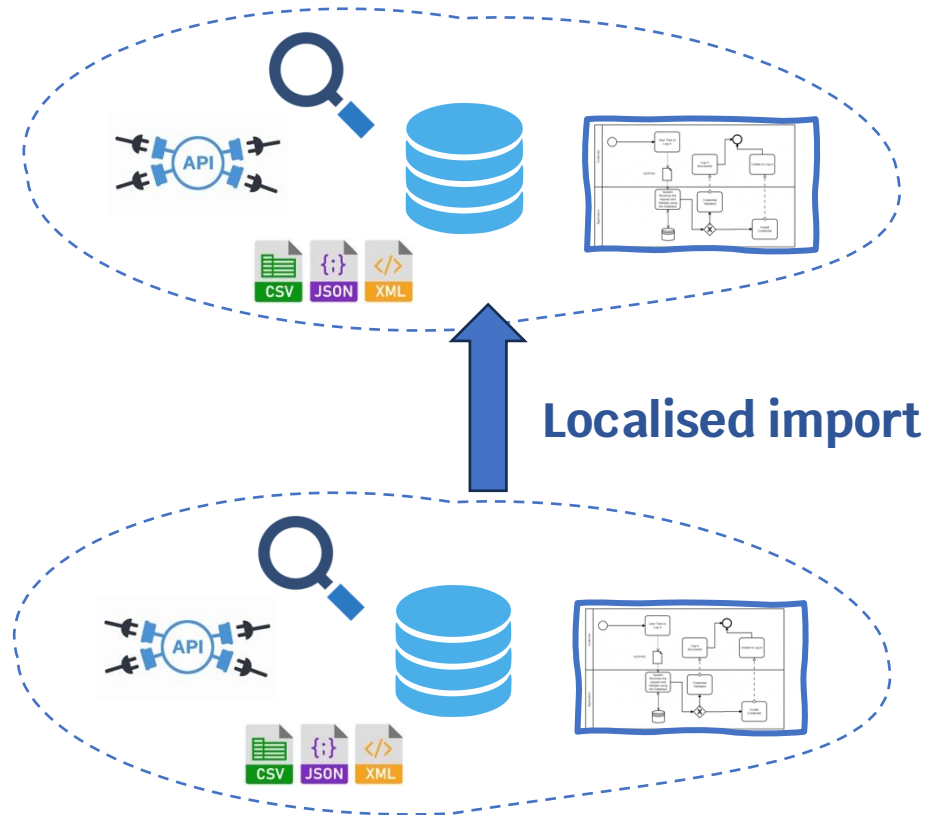
What does it mean to import a project?



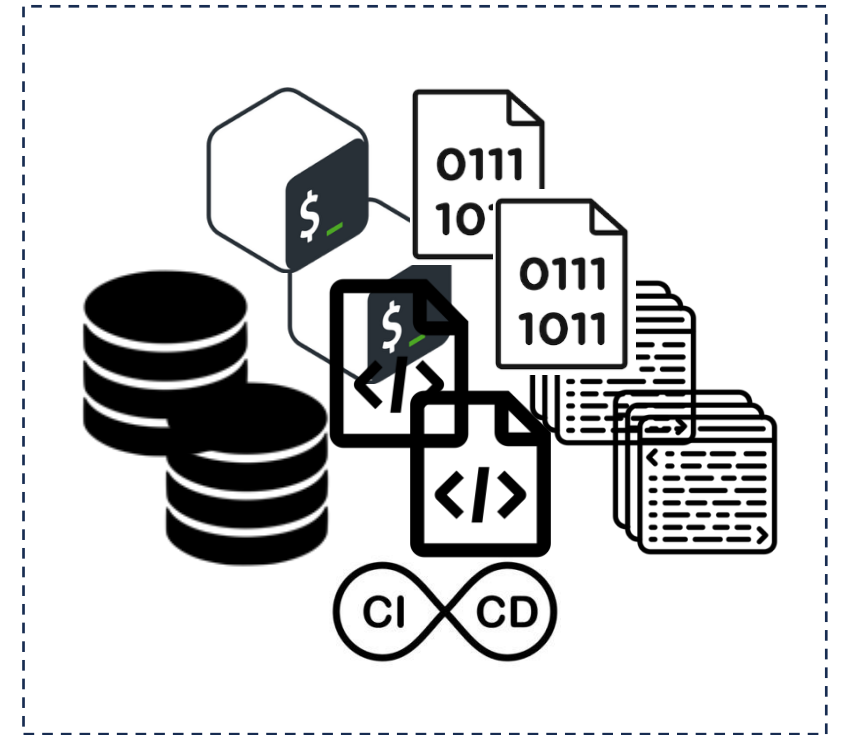
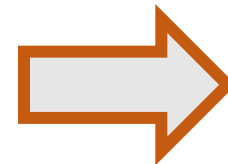
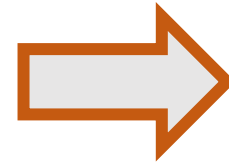
Non-localised import



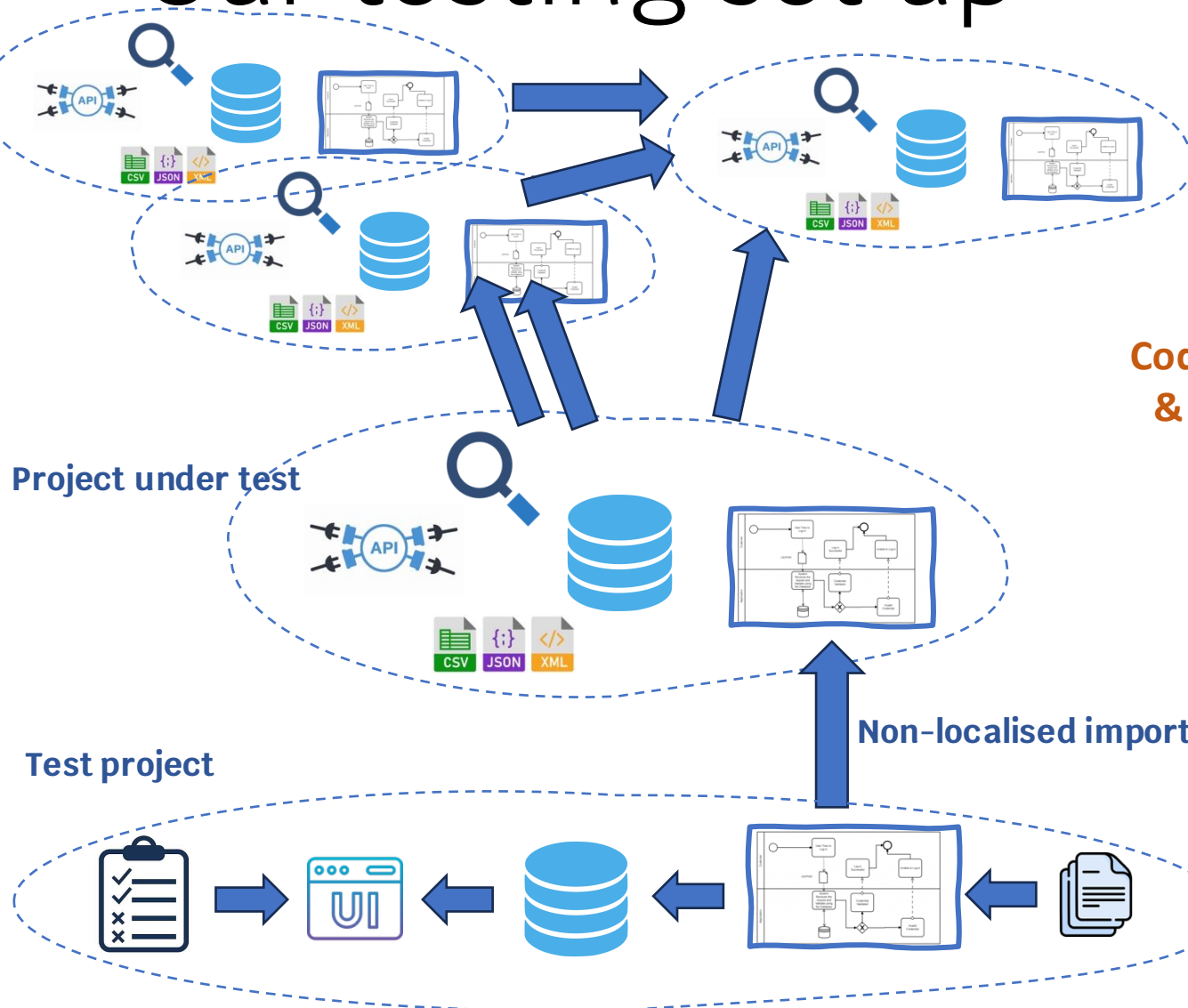
Localised import



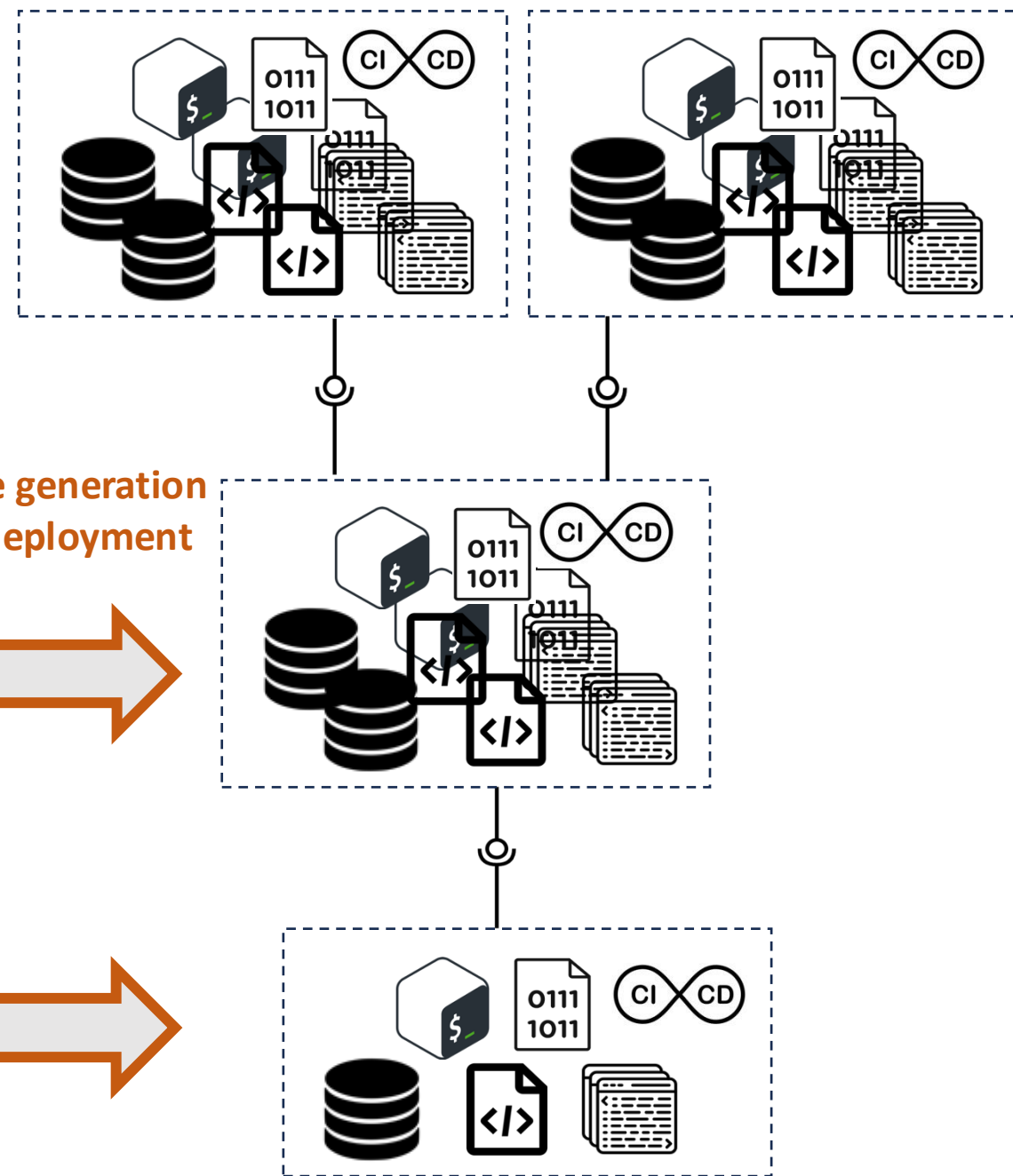
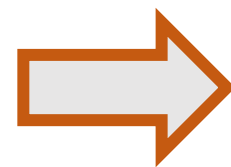
Code generation
& deployment



Our testing set up

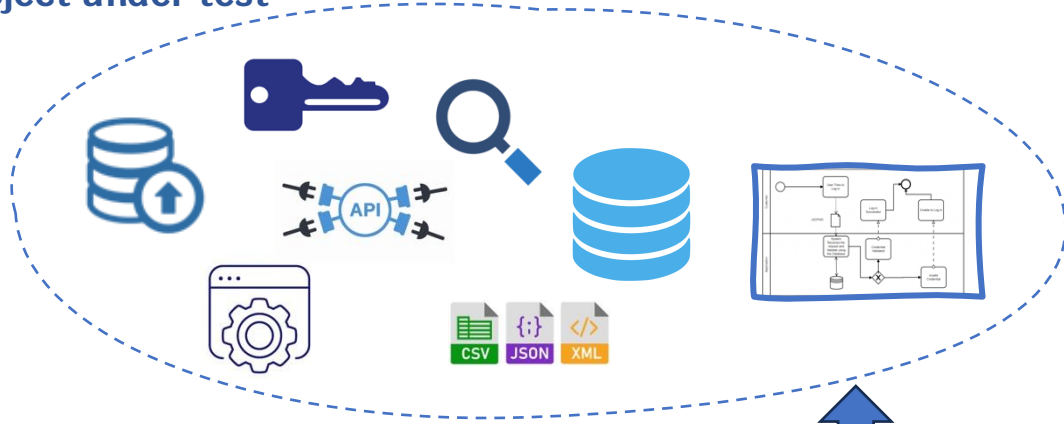


Code generation
& deployment

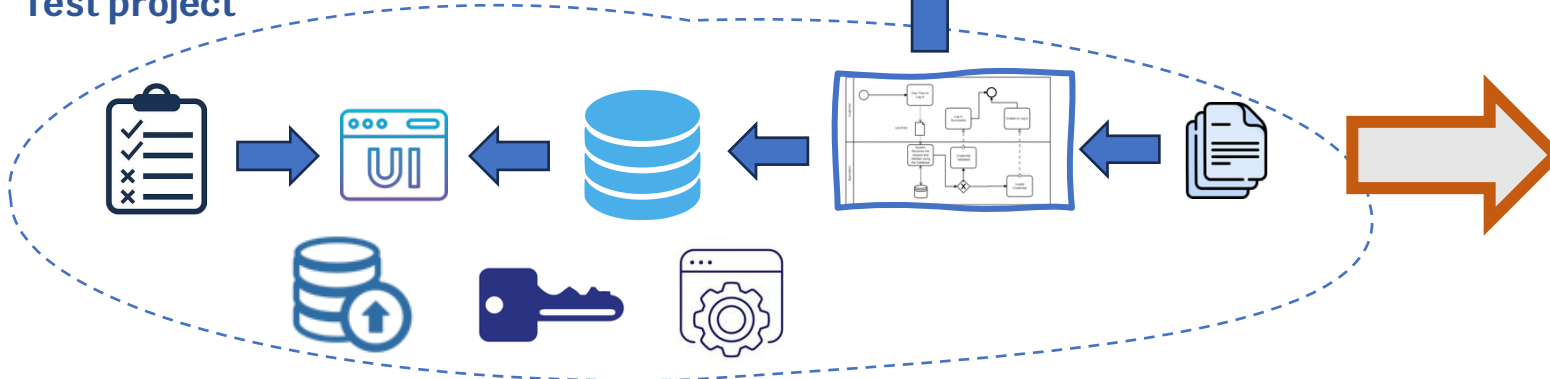


... but there is more

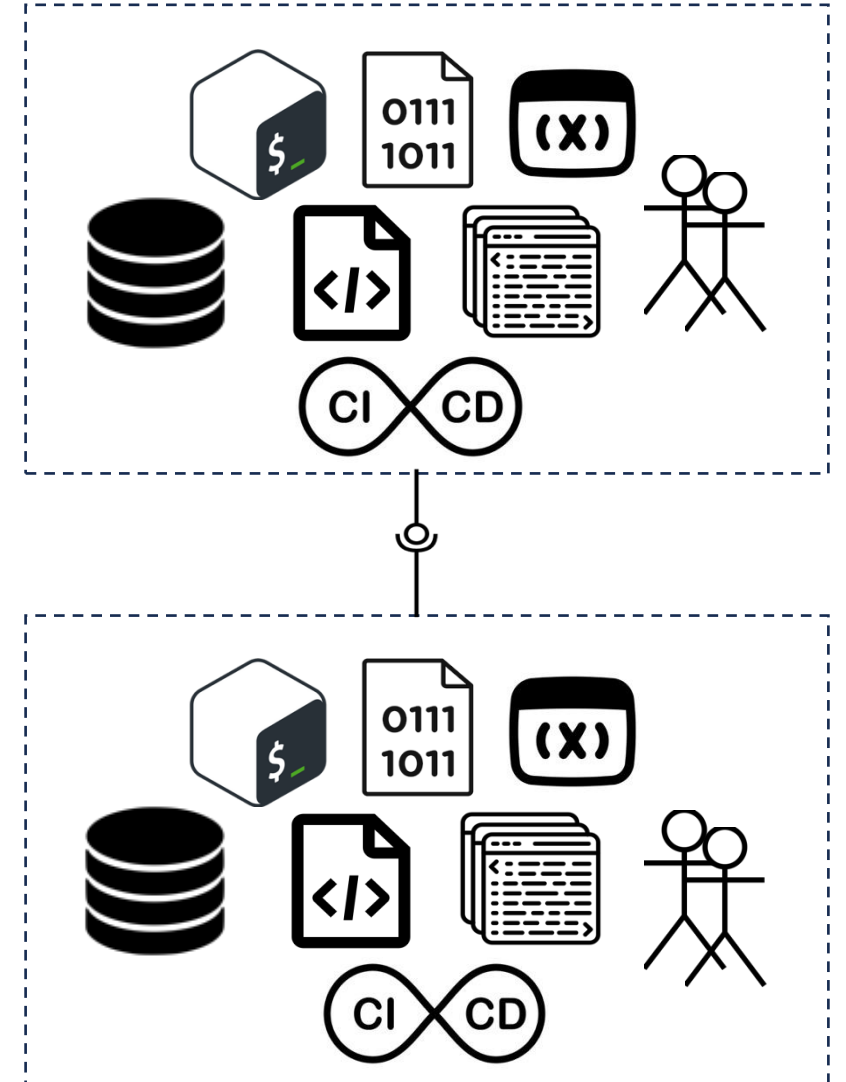
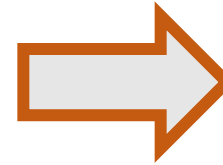
Project under test



Test project



Code generation
& deployment



Expressiveness: what works well

- Loading test data from input files
- Assertions as queries for the data base
- Storing test results in a data base
 - Careful with the timestamps
 - Export as a report file

Expressiveness: what doesn't work well

- Executing the whole test suite
 - We don't have a collection of business processes
- Assigning a separate input file per test case
 - We don't have a collection of files
- Assertions flow with the intermediate test results

Assertion	True	False
Does "X" exist?	checked, go on	Fail test "message A"
Does "Y" exist?	checked, go on	Fail test "message B"
Is field "Z" equal to "123"?	checked, go on	Fail test "message C"
...		

Conclusions

- Bootstrapping for functional testing
 - 33 test cases
 - happy flow & bad weather
- From the idea to the working tests: 3 months
- Limited expressiveness
 - Import, set of test cases, assertions flow
- Adoption of the approach by business engineers
 - Need support of meta-programming in their own DSL
- The pipeline as a whole
 - Every change in the project under test should be deployed first, before it can be tested

