How deep are your tests?

Thomas Sundberg

Think Code AB
http://www.thinkcode.se
@thomassundberg
thomas@thinkcode.se

My opinions

If you don't agree, ask yourself why

and find me for a discussion

Why should we test?

Create confidence to release

Create value

Reduce waste

Get feedback from real users

Was this what they wanted?

Who are we testing for?

Not QA...

TDI

Ordinary persons

Quality is what the end user experiences

HOW?

End-to-end

From the user interface

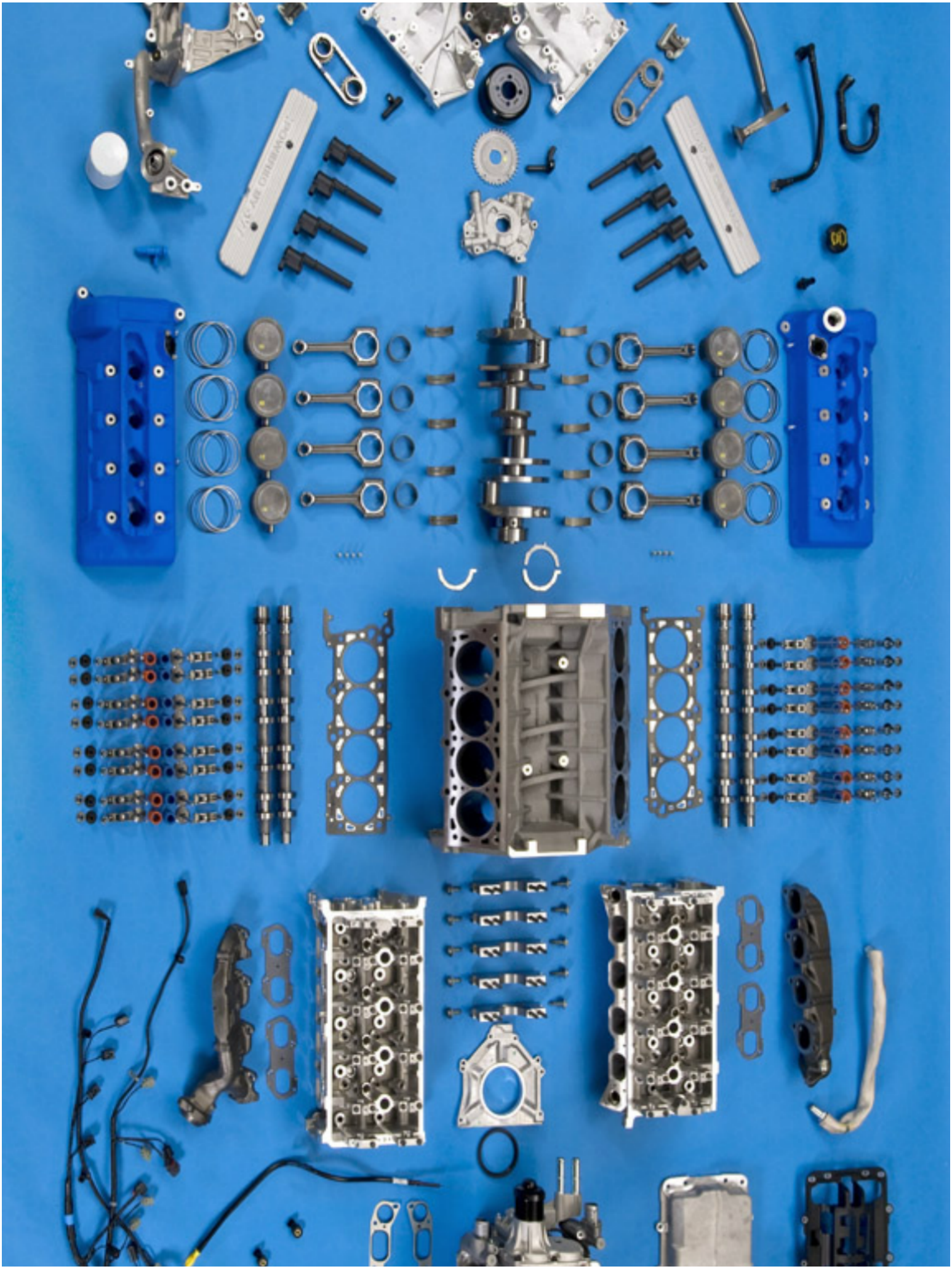
From an endpoint

Integrated

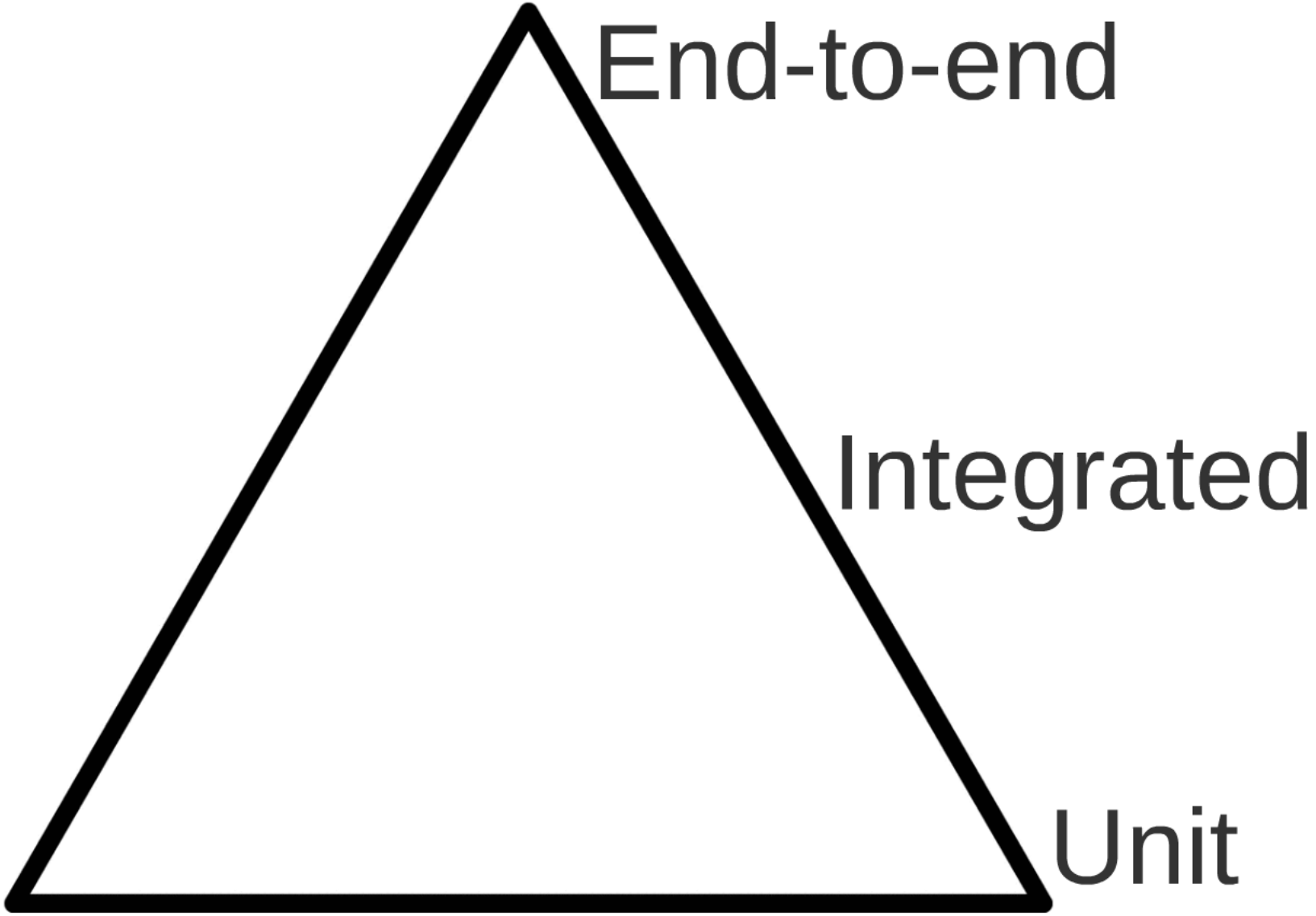
Interesting behavior with many objects
(interesting as in complicated)

Unit

Boring behavior with few objects



Testing pyramid



End-to-end

Typical properties

Slow

Done by someone else

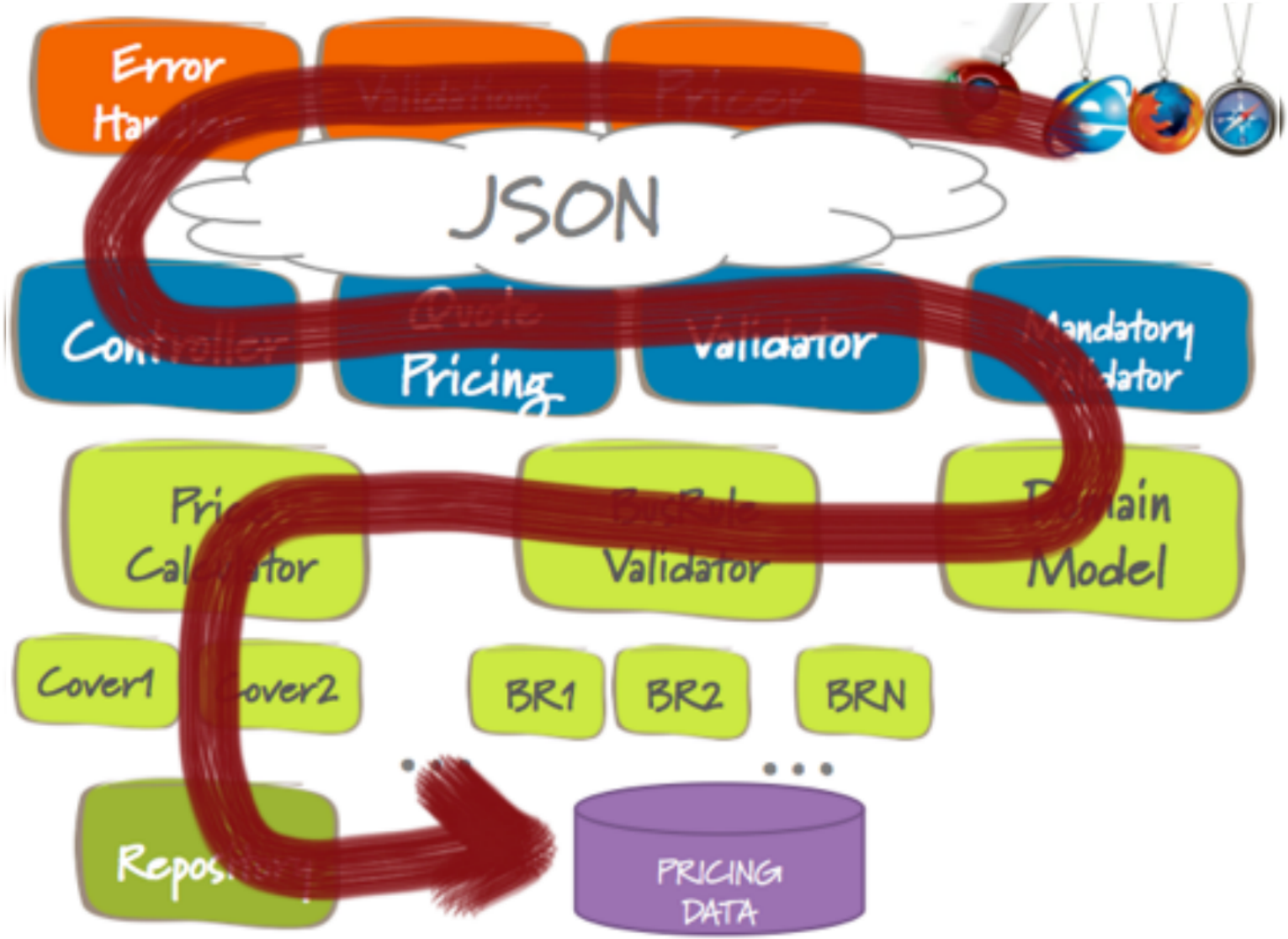
Risk based - acceptance

Verifies that everything is connected - wiring works

Many execution paths

Bad feedback - where did it break?

The wider the scope - the more fragile tests



http://fabiopereira.me/blog/2012/03/18/introducing-depth-of-test-dot/

Focus

Depth of field

Depth of field is the amount o
the nearest and farthest obje
acceptably sharp focus in a p

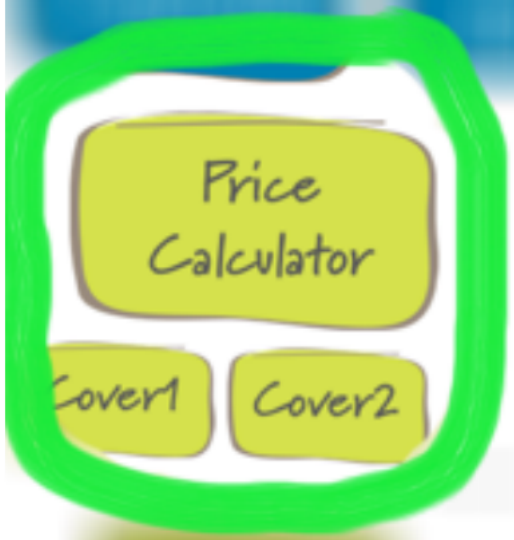
Depth of field

Depth of field is the amount of distance between the nearest and farthest objects that appear in acceptably sharp focus in a photograph.

It is very subjective what you think is ok.


8

Shallow Depth of Test



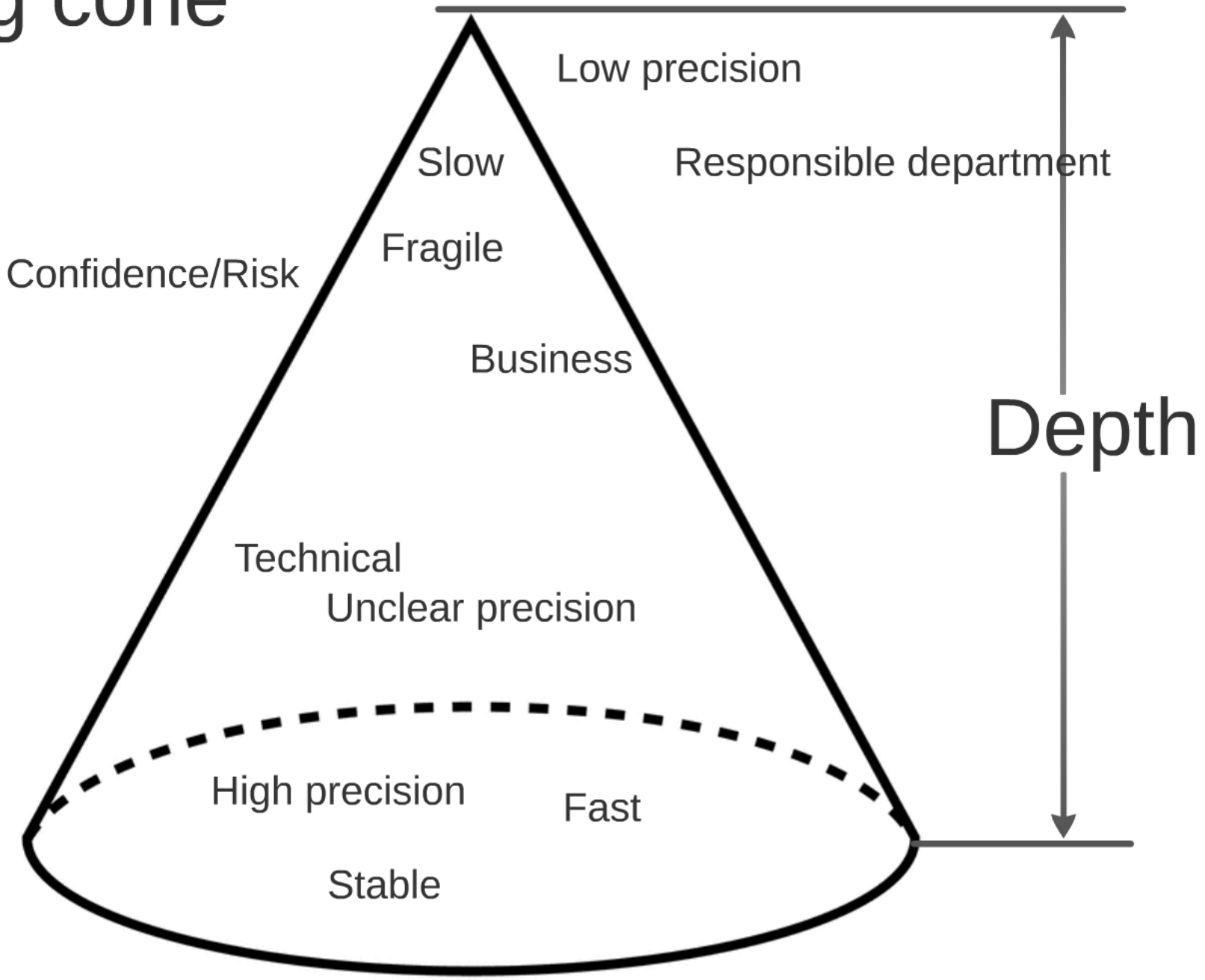
http://fabiopereira.me/blog/2012/03/18/introducing-depth-of-test-dot/

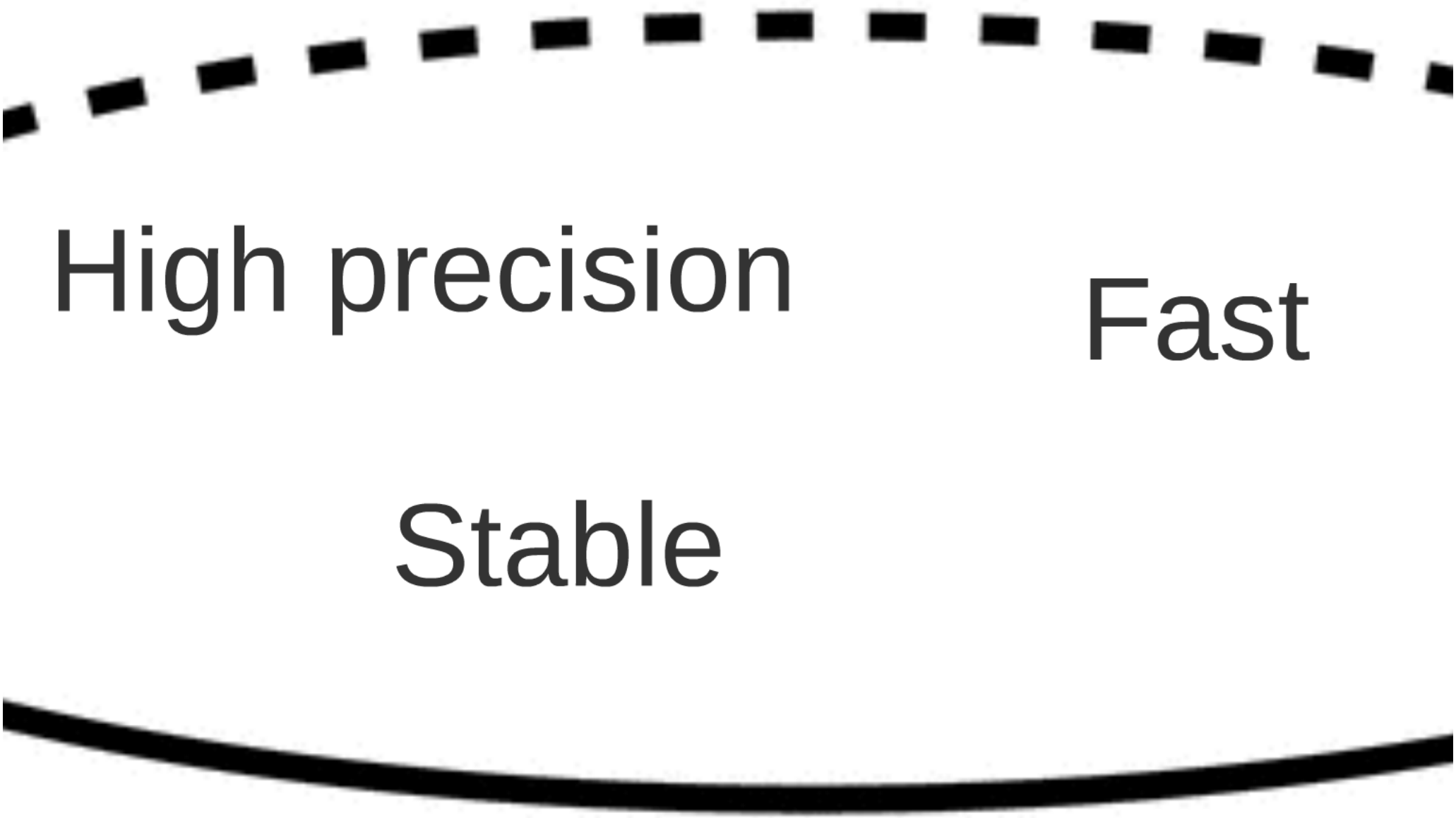
Price Calculator

Cover1

Cover2

Testing cone




High precision
Fast
Stable

Integration or integrated?

Integrated as in large chunks of functionality at the same time.

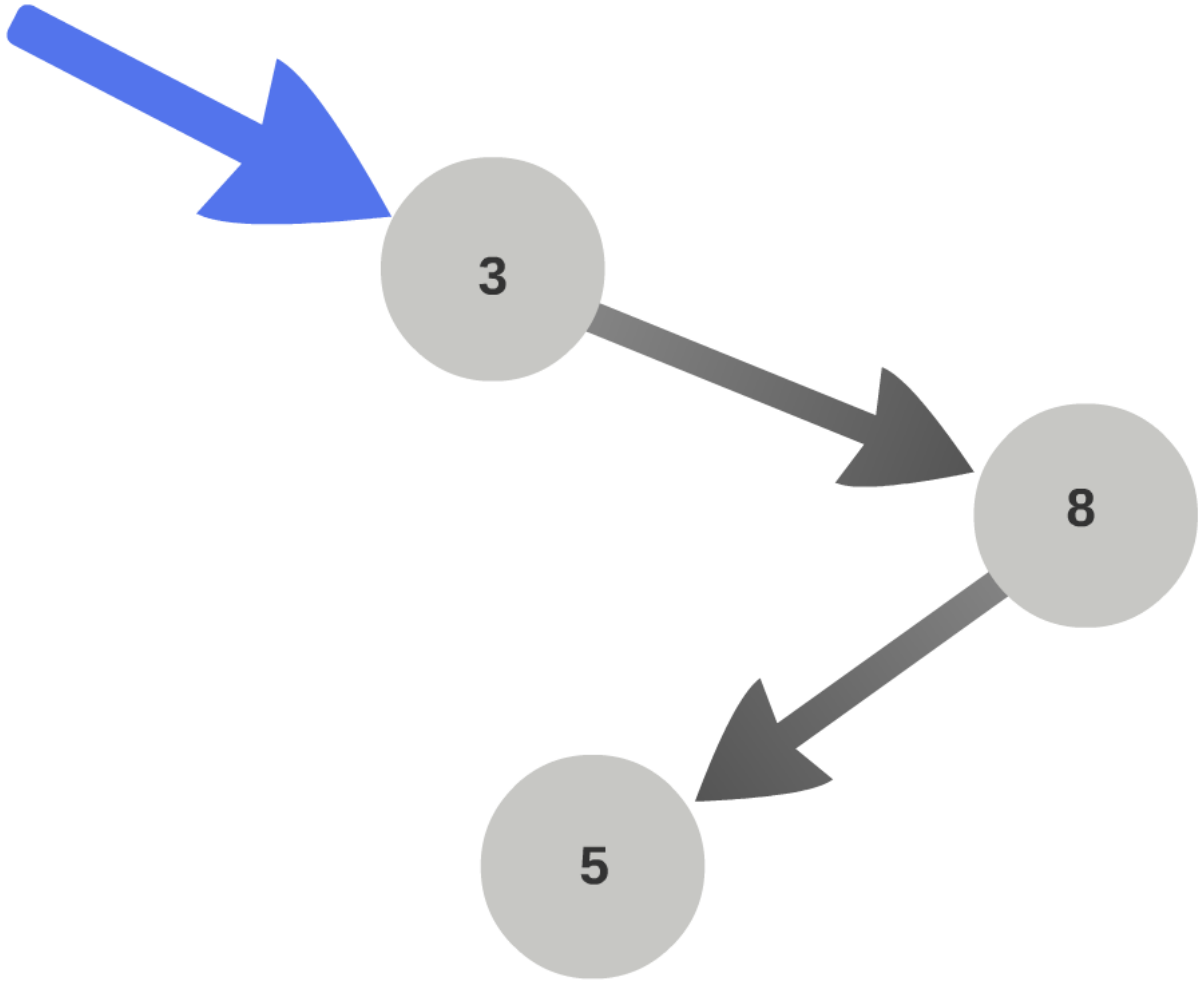
NOT integration as in integration with external services.

Probably out of your control

Integration with external services are needed, but thats (kind of) another topic.

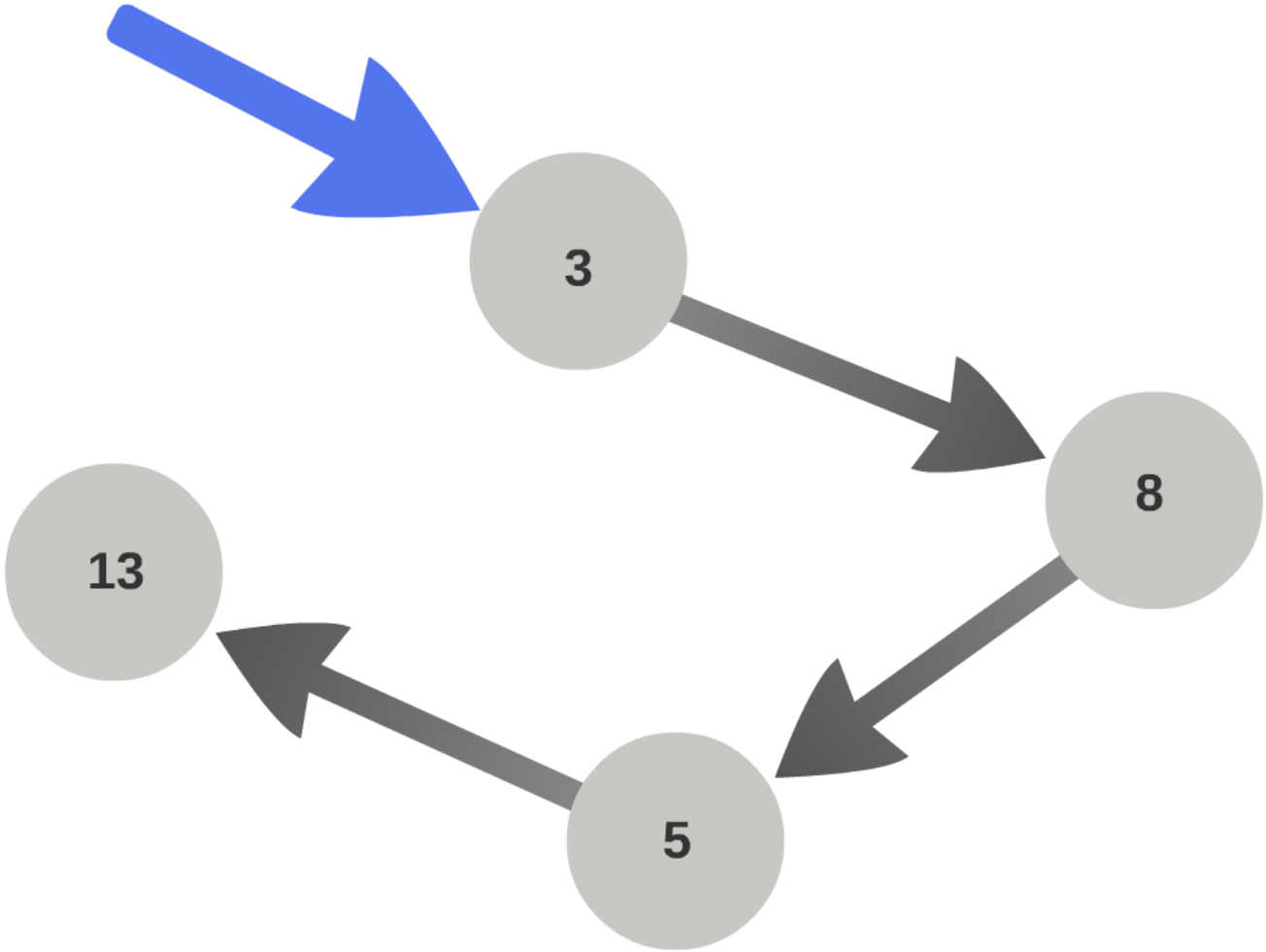
Integrated tests

Executions paths



3 x 8 x 5 = 120

Executions paths

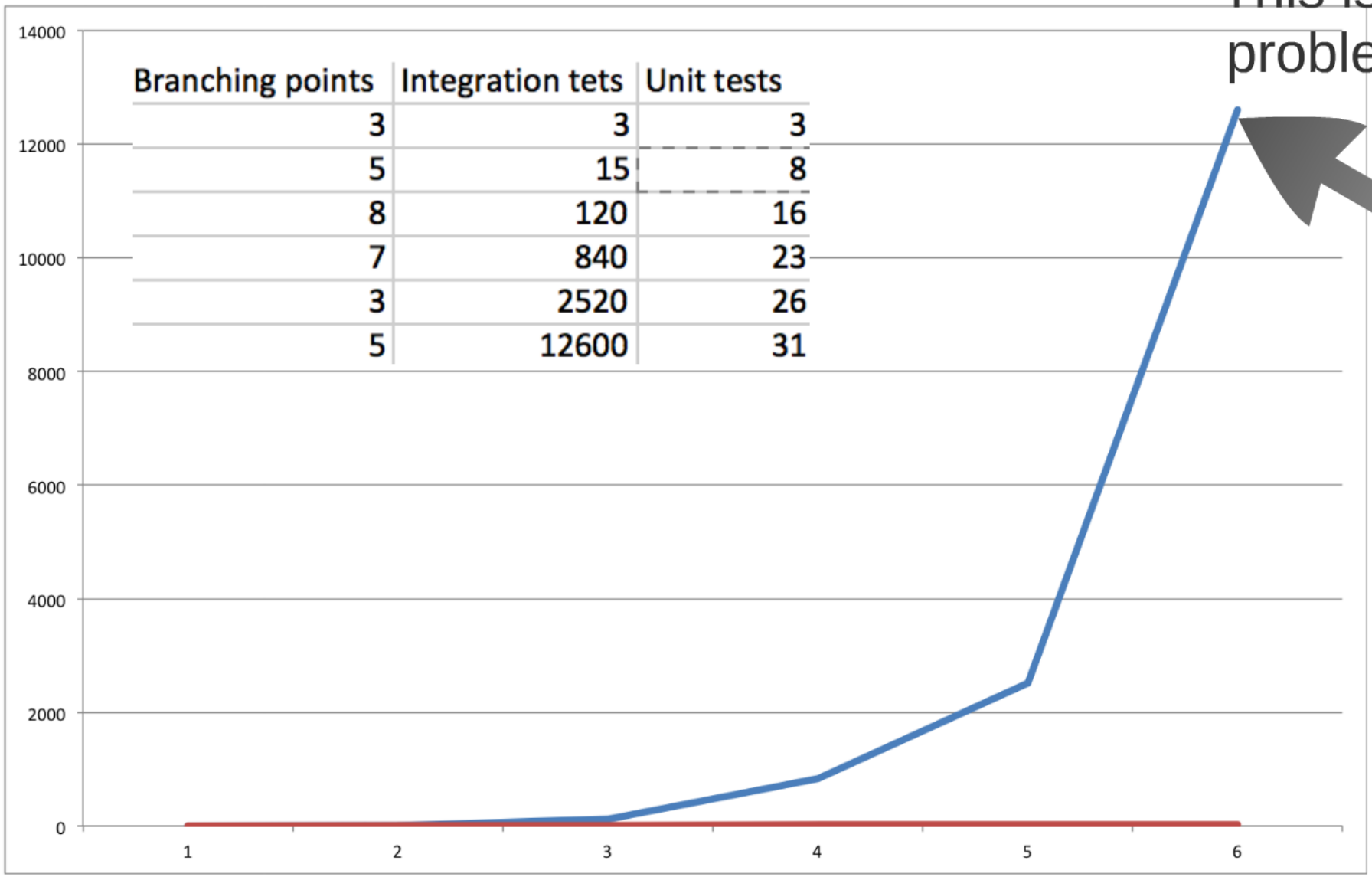


3 x 8 x 5 x 13 = 1560

Executions paths - what is the problem?

This is the problem...

Branching points	Integration tets	Unit tests
3	3	3
5	15	8
8	120	16
7	840	23
3	2520	26
5	12600	31



Ockhams razor

Among competing hypotheses, the one with the fewest assumptions should be selected.

William of Ockham (c. 1287–1347)

Interpretation

The fewest assumptions → less risk

Unit tests

Unit test definition

A well know

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

Roy Osherove

Test a small part

One, and only one, reason to f

Unit test definition

A well known mine field...

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

Roy Osherove

Test a small part

One, and only one, reason to fail

Can be executed in parallel

Fast - sub seconds

A unit test does not

have more than one reason to fail

offer slow feedback

require exclusive execution

use the network

use the file system

talk to the database

Michael Feathers

Conversation

Consumer

Can I ask particular questions?

Do I get the expected answers back?

Provider

Am I able to understand her?

Am I able to answer her questions?

Verify interaction with mocks (expectations)	Define expected methods
Stubs returns expected values (answers)	Implement the methods correct

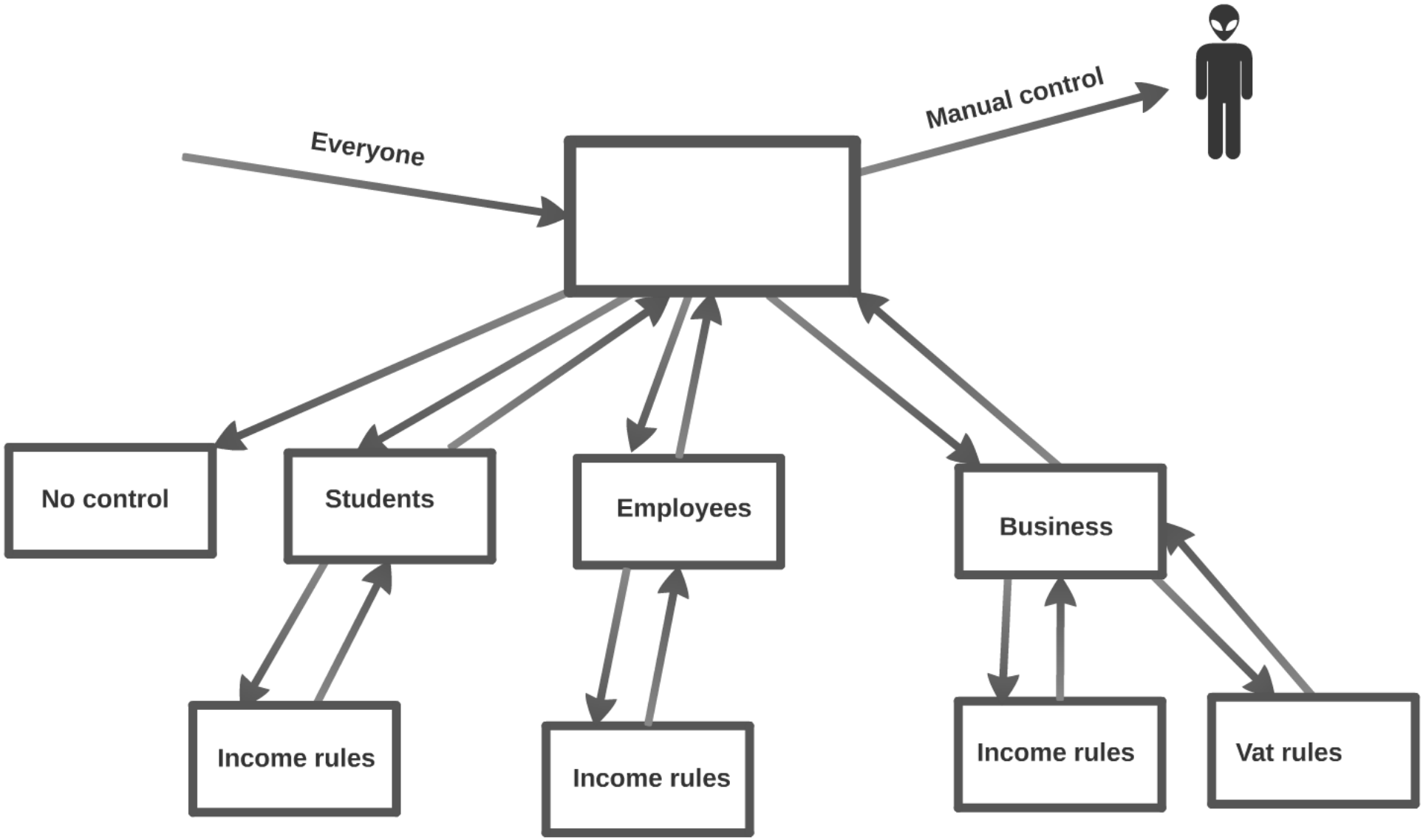
Interaction → Test action

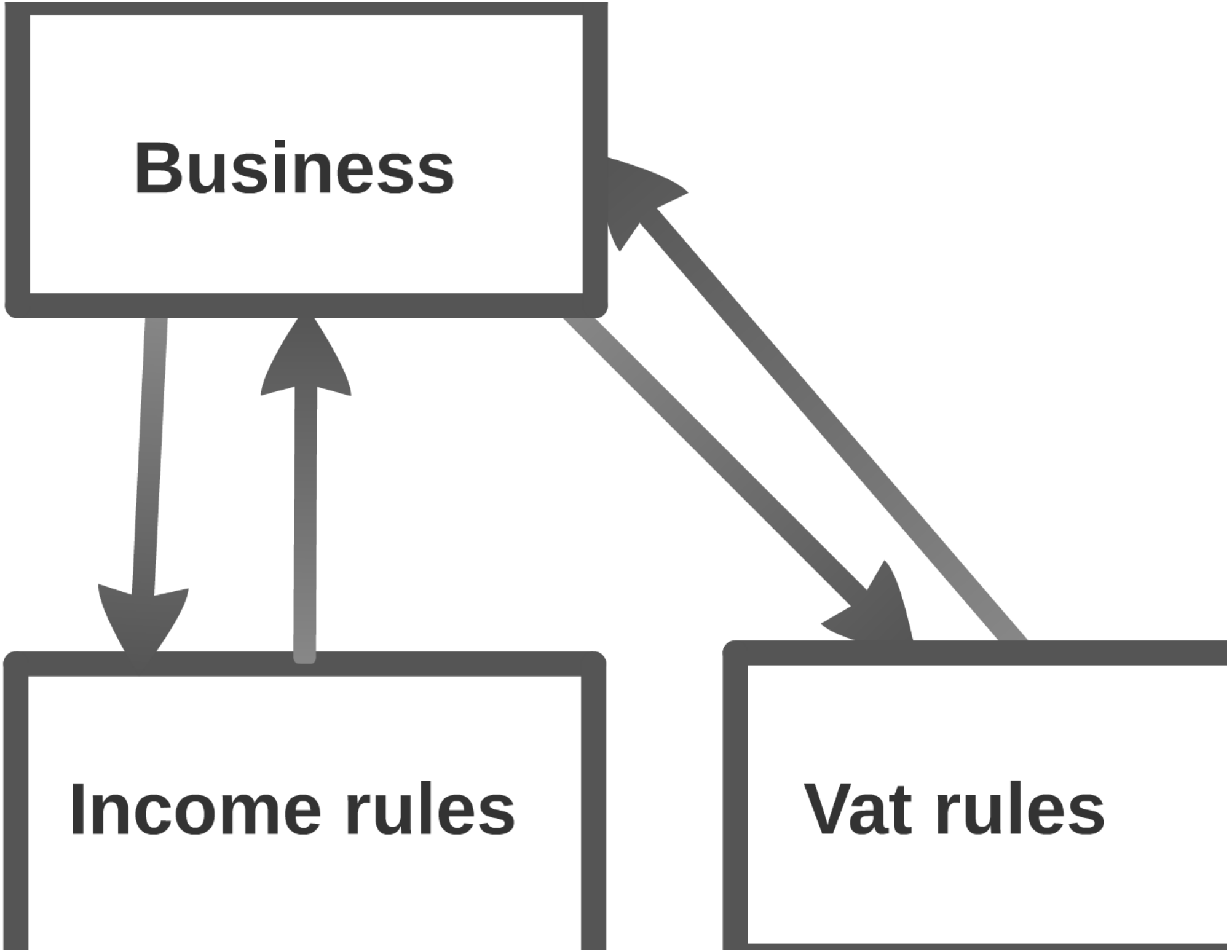
Answer → Assert

That was abstract...

An example would be nice

Checking taxes




Business
Income rules
Vat rules

Verify interaction with mocks (expectations)

```
@Test
public void check_that_the_vat_rule_engine_is_used() {
    VatRules vatRules = mock(VatRules.class);
    Business business = new Business("any string", vatRules);

    business.getVatDueDate();

    verify(vatRules).getVatDueDate(anyString());
}
```

Forces me to implement the method, getVatDueDate(String orgNumber)

Stubs returns expected values (answers)

```
@Test
public void check_due_date_for_vat_when_you_invoice_within_eu() {
    LocalDate expected = LocalDate.parse("2016-02-28");

    VatRules vatRules = mock(VatRules.class);
    when(vatRules.getVatDueDate("5569215576")).thenReturn(LocalDate.parse("2016-02-28"));
    Business business = new Business("5569215576", vatRules);

    LocalDate actual = business.getVatDueDate();

    assertThat(actual, is(expected));
}
```

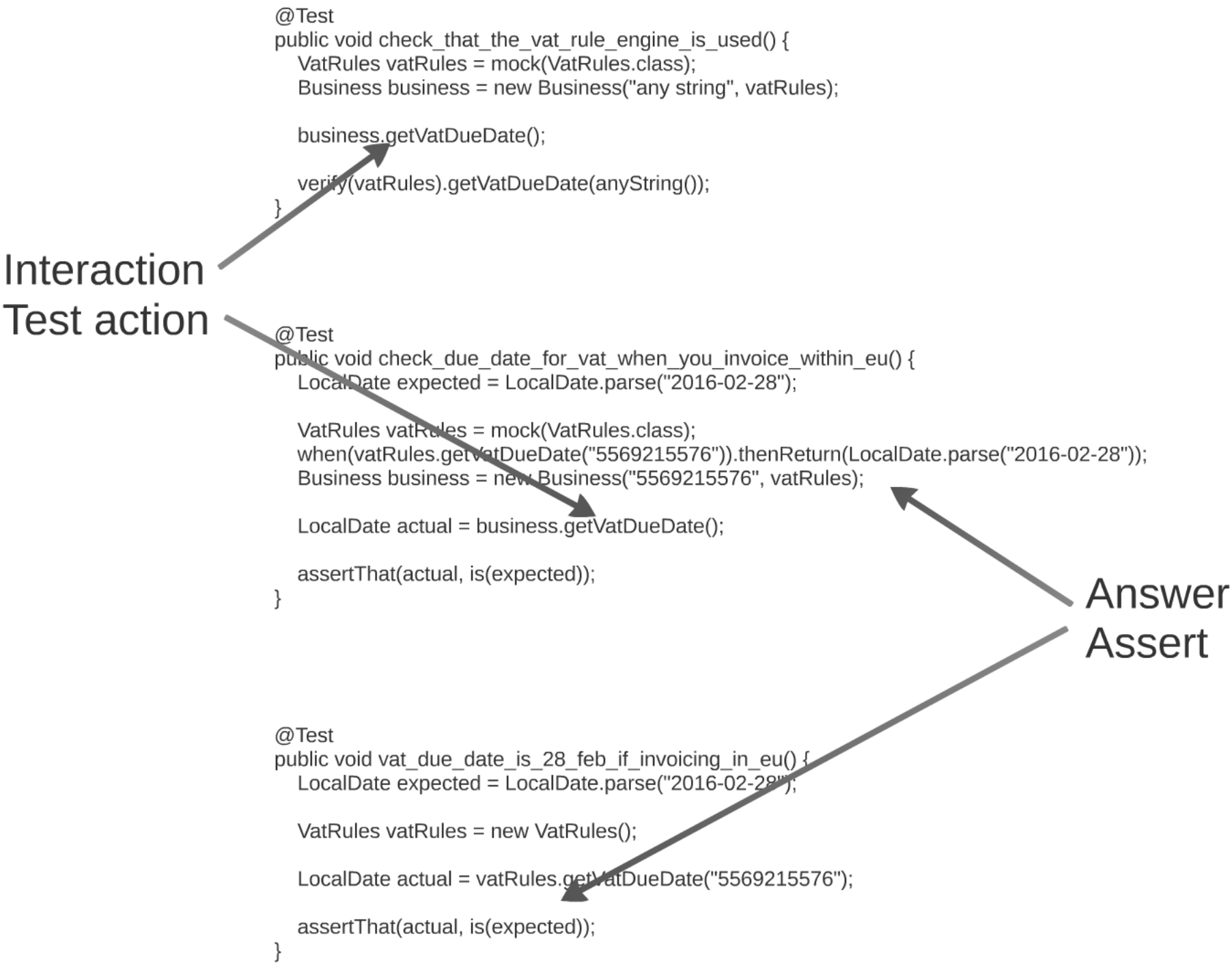
Implement the methods correct

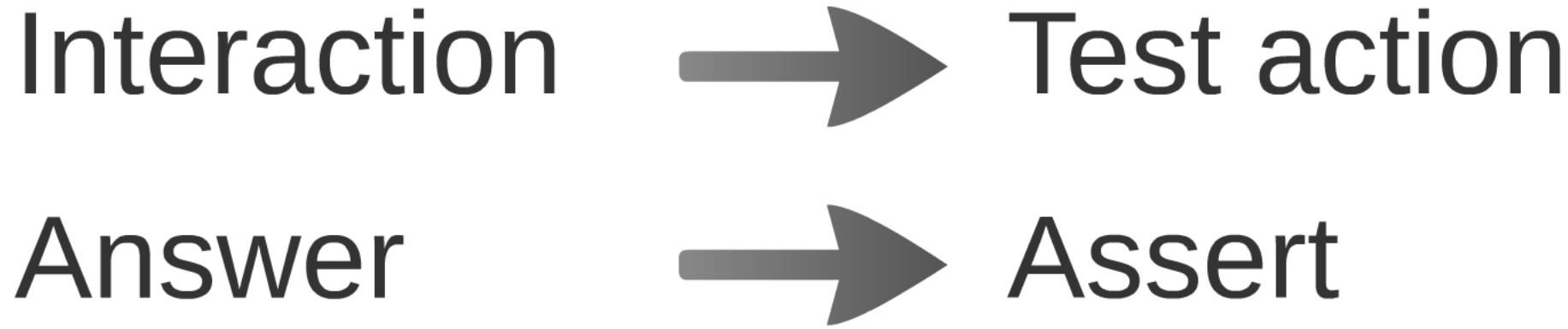
```
@Test
public void vat_due_date_is_28_feb_if_invoicing_in_eu() {
    LocalDate expected = LocalDate.parse("2016-02-28");

    VatRules vatRules = new VatRules();

    LocalDate actual = vatRules.getVatDueDate("5569215576");

    assertThat(actual, is(expected));
}
```

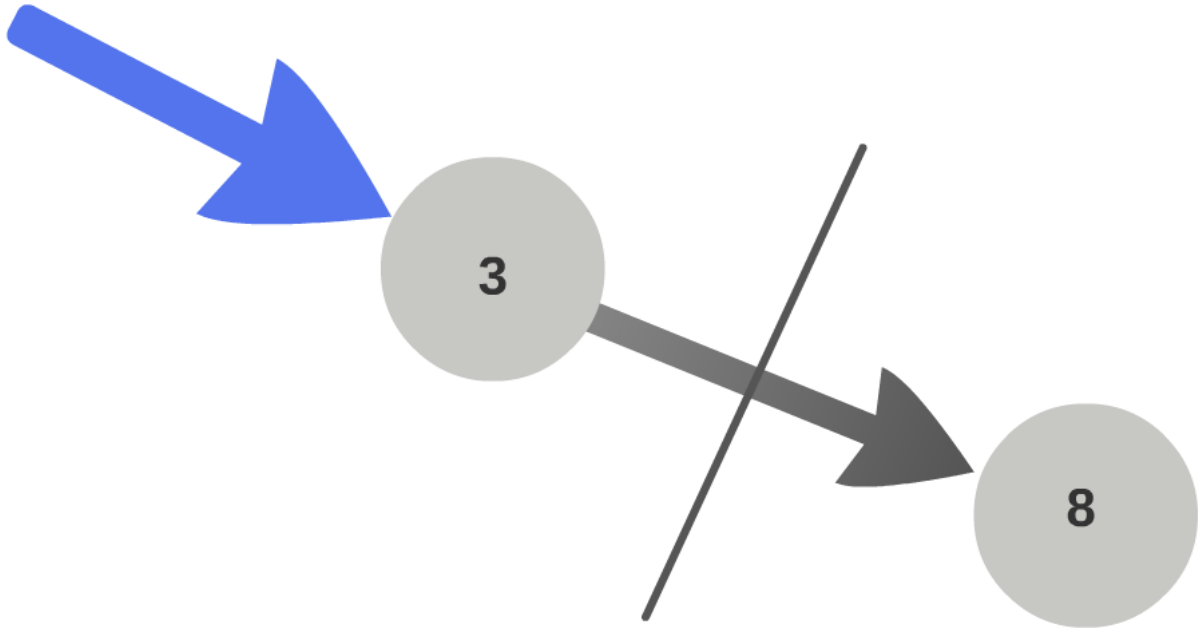

@Test
public void check_that_the_vat_rule_engine_is_used() {
VatRules vatRules = mock(VatRules.class);
Business business = new Business("any string", vatRules);
business.getVatDueDate();
verify(vatRules).getVatDueDate(anyString());
}
Interaction
Test action
@Test
public void check_due_date_for_vat_when_you_invoice_within_eu() {
LocalDate expected = LocalDate.parse("2016-02-28");
VatRules vatRules = mock(VatRules.class);
when(vatRules.getVatDueDate("5569215576")).thenReturn(LocalDate.parse("2016-02-28"));
Business business = new Business("5569215576", vatRules);
LocalDate actual = business.getVatDueDate();
assertThat(actual, is(expected));
}
Answer
Assert
@Test
public void vat_due_date_is_28_feb_if_invoicing_in_eu() {
LocalDate expected = LocalDate.parse("2016-02-28");
VatRules vatRules = new VatRules();
LocalDate actual = vatRules.getVatDueDate("5569215576");
assertThat(actual, is(expected));
}



Like book keeping,
every transaction has to balance.

(The sum should be 0)

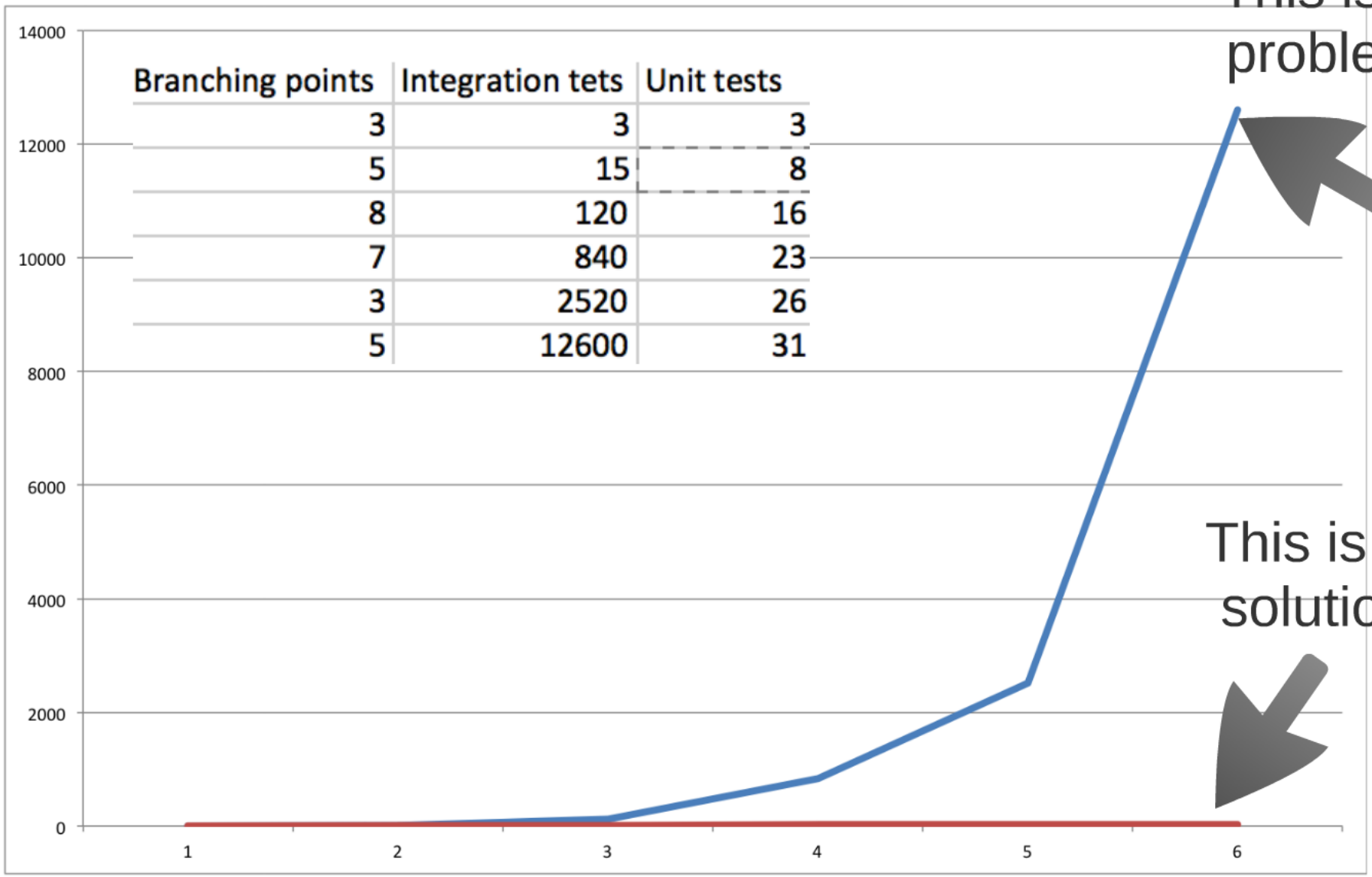
Executions paths



3 + 8 = 11+

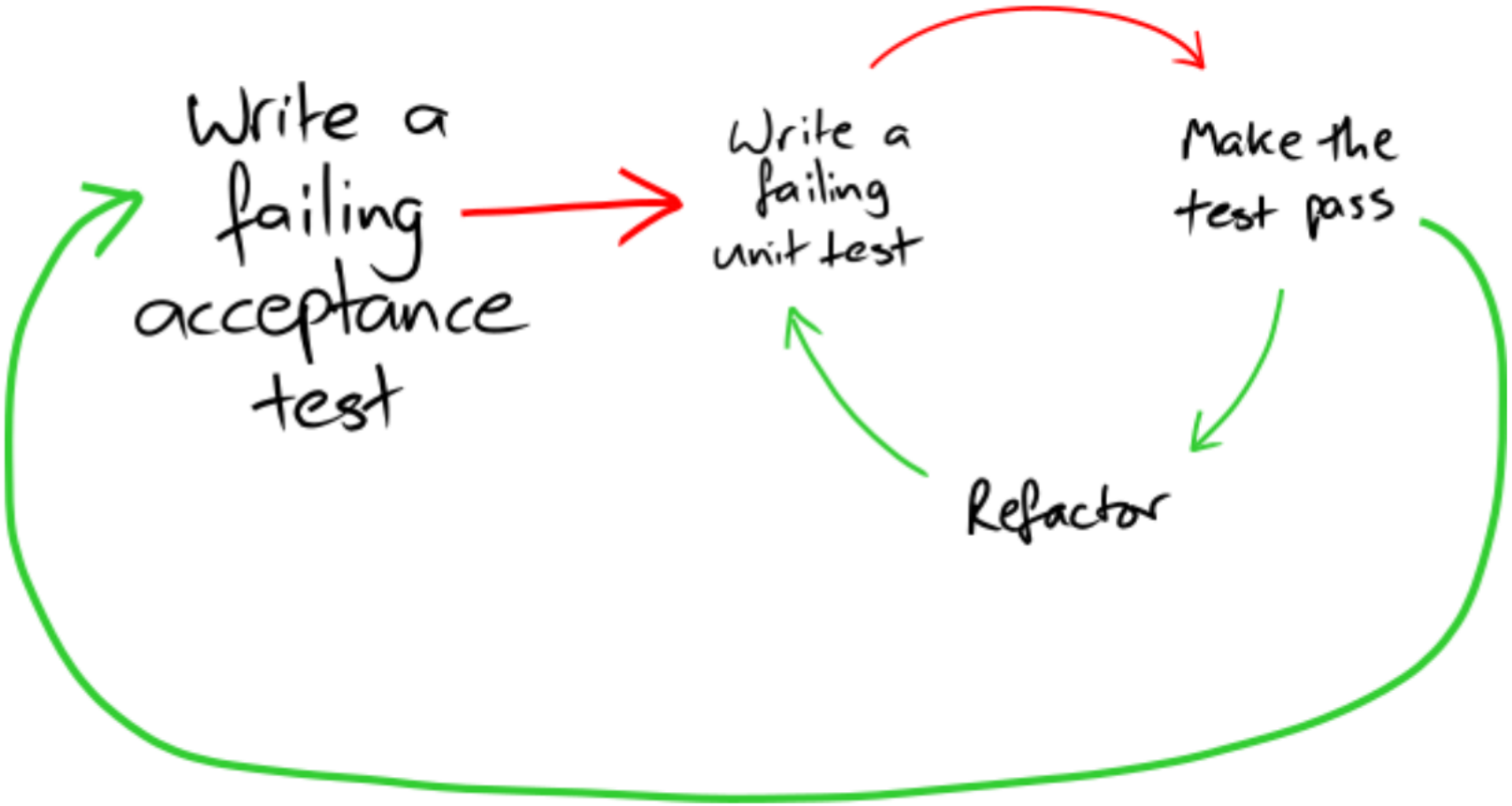
Executions paths - what is the problem?

Branching points	Integration tets	Unit tests
3	3	3
5	15	8
8	120	16
7	840	23
3	2520	26
5	12600	31



Outside in

Double loop TDD



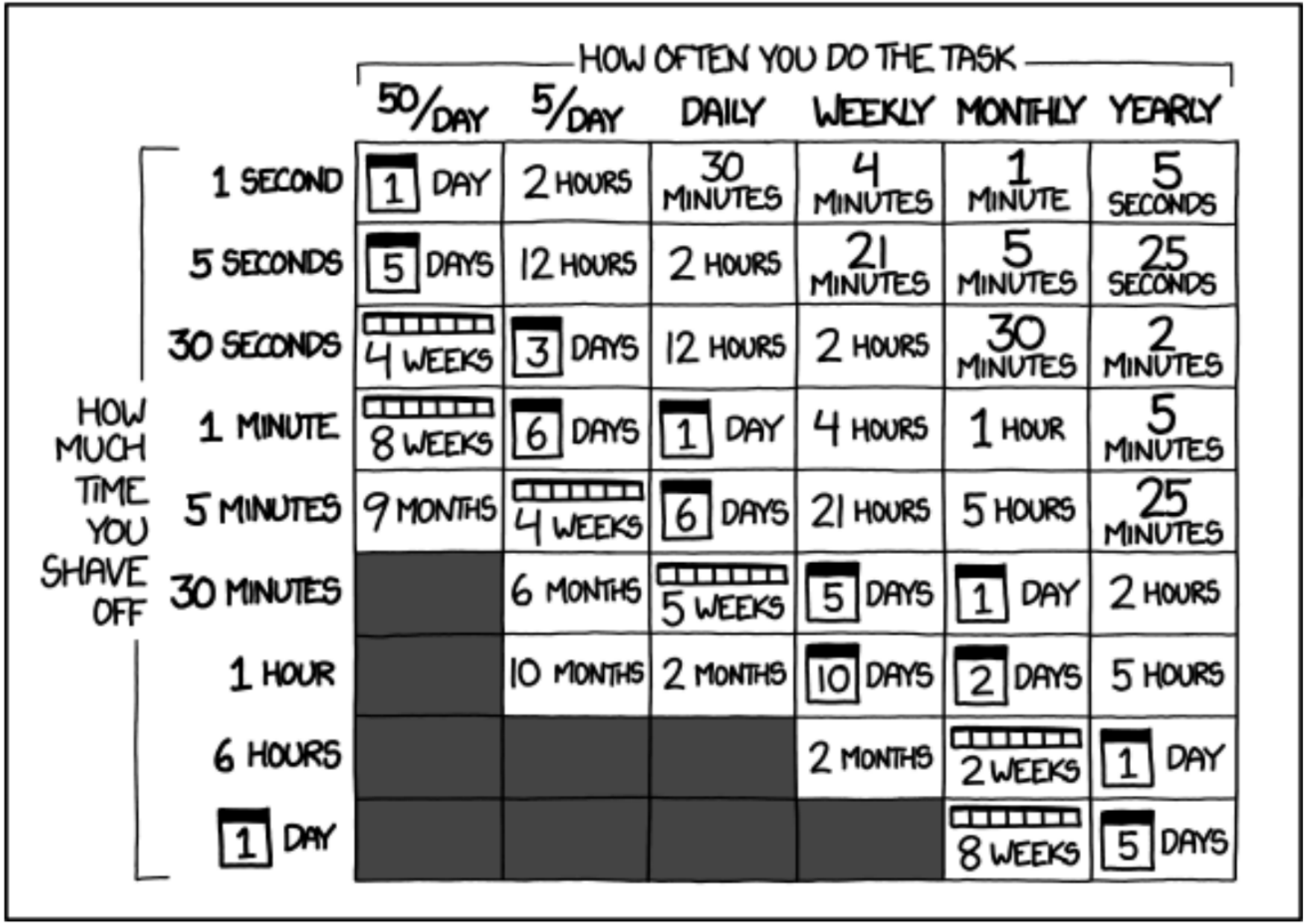
From Growing Object-Oriented Software by Nat Pryce and Steve Freeman

It takes so long time!

You have plenty of time...

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?

(ACROSS FIVE YEARS)



https://xkcd.com/1205/

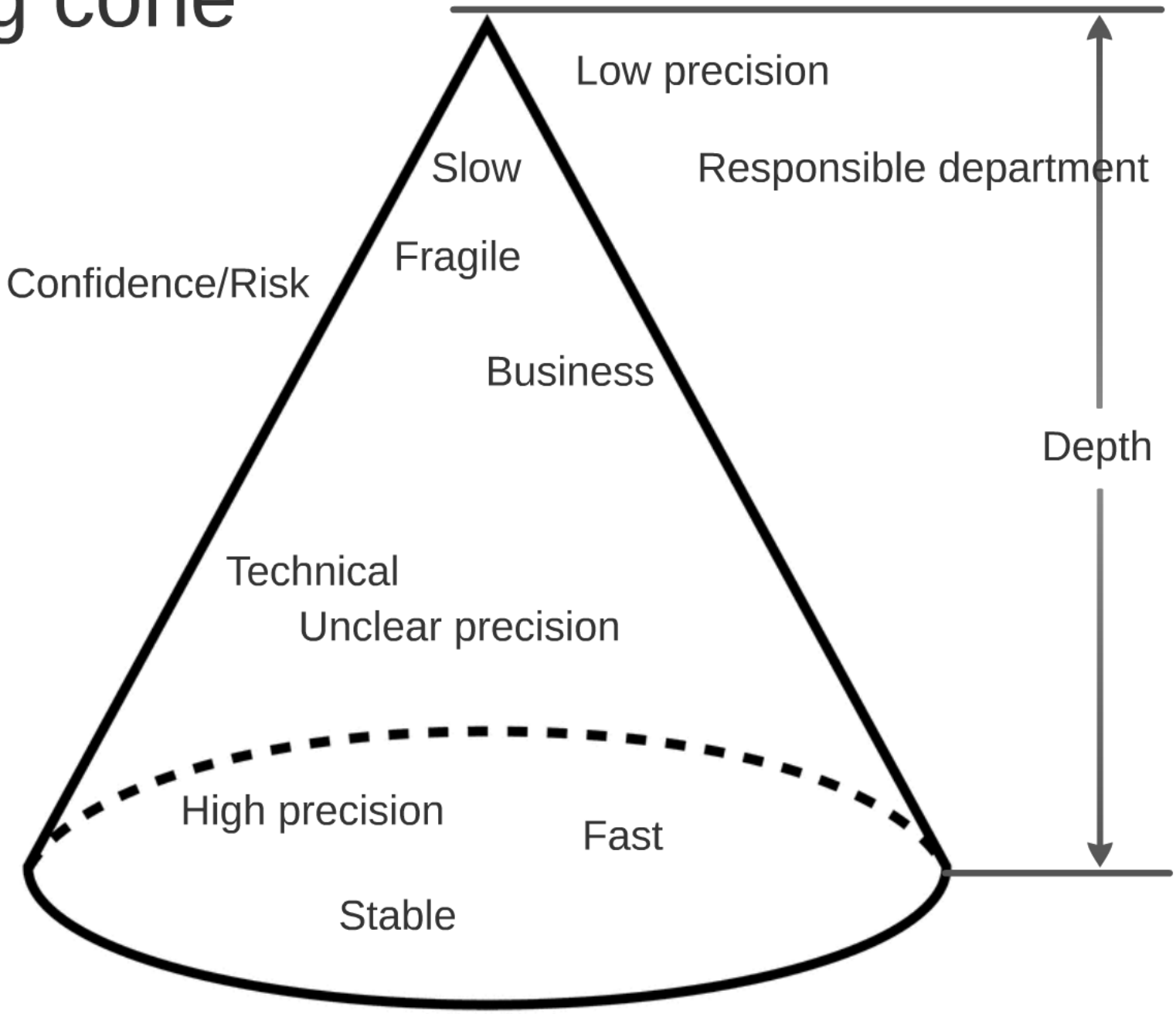
Always ask yourself
"How can this be tested?"

If it can't be tested, can it be used?

Can you test this with a unit test?

If not, what is the problem with your design?

Testing cone



Conclusion

A few deep tests

Does the system start?

Is it properly wired?

Do we have enough confidence to release?

Most shallow tests

One reason to fail - good feedback

Fast

Small - forces you to simplify collaboration

Can drive a good design

Crashing hard is not a problem,
if the altitude is low enough.

Kent Beck

Acknowledgments

@he_313

@codecopkofler

Software Dev Gang
Berlin, October, 2015

How deep are your tests?

Thomas Sundberg

Think Code AB
http://www.thinkcode.se
@thomassundberg
thomas@thinkcode.se