

# Test Automation

Daniel Ståhl  
daniel.stahl@liu.se

# Who am I?

## DANIEL STÅHL

Developer

Configuration manager

Technical coordinator

Architect

Researcher

Subject matter expert

...

## ERICSSON

Strategic studies

AI strategy

## RESEARCH

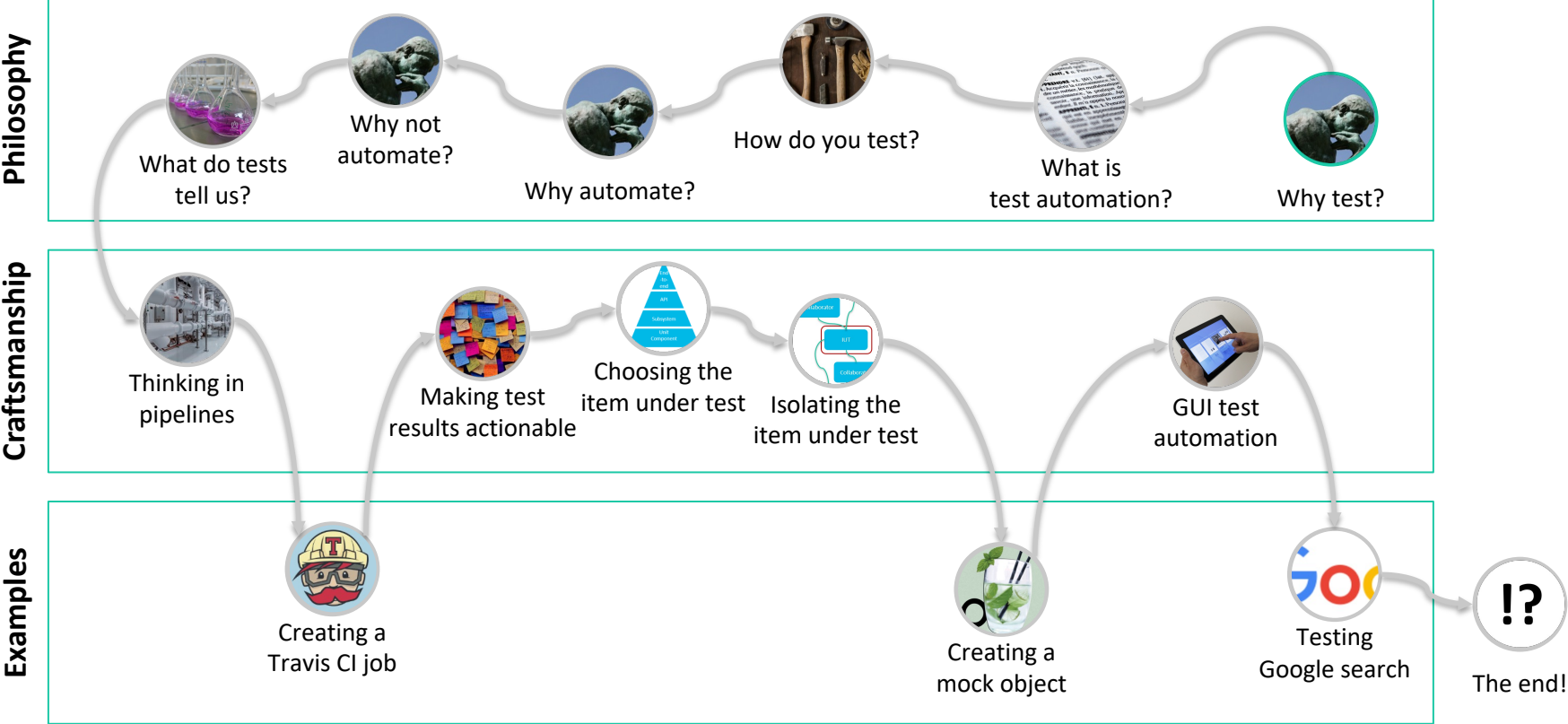
Continuous practices

Software engineering  
practices

PhD from Groningen, NL  
Books

Associate Professor

# Things to cover



# Why test?



## **WE CLEARLY WANT TO FIND FAULTS...**

These could be small.

They could also be major.

They could be about the user experience.

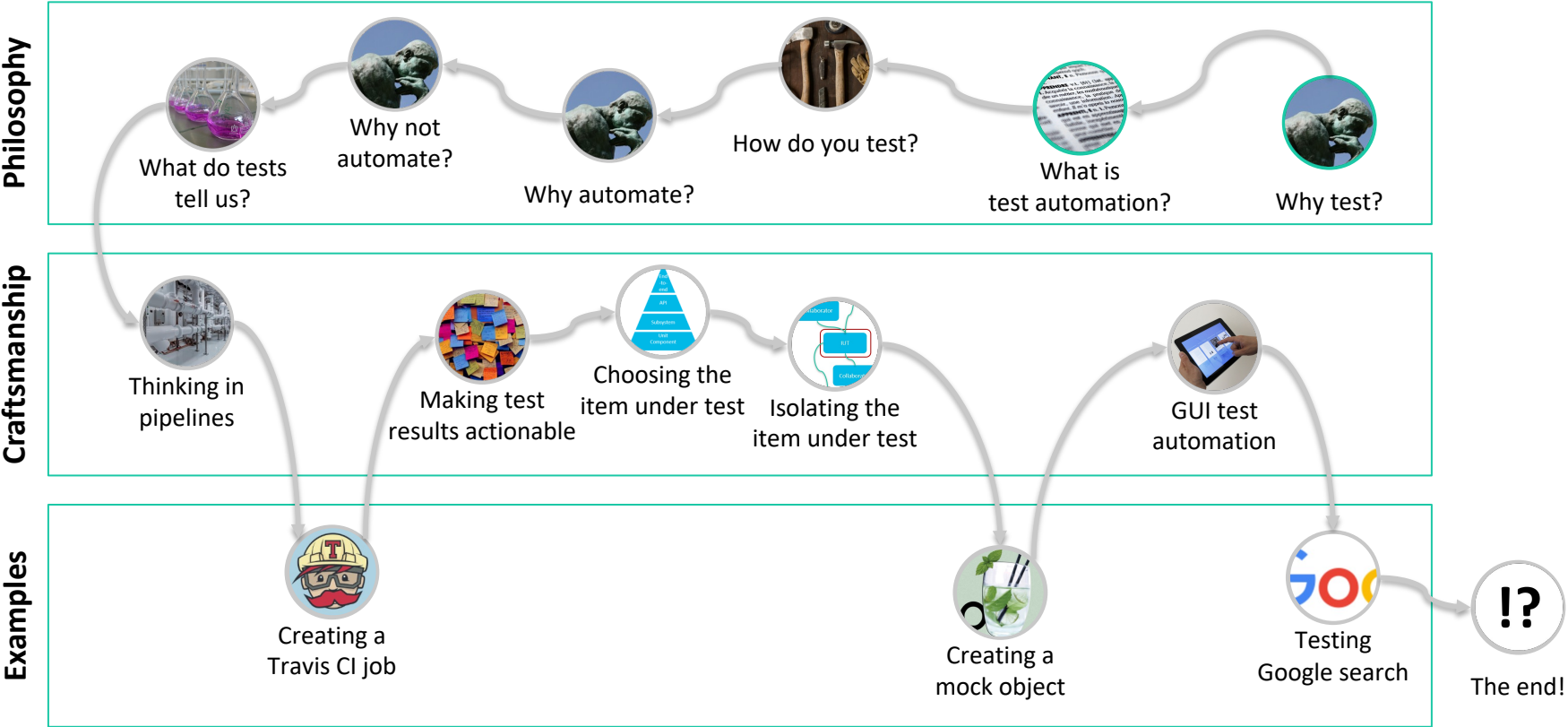
They could also be about life and death.

## **... BUT TESTING IS NOT JUST ABOUT FAULTS**

Testing is interacting to learn about the Item Under Test.

There are many things we might want to learn.

# Things to cover



# What is test automation?



## AUTOMATIC EXECUTION OF TEST SOFTWARE

If testing is interaction to learn, then automated testing is automated interaction.

Or...  
Execution of software separate from the system being tested (Item Under Test, IUT), for the purposes of evaluating that Item Under Test.

## SPECIFIC TOOLS, FRAMEWORKS AND ENVIRONMENT

Apart from the automated test cases, you need tools for...

- ... scheduling
- ... deployment
- ... execution
- ... simulation
- ... data collection
- ... data analysis
- ... presentation
- ...

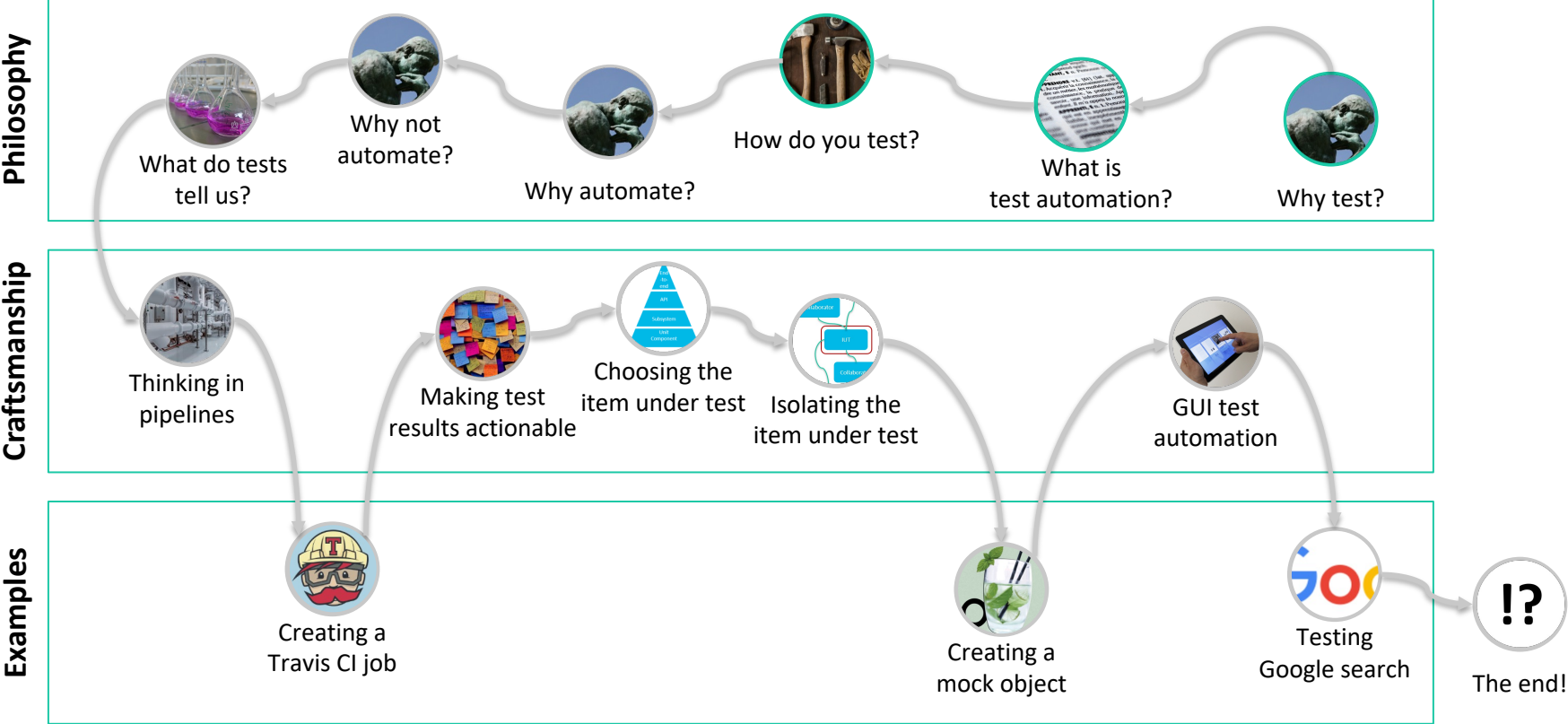
## OUTCOME COMPARISON

Typically the outcome is compared to some expected result.

Either the test case succeeds in matching this result, or it fails.

KPIs may also be measured. Particularly relevant for non-functional requirements.

# Things to cover



# How do you test?



## **METHODOLOGY VARIES**

The type of Item Under Test (IUT) matters!

- ... Is it a GUI?
- ... Is it an API?
- ... Is an embedded system?
- ... Is it a physical interface?
- ...

## **AT WHAT LEVEL AND MATURITY ARE YOU TESTING?**

Are you testing the whole system, or a small piece of it?

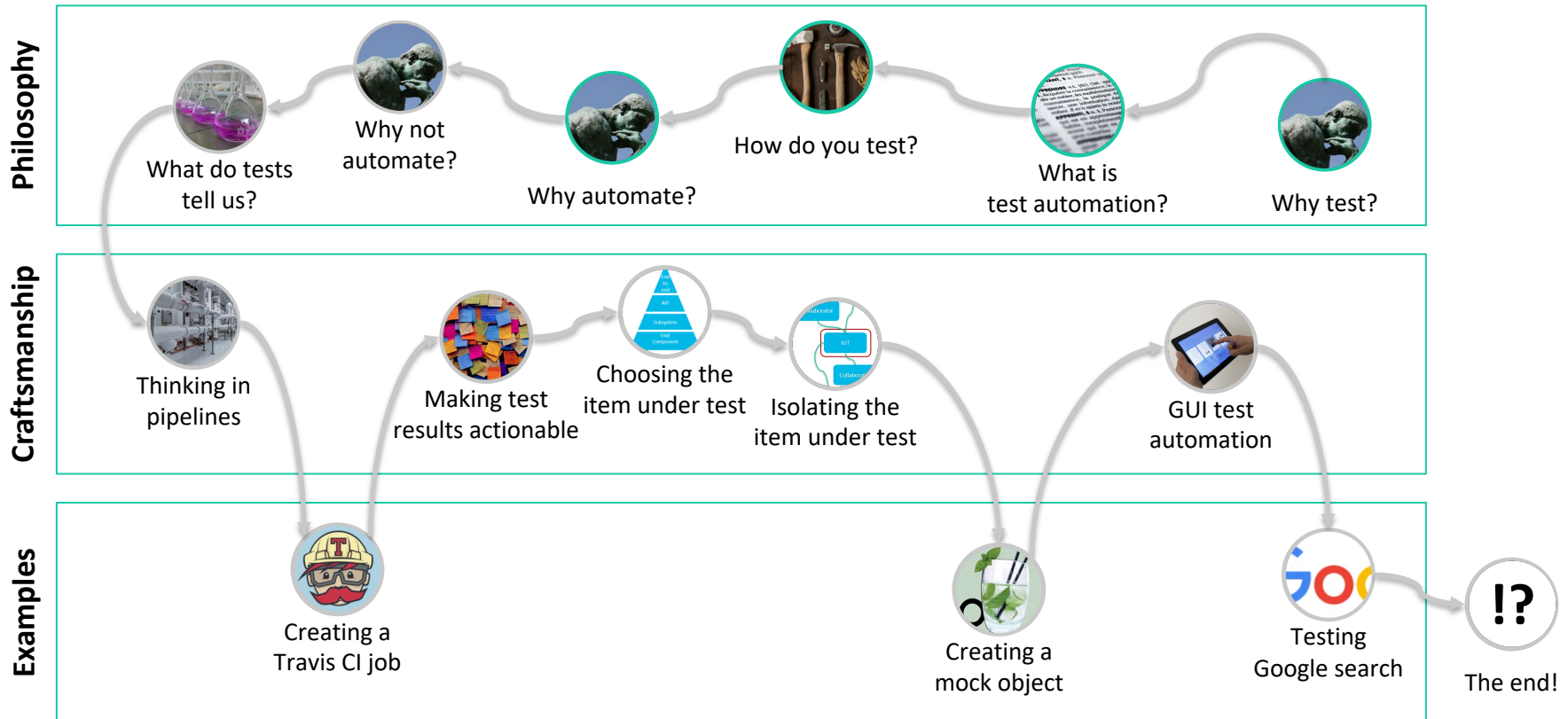
Is the IUT an early prototype or a mature in-service system?

## **WHY ARE YOU TESTING?**

What are you trying to achieve by testing the IUT?



# Things to cover



# Why automate? 1 (3)



## **(HUMAN) TESTERS ARE EXPENSIVE**

In software engineering, the main cost driver is (usually) the engineers' time.

## **(HUMAN) TESTERS ARE INCONSISTENT**

Even with rules and checklists, humans are notoriously inconsistent.

We get bored.

We like to improvise.

We need some creative freedom.

Whether that's good or bad depends on context!

## **(HUMAN) TESTERS ARE SLOW**

Executing algorithms is not what we do best.

# Why automate? 2 (3)



## **(HUMAN) TESTERS HAVE NEEDS**

Training, vacation, lunch breaks, sense of purpose...

Test scripts need none of that.

## **(HUMAN) TESTERS HAVE BETTER THINGS TO DO**

There are many things humans still do much better than machines.

Automation shouldn't be about getting rid of people, but about making the most of them.

## **CONFIDENCE**

Having a wall of green lights tell you everything works after you push a change is really nice...

...but is it a false sense of security?

"It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so."

- Mark Twain

# Why automate? 3 (3)



## **RAPID FEEDBACK**

Automated tests can provide immediate developer feedback.

In unfamiliar code, they serve as guardrails.

Automated tests can catch integration errors early.

In larger systems with interdependencies, you can author tests to protect "your" parts.

## **CONTRACT SATISFACTION**

Automated acceptance tests can help clarify and safeguard contractual agreements.

Automated acceptance tests allow customers and/or stakeholders to track progress in real time.

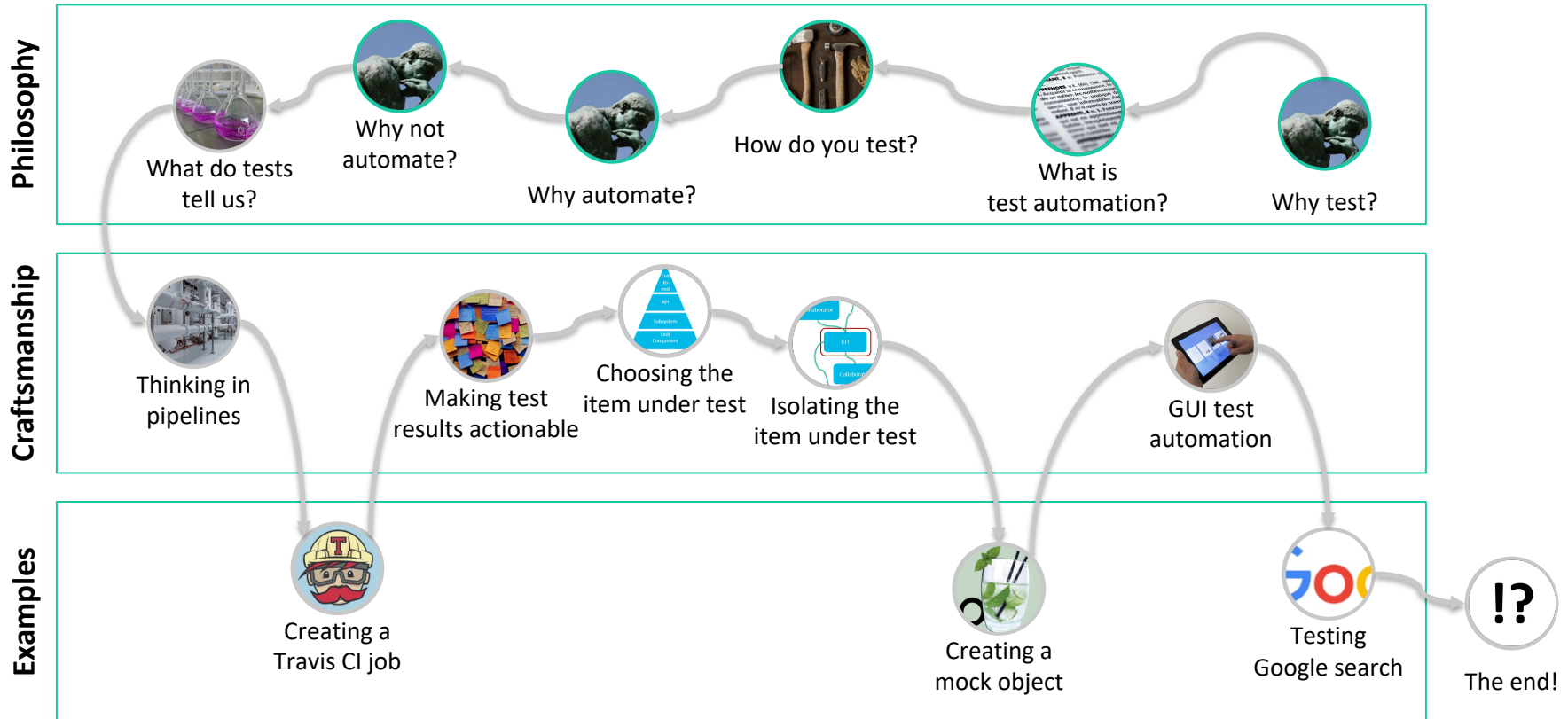
## **BECAUSE IT MAKES YOU LOOK GOOD**

Automated tests produce lots of data.

Data can be turned into fancy graphs and tables.

Fancy graphs and tables help you look like you know what you're doing.

# Things to cover

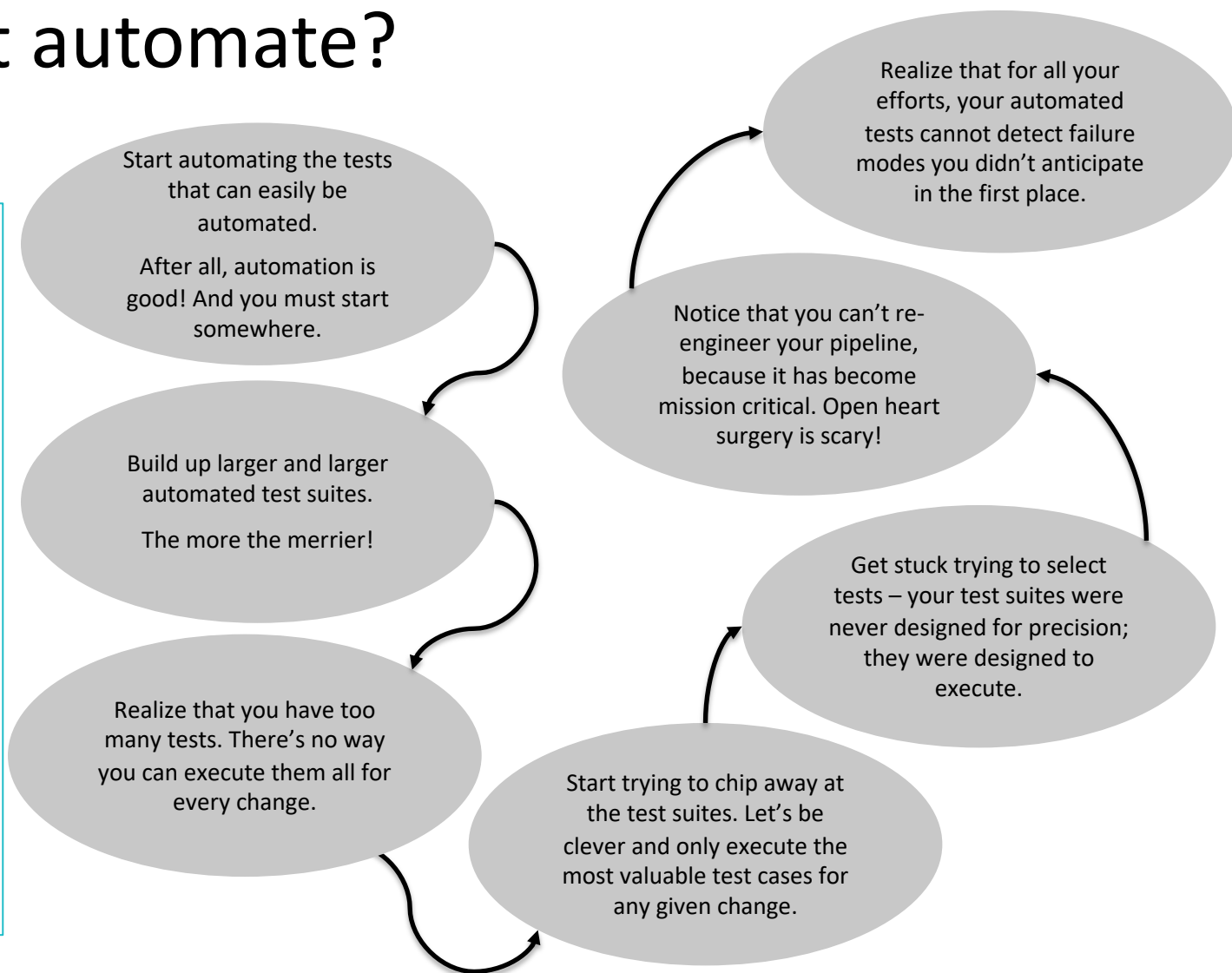


# Why not automate?

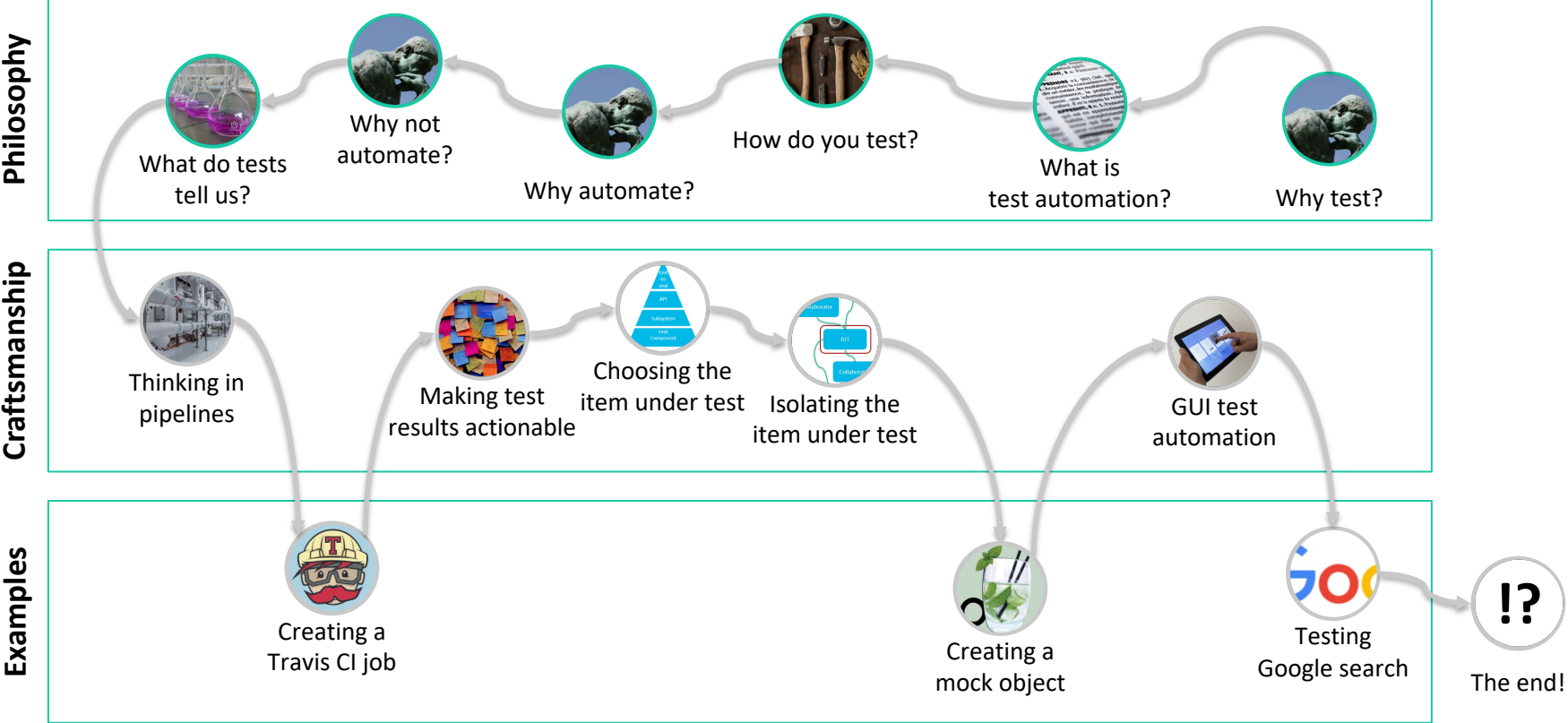
## THE QUESTION IS NOT WHETHER TO AUTOMATE OR NOT

The question is what to automate.

How can manual and automated testing techniques complement one another?



# Things to cover



# What do tests tell us?



## WHAT DOES A TEST CASE RESULT MEAN?

Does success mean the IUT is working?

Does failure mean it's not working?

Automated test case execution is a great way of protecting against predictable failure modes.

## NON-TEST CASE BASED APPROACHES

Automated fault injection.

Monitoring system parameters to detect anomalies or measuring quality characteristics.

Applying machine learning to predict failure modes, then monitoring the leading indicators.

...

## TEST RESULTS AS DECISION BASIS

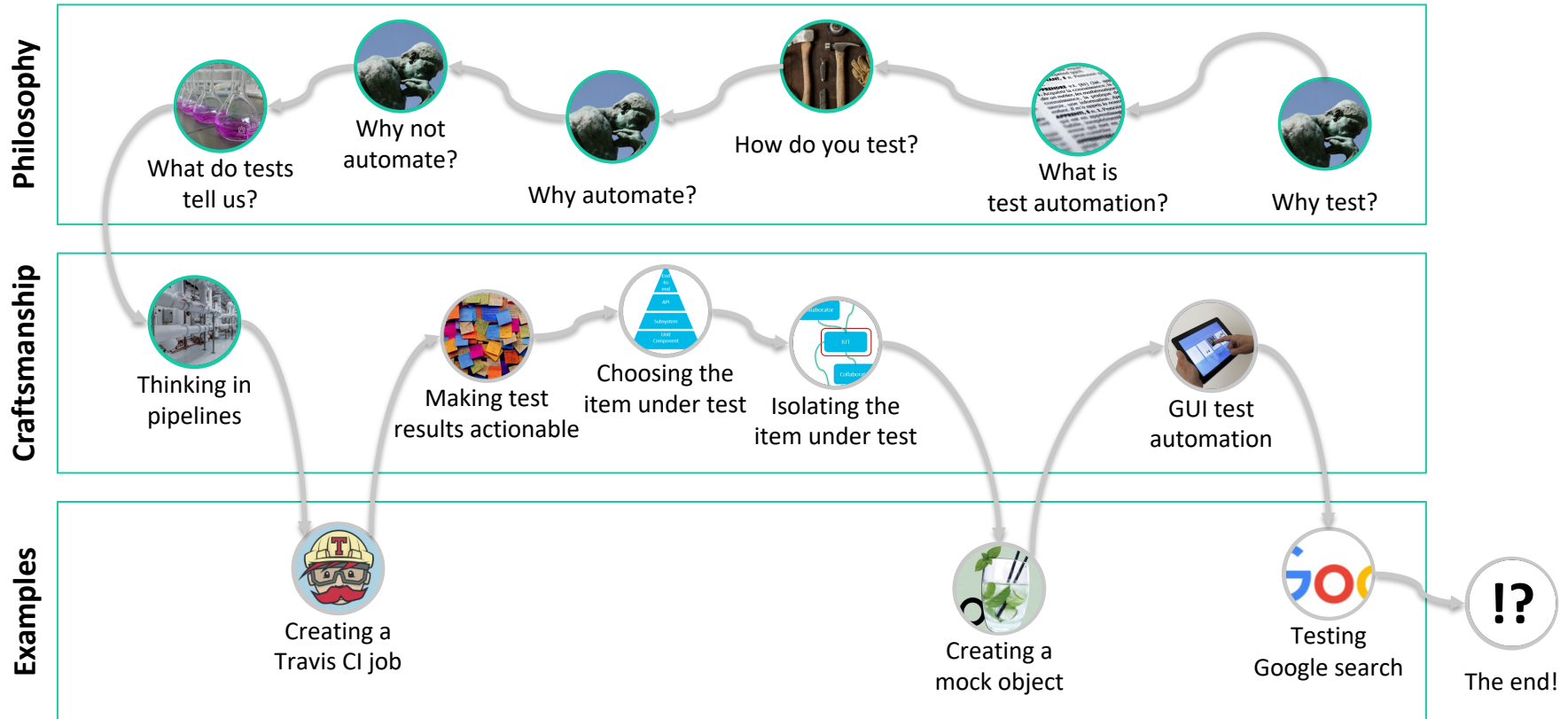
Test results (if useful) are always input to some decision (made by machine or human).

Be aware of what that decision is.

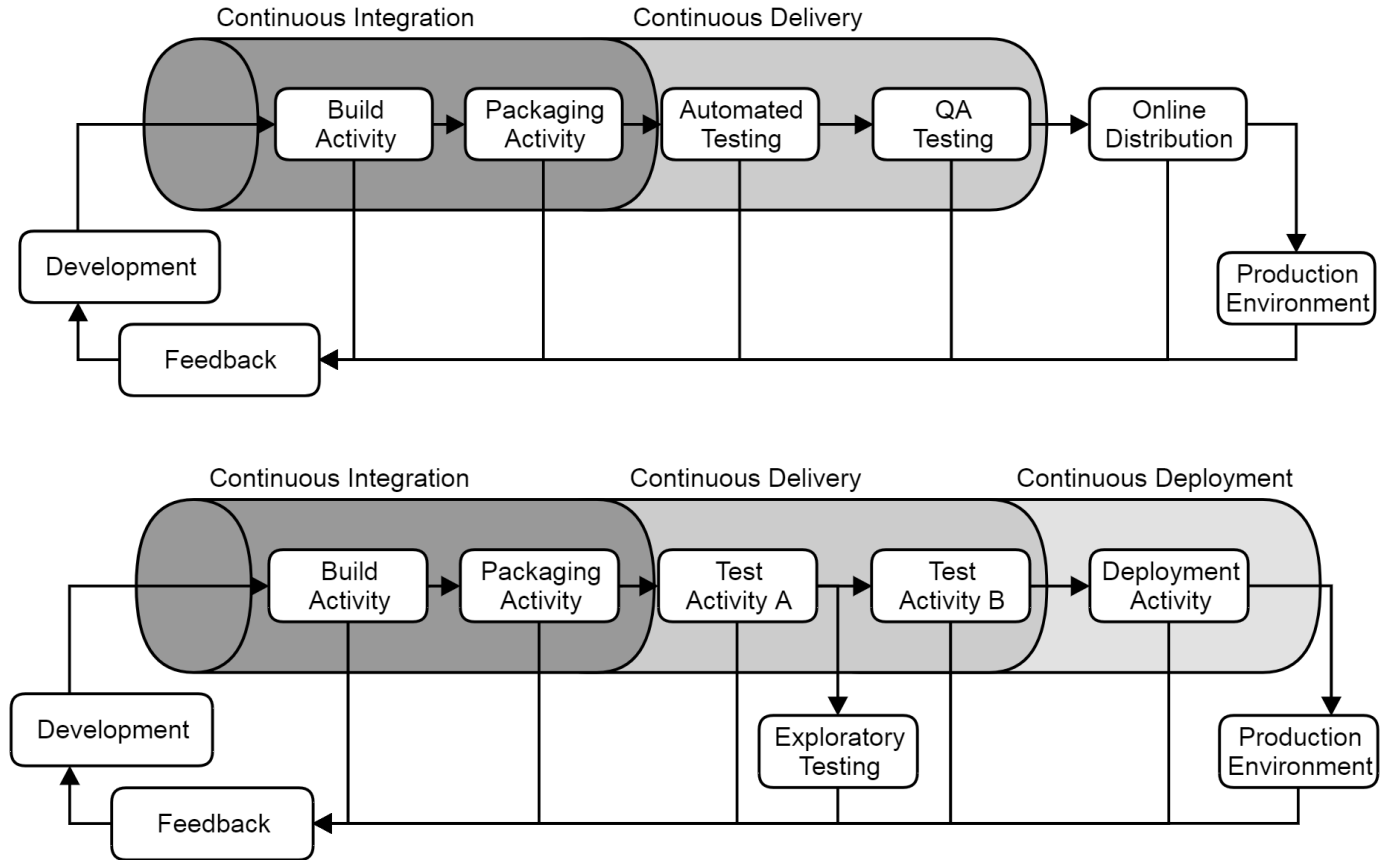
Consider whether the test results answer the questions you think they do.



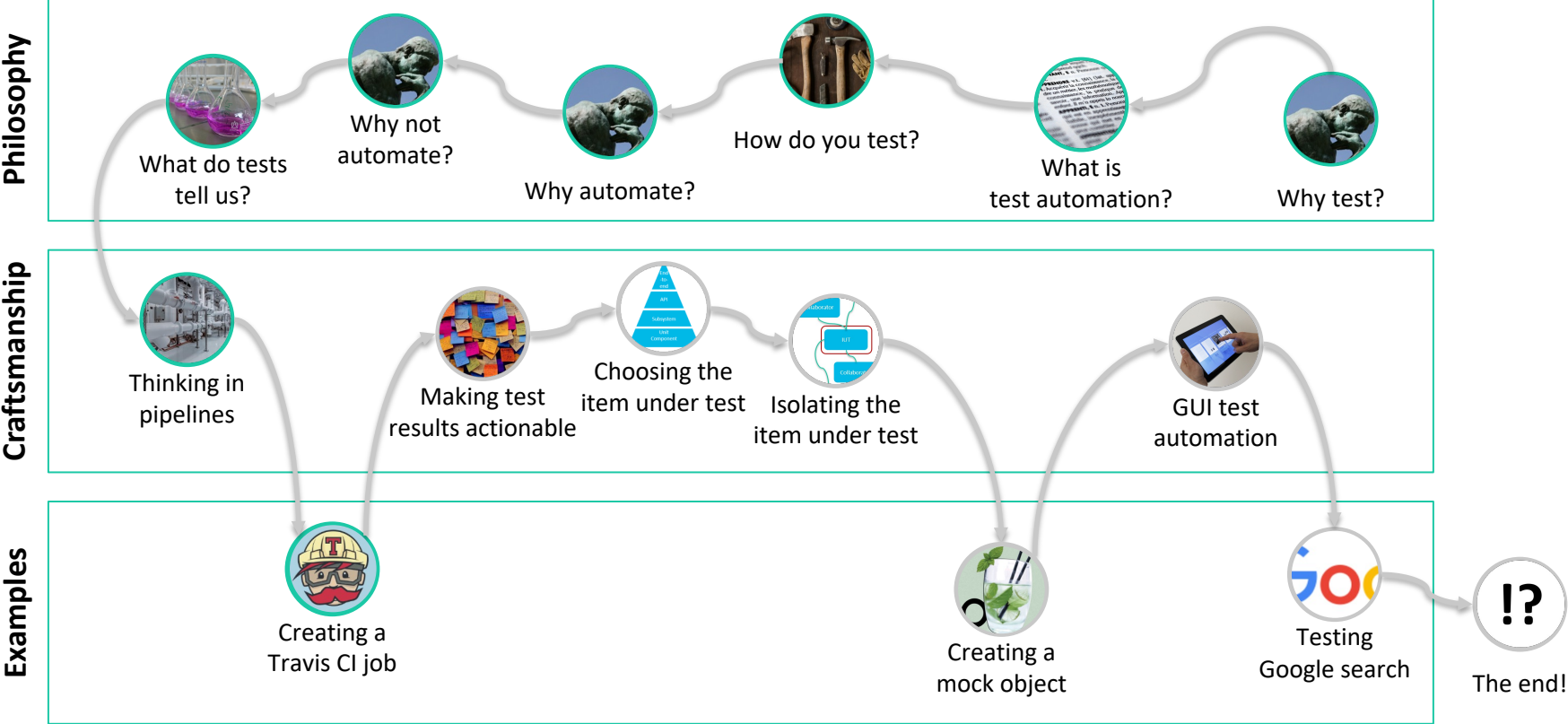
# Things to cover



# Connecting to continuous practices

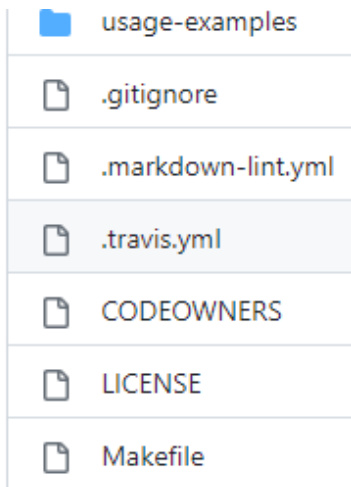


# Things to cover



# Creating a Travis CI job

## 1. GIVE TRAVIS ACCESS, CREATE A .TRAVIS.YML



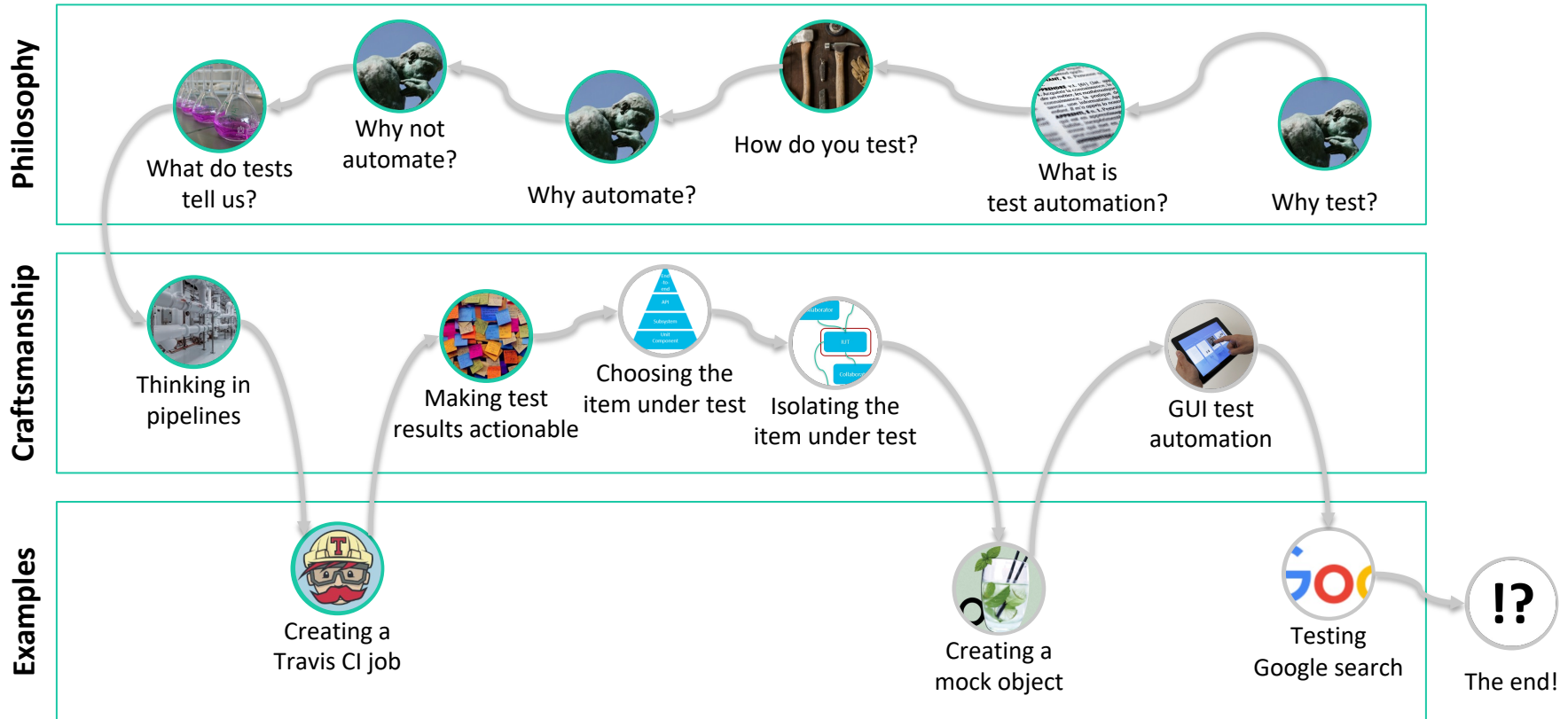
## 2. INSTRUCT IT TO EXECUTE SOMETHING, WHEN AND HOW

```
language: python
python:
- 3.5.1
script:
- python examples/validate.py
before_deploy:
- python find-latest-schemas.py schemas/ schemas-latest/
- tar -zcvf event-types-all.tar.gz schemas/*
- tar -zcvf event-types-latest.tar.gz schemas-latest/*
deploy:
```

## 3. WRITE A SCRIPT THAT EXITS WITH CODE 0 IF SUCCESSFUL

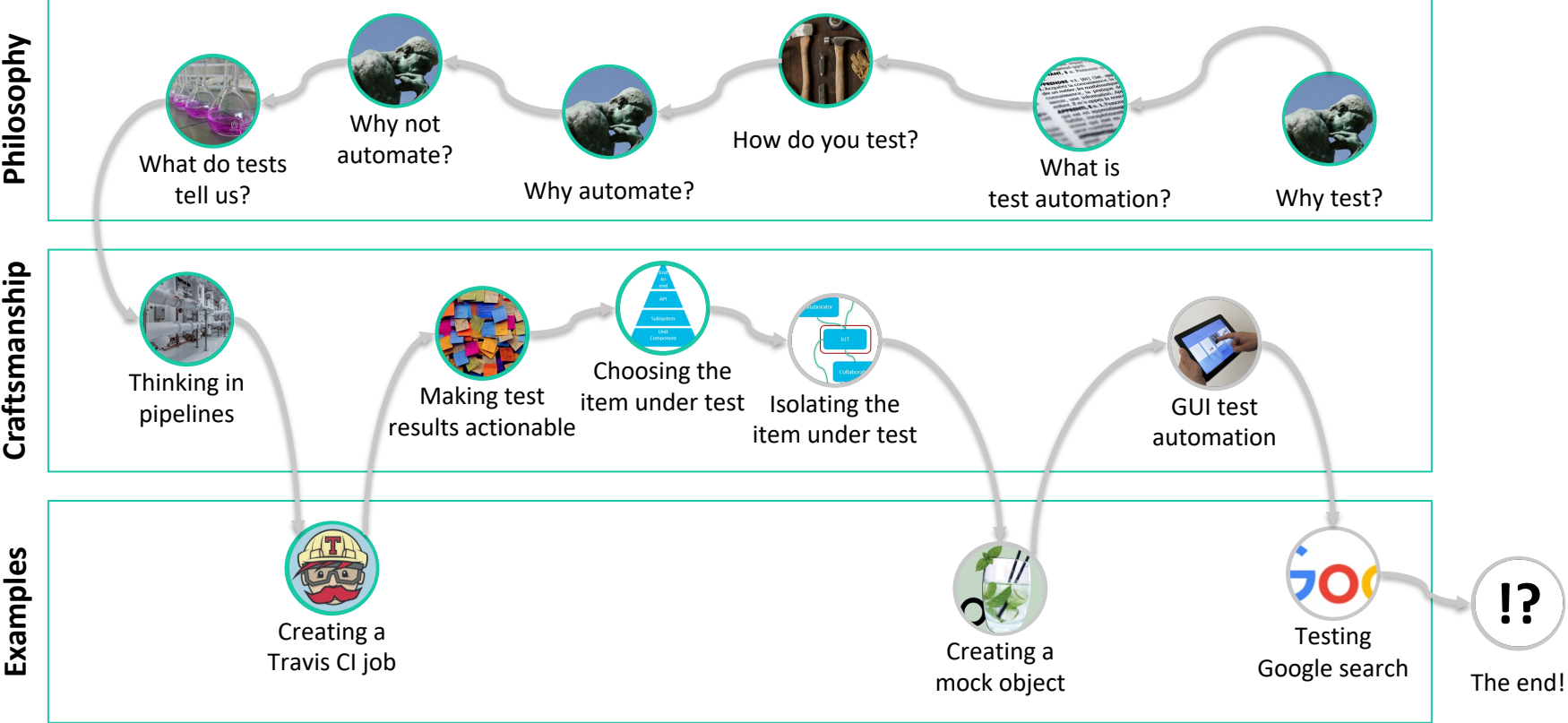
```
193  def main(maxExamples, includeArchives, shuffle):
194      if includeArchives:
195          extractionPaths = extractArchives()
196
197          schemas, badSchemaFiles = loadSchemas()
198          print("Loaded", len(schemas), "schemas.", flush=True)
199
200          examples, badExampleFiles = loadExamples()
201          print("Loaded", len(examples), "examples.", flush=True)
202
203          failures, unchecked, numberOfSuccessfulValidations = validateExamples(
204              examples, schemas, maxExamples, shuffle
205          )
```

# Things to cover

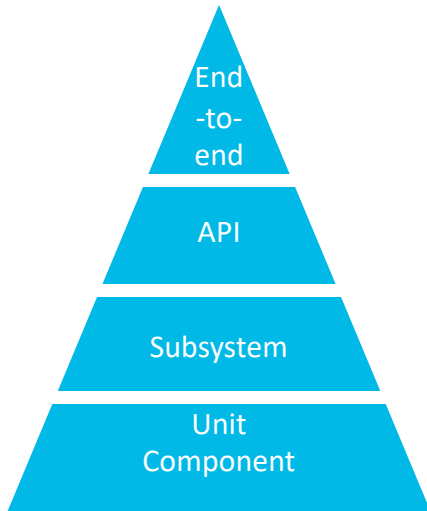




# Things to cover



# Choosing the item under test



## WHERE DO YOU TEST THE SYSTEM?

Testing requires resources. Where we choose to execute influences the cost.

## HIGH LEVELS OF SYSTEM COMPLETENESS

Testing the complete system is "easy"...

... but slow...

... and expensive.

## LOW LEVELS OF SYSTEM COMPLETENESS

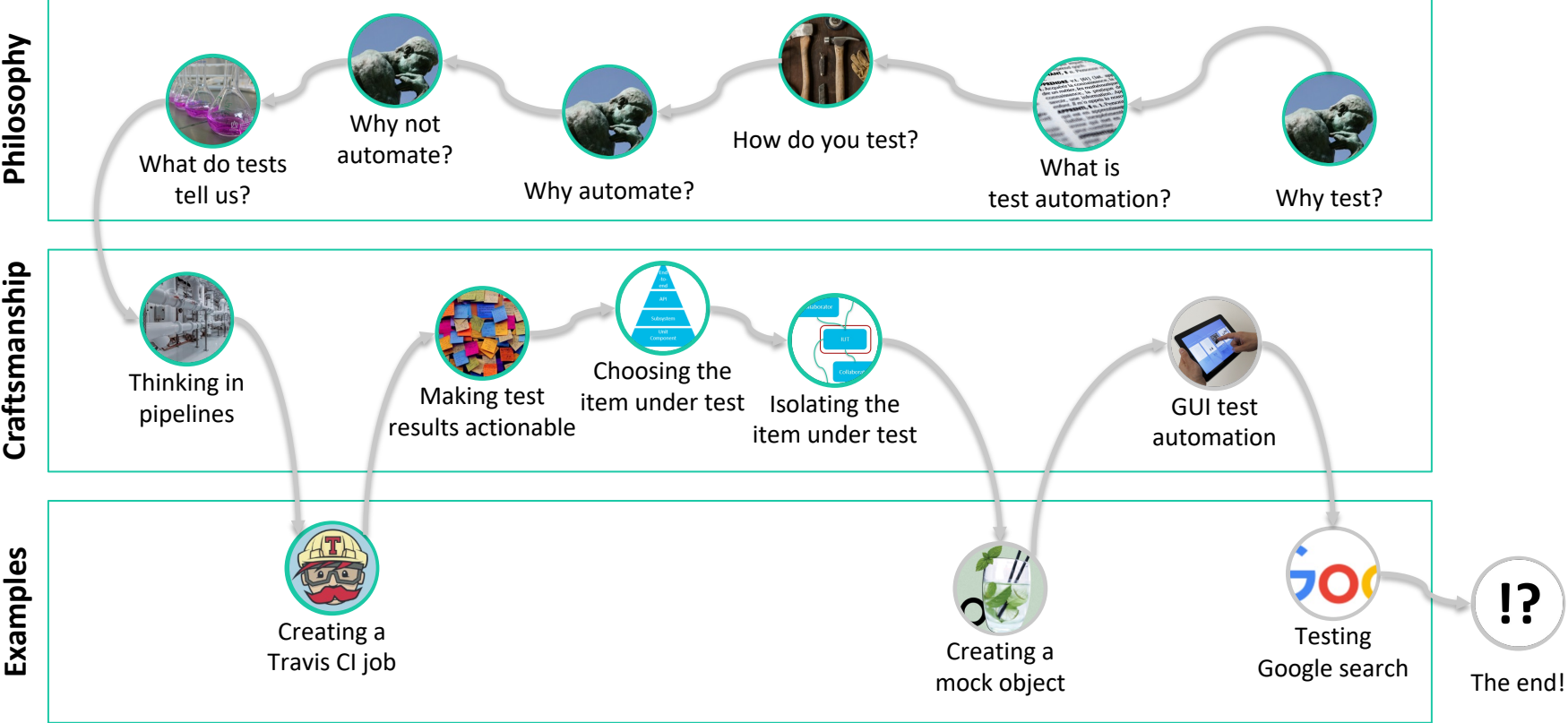
Testing at low levels of system completeness is fast and cheap...

... but requires you to think hard about what the result actually means...

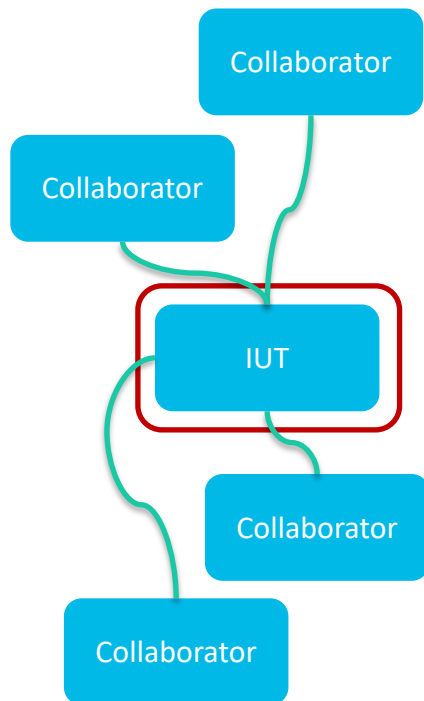
... and what kind of feedback you'll get out of it depends on the architecture.



# Things to cover



# Isolating the item under test



## WHY ISOLATE IT?

Speed and efficiency!

To reduce uncertainties.

Isolating the IUT from its collaborators lets you study precisely how the IUT behaves...

... at the cost of not knowing how it behaves together with those collaborators.

## TEST DOUBLES

Various techniques for replacing actual collaborators exist.

Nomenclature somewhat confused: fakes, stubs, spies, mocks...

Very helpful for slicing the system into testable parts, but...

... incurs overhead and maintenance costs.

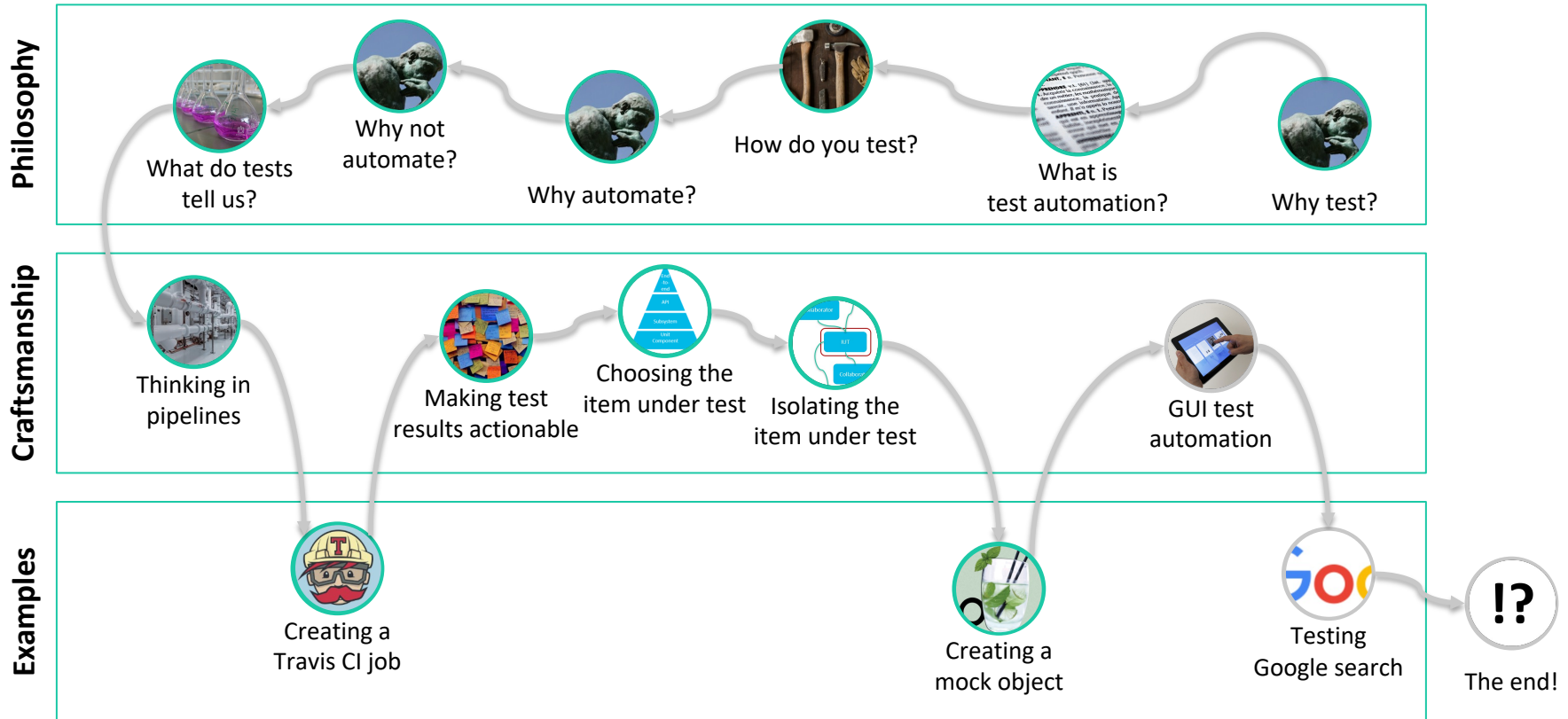
## LESS DUMB TEST DOUBLES

Opinions on what the difference is vary, if there is a difference, but...

Some test double techniques are more dynamic.

Some types of test doubles can inform on the IUT.

# Things to cover



# Creating a mock object

## STEP 1

We have an interface for the Item Under Test. It interacts with a missile, but we don't want to use an actual missile.

```
3  public interface IMissileController {
4      void load(final IMissile missile);
5      void issueLaunchOrder();
6      void setNumberOfLaunchOrdersRequired(final int numberOfLaunchOrders);
7  }
```

## STEP 2

In the test for the missile controller, we instruct our test framework (JUnit) to make preparations before each test. In these, we create a mock implementation of the missile interface, and instruct it to behave in a particular way.

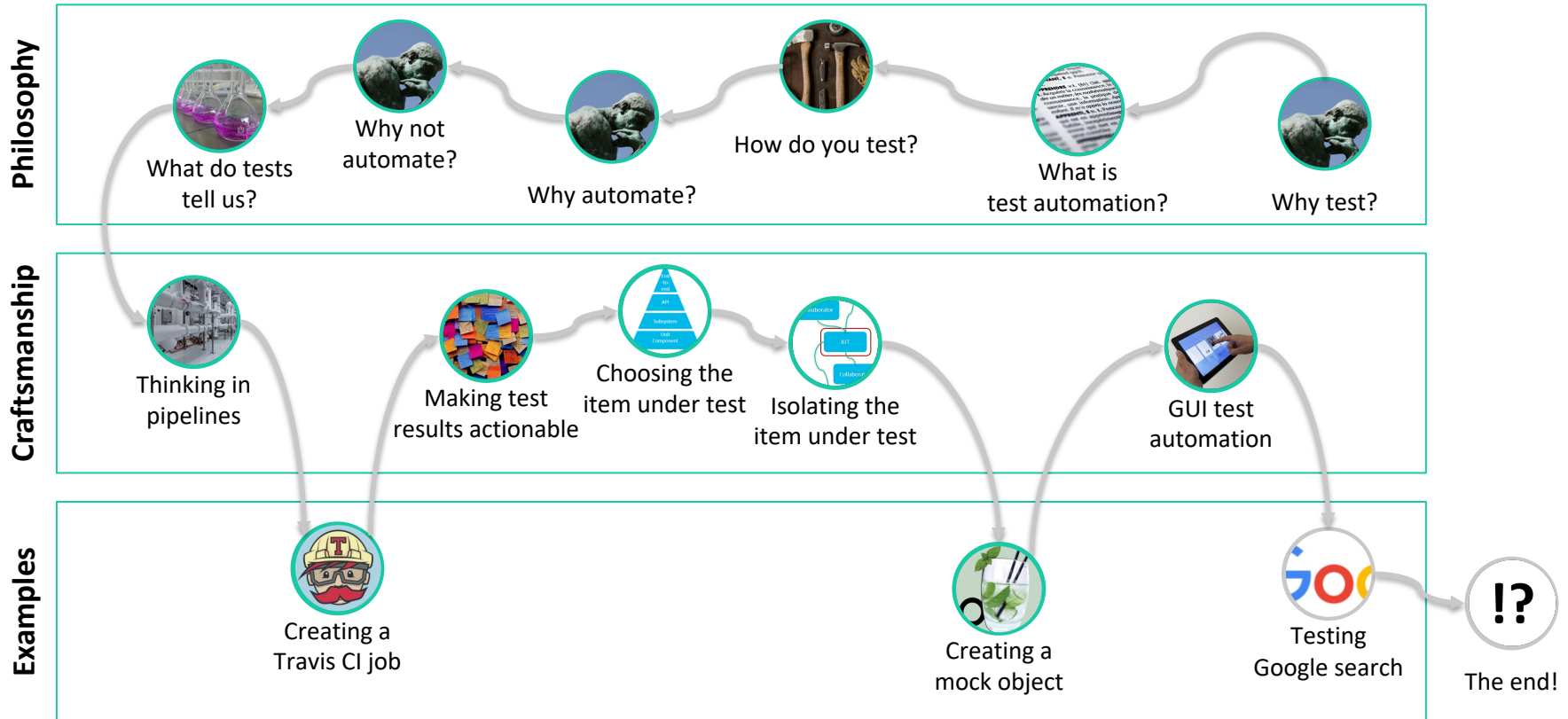
```
12  @Before
13  public void setUp() {
14      missile = mock(IMissile.class);
15      when(missile.getFuelLevel()).thenReturn(1.0f);
16      controller = new FancyMissileController();
17  }
```

## STEP 3

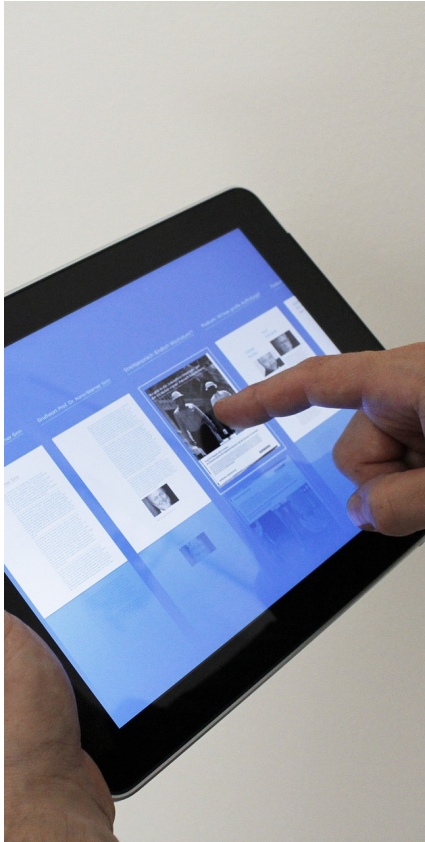
When we testing the controller, we load it up with our mock missile, tell it to launch and then interrogate the mock object to see what happened to it. We expect the controller to have attempted to launch it.

```
19  @Test
20  public void issuingLaunchOrderCausesMissileLaunch() throws Exception {
21      controller.load(missile);
22      controller.issueLaunchOrder();
23      verify(missile).launch();
24  }
```

# Things to cover



# GUI test automation



## WHAT ABOUT WHEN THE IUT ISN'T (EXPOSED) CODE?

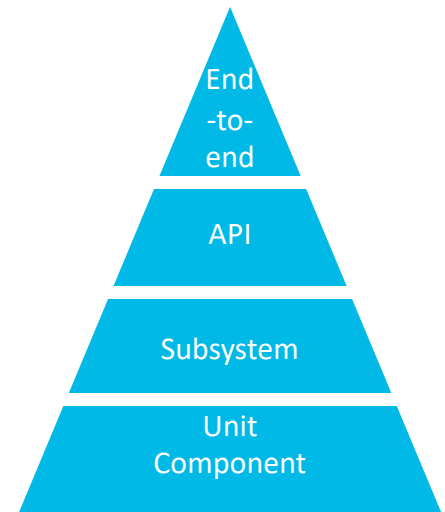
Prominent example is GUI, but analogous to all human-machine interfaces and physical interfaces.

## GUI TESTING PARADIGMS

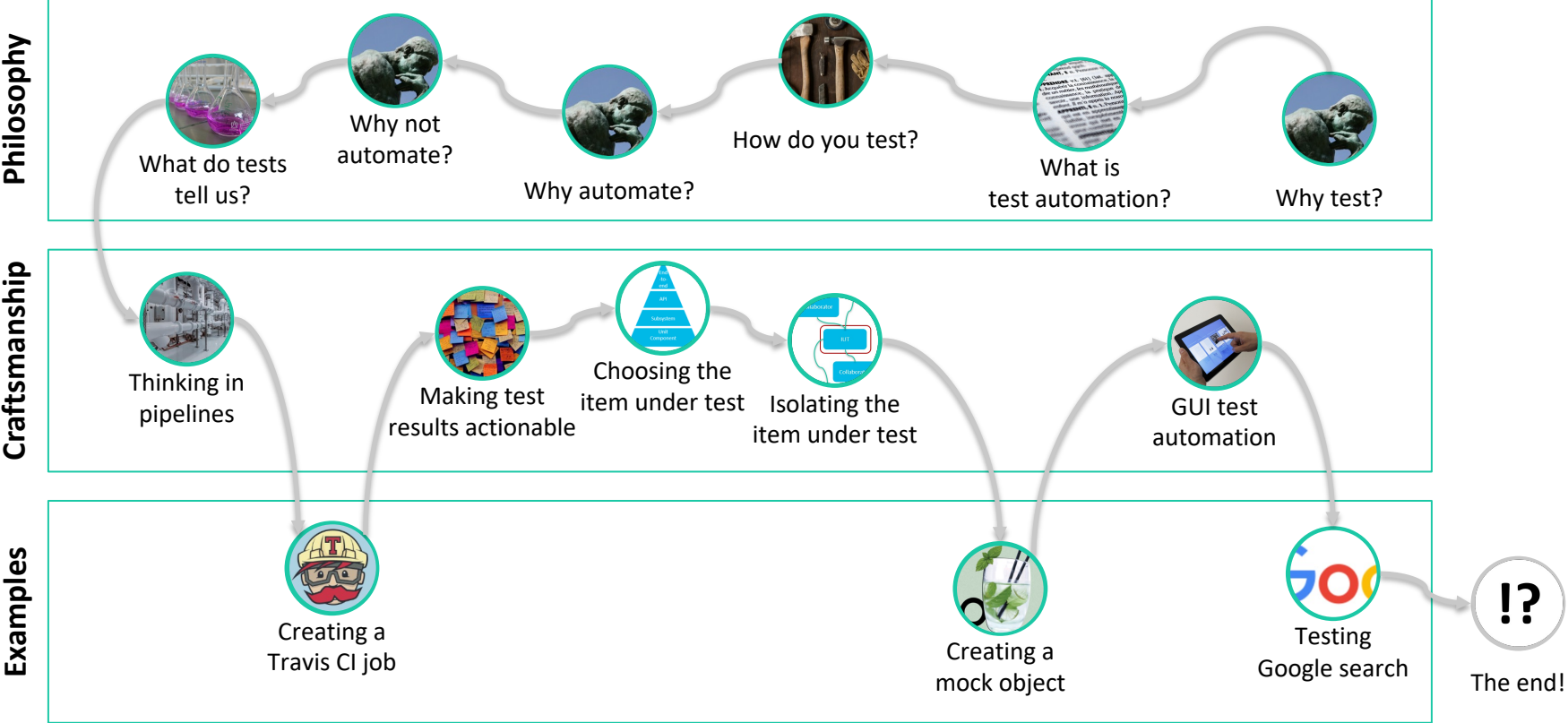
User behavior capture and playback.

Event capture.

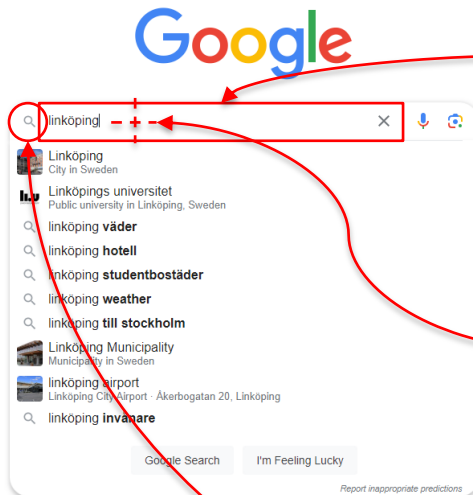
Image/text recognition.



# Things to cover



# Testing Google Search



## OPTION 1

Pick out the HTML elements directly, populate them and trigger them via code.

```
<textarea id="..." />
```

Advantage: Easy! You can do this with a simple Javascript.

Disadvantage: You have no idea what it actually *looks like*.


## OPTION 2

Use exact coordinates and simulate clicks and keystrokes.

Advantage: Simulating actual human interaction.

Disadvantage: If for any reason you change look and feel, all your tests will fail.

## OPTION 3

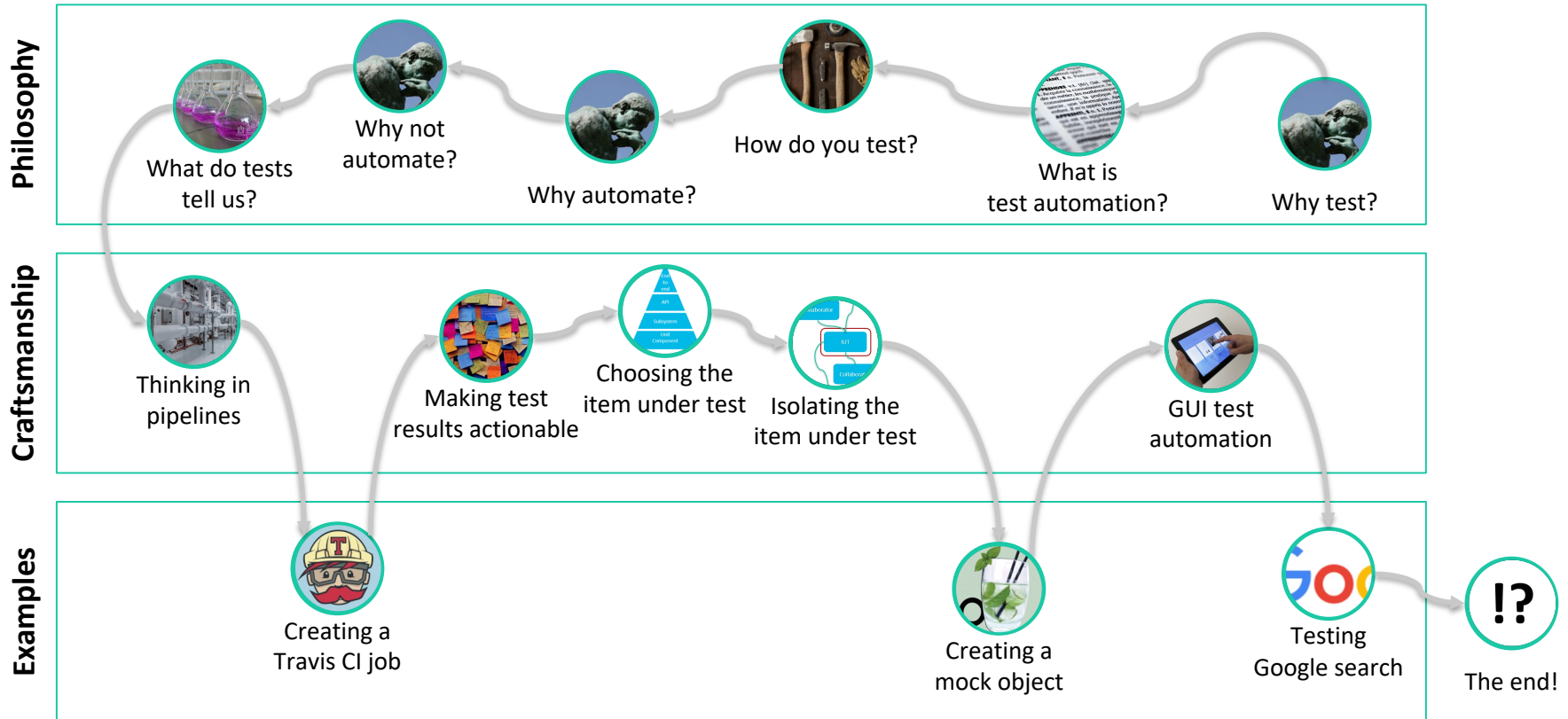
Use image recognition to find elements. Example: Click where you find this image: 

Advantage: Less fragile than Option 2.

Disadvantage: Requires specific tooling.

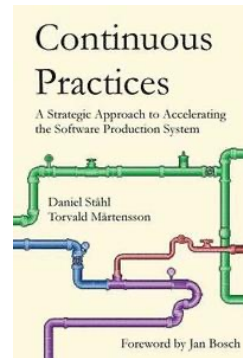
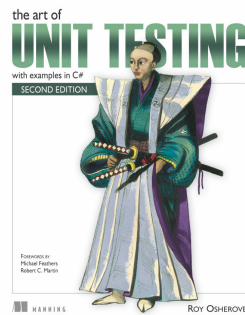
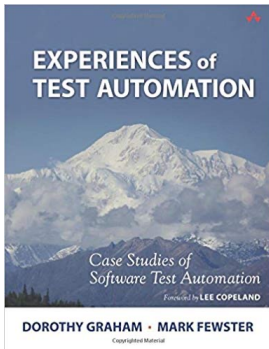


# Things to cover



# The end!

## FURTHER READING



!?

Daniel Ståhl

[www.liu.se](http://www.liu.se)