Unit tests
a philosophy and a help face to its own software

Feedback on 13 years of personal practice

GRAY SCOTT RELOADED SCHOOL – LAPP - ANNECY – 1/07/2024

Sébastien Valat

Inria

LABORATOIRE JEAN KUNTZMANN
MATHÉMATIQUES APPLIQUÉES • INFORMATIQUE

UGA
Université Grenoble Alpes

Tests unitaires une philosophie et une aide face à son logiciel

Retour sur 13 ans de pratique personnelle en HPC

GRAY SCOTT RELOADED SCHOOL – LAPP - ANNECY – 1/07/2024

Sébastien Valat

LABORATOIRE JEAN KUNTZMANN
MATHÉMATIQUES APPLIQUÉES • INFORMATIQUE

Université Grenoble Alpes

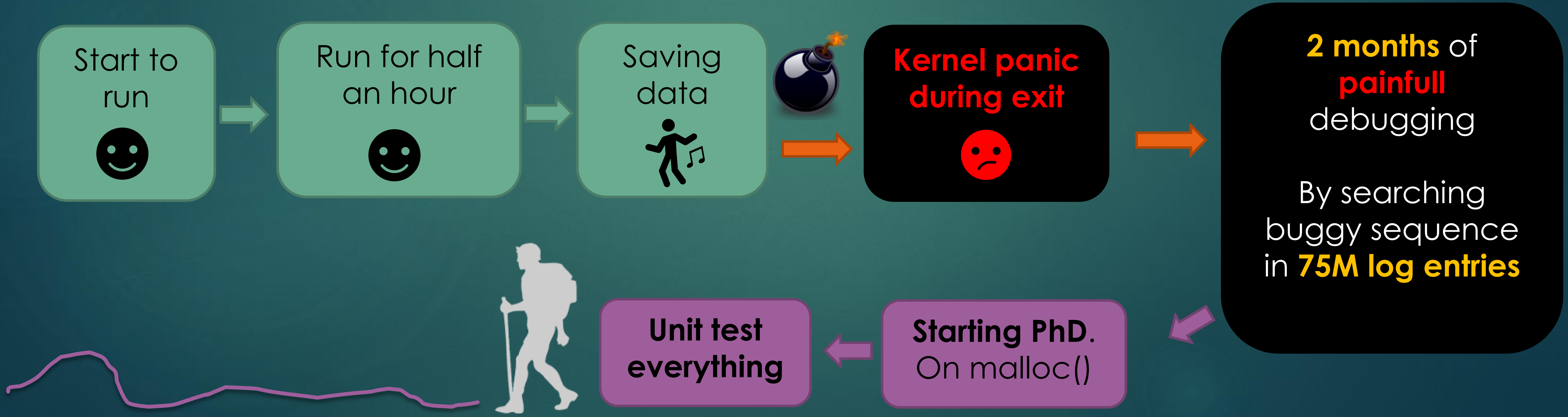
Plan

1. **Why I started**
2. **A little bit of philosophy & motivation**
3. **Thinking about testing methods**
4. **My own experience, feelings**
5. **Timings**

Why I started

Once upon a time…

- **Master theses (2009)** => **Linux kernel module**
- **5 months** : module is **working** well on **full KDE session** !
- **Lets try** on a real CEA **simulation** (**1,5 millions C++ lines app & 16 threads**)



Start to run

Run for half an hour

Saving data

Kernel panic during exit

2 months of **painfull** debugging

By searching buggy sequence in **75M log entries**

Starting PhD. On malloc()

Unit test everything

My source of thinking

- **Mostly my own** (home / PhD / post-docs / engineering) **work**
  - I hardly unit test since **13 years**
  - 4 years of **scrum** dev in team
- Sample
  - **17** projects
  - **190129** code lines
  - **C++** / **C** / rust / python / NodeJS / Java / GO
  - From **3700** lines to **33173** lines
  - Code coverage starting from **43%** to **93%**
- **Some projects without unit tests !**
  - **150 000 lines project & 50 devs**

A little bit of philosophy & motivation

How much mistakes costs later .. ?

- **Manhattan** project, **1945**, **Hanford**

- There was a **nuclear reactor**
- For **plutonium** production

- **Takes** water in
- **Cooled** the reactor
- ….and **dump** the water **out**…

https://commons.wikimedia.org/wiki/File:Hanford_N_Reactor_adjusted.jpg

Then there was wastes to handle…

- **Easy** and **quick** and **cheap** solution

- Make a **hole**,
- **Dump** everything in
- **Cover** with sand.

- **Costs** estimation…. ~12 mens,
- An excavator
- A truck

http://www.planetexperts.com/hard-lessons-in-the-us-environmental-protection-agency/

Then there was wastes to handle…

- For **liquids / muds**….
- Solution was to build 177 **tanks**
- **Store** 710,000 m^3

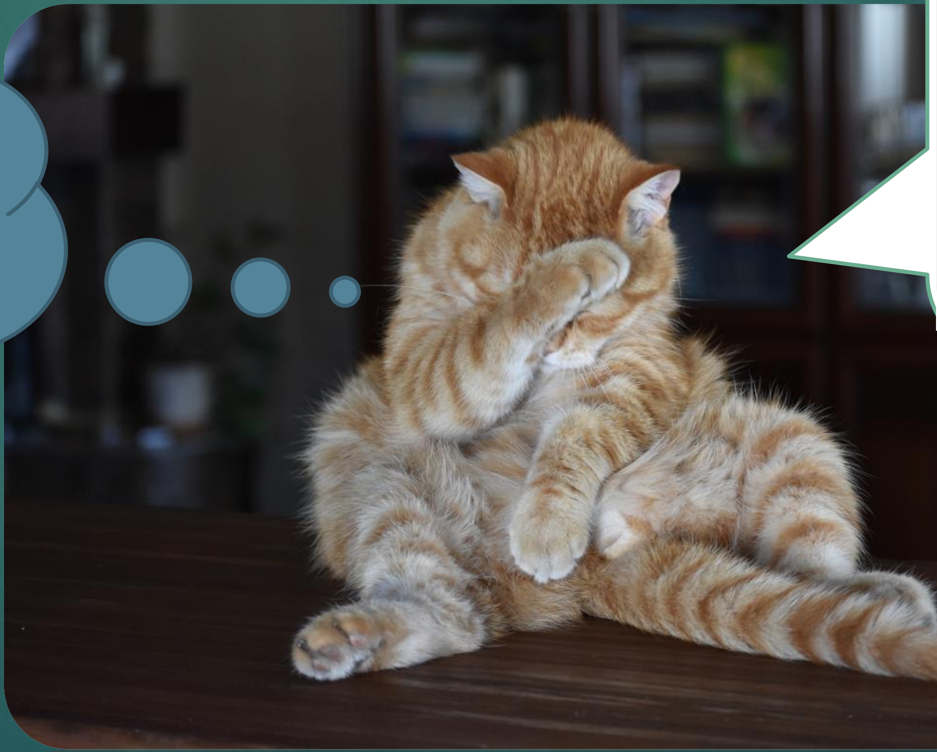
- In the **desert**,
- Dump wastes in
- And **cover with sand**….

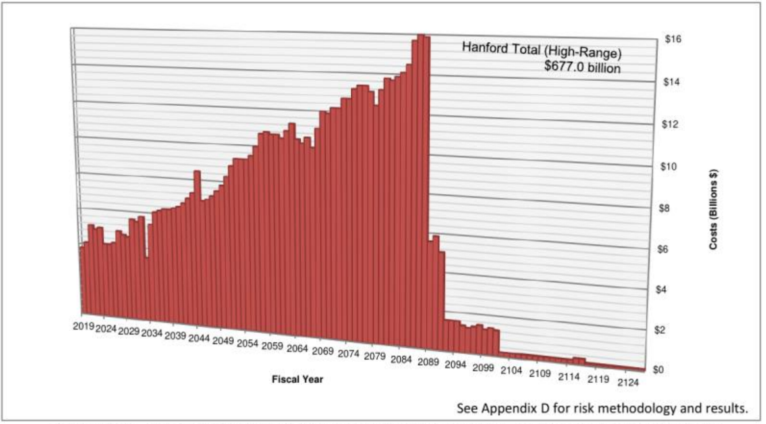
- Now, **55 years** later….
- They now (2010) **start to leak**…

https://tlarremore.wordpress.com/2016/02/28/uncontrolled-spread-of-contamination-nuclear-waste-material-hanford-nuclear-reservation-usa/

Today: that's **technical debt**

Cleanup until 2090
And estimated
~300-600 billions $.





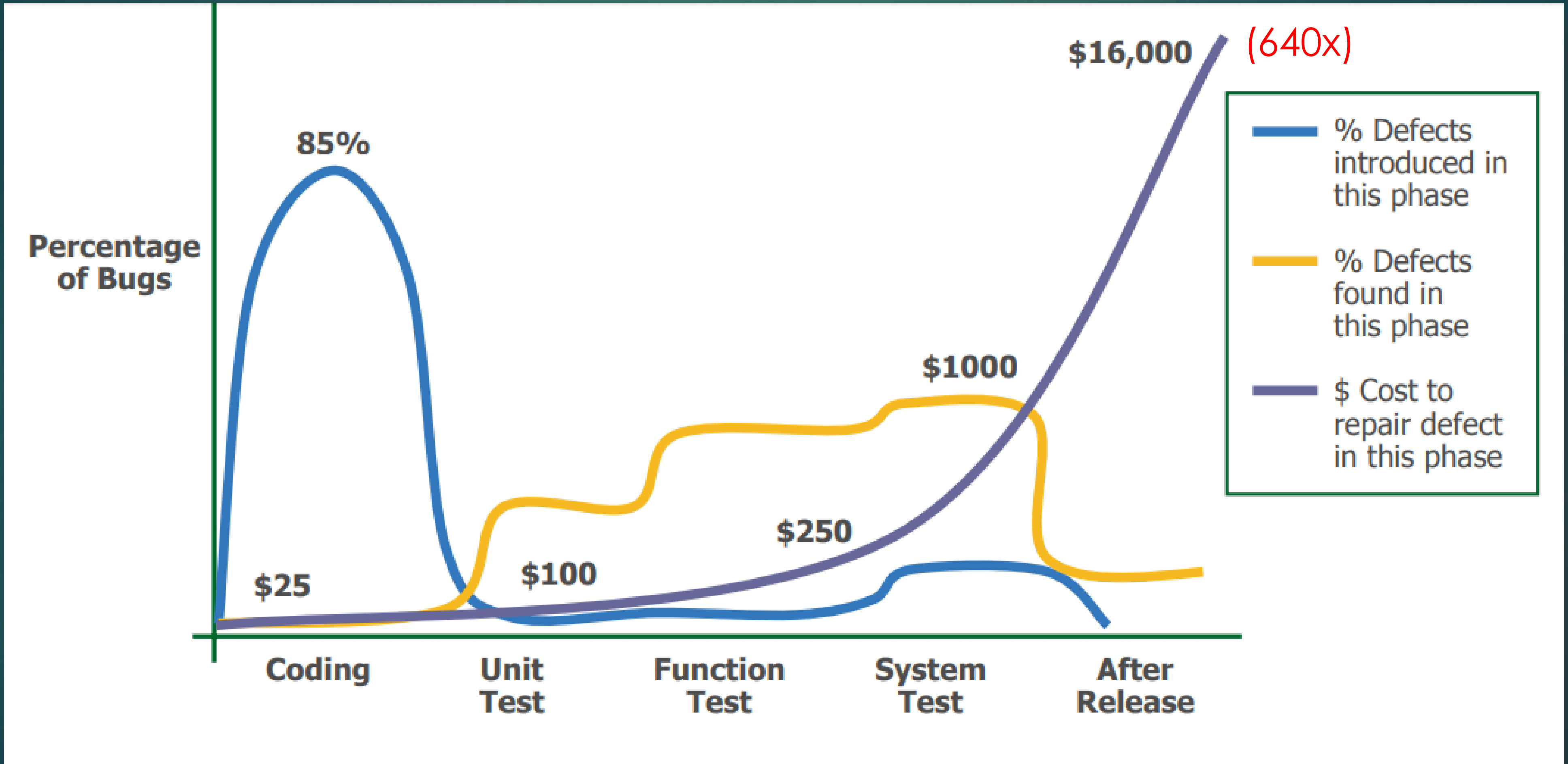
See Appendix D for risk methodology and results.

Figure ES-2. Hanford Site Remaining Estimated Cleanup Costs (High-Range) by Fiscal Year (includes both RL and ORP).

https://pixabay.com/photos/cat-redhead-striped-funny-posture-3602557/
https://www.hanford.gov/files.cfm/2019_Hanford_Lifecycle_Report_w-Transmittal_Letter.pdf

Came back to software….

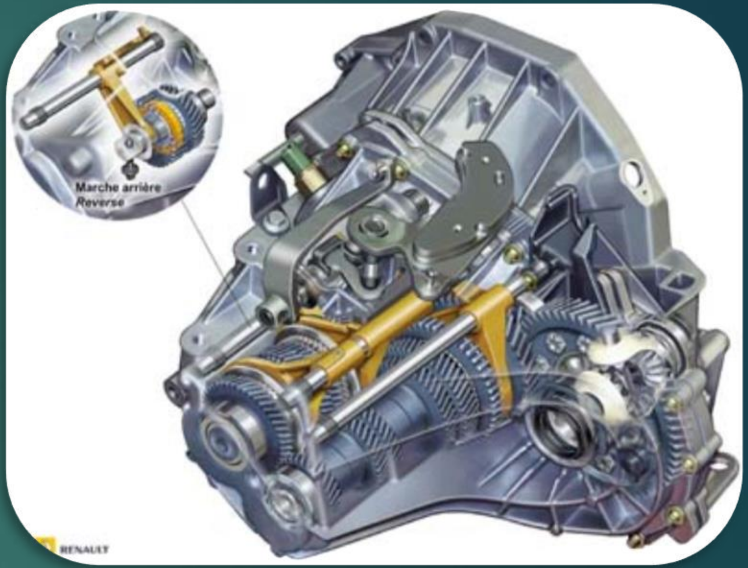
Capers Jones, 1996



Source: Applied Software Measurement, Capers Jones, 1996

Thinking about testing

Lets think you are a car engineer



http://dwww.auto-innovations.com/site/images8b/Renault_scenic_TL4.jpg

- You work for Renault (we are French… :D)
- You want to **build a car**
- You work on the **gear box**

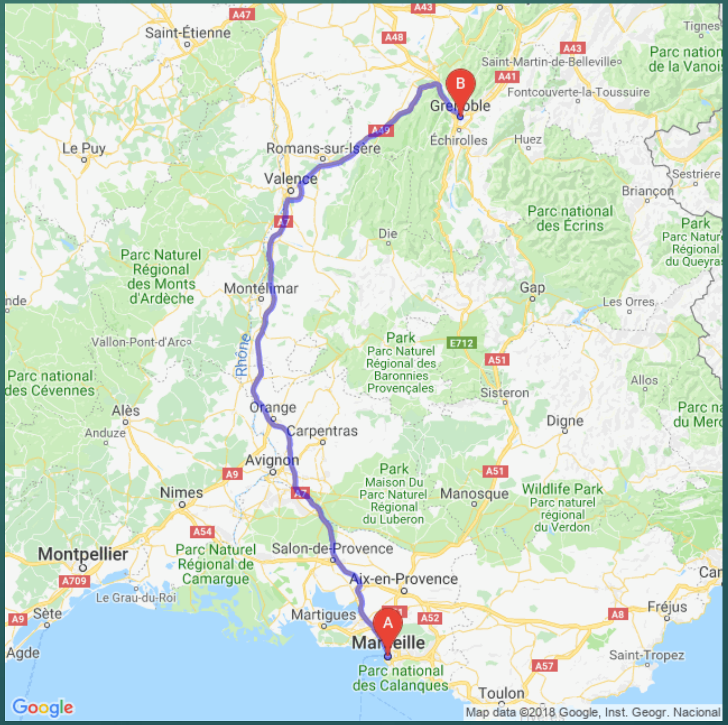
You make no test…

- **Sell** the car **directly to customer** and **see**

- **Would you by ?**

Method 1 : manual test

- Way to test a **new gear** we added
- Make a Grenoble – Marseille

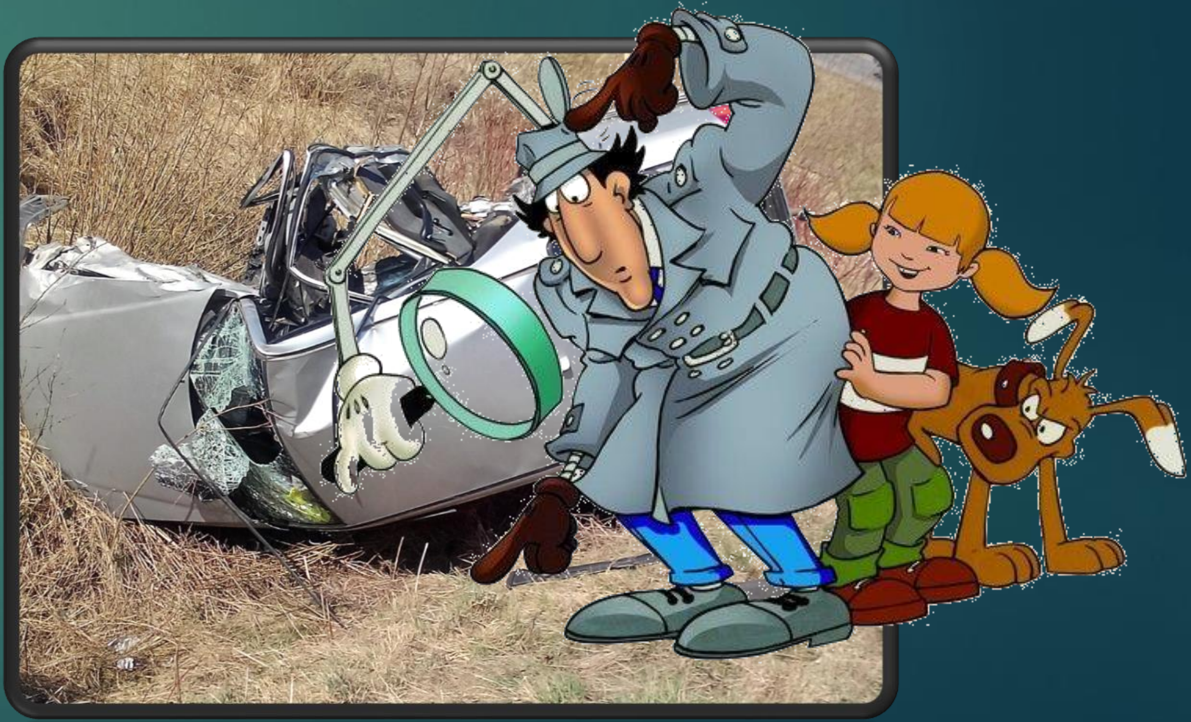

Saint-Étienne
Saint-Martin-de-Belleville
Parc national de la Vanoise
B
Grenoble
Fontcouverte-la-Toussuire
Échirolles
Huez
Le Puy
Romans-sur-Isère
Valence
Sestriere
Briançon
Parc national des Écrins
Die
Parc Naturel Régional des Monts d'Ardèche
Montélimar
Gap
Les Orres
Vallon-Pont-d'Arc
Rhône
Park Parc Naturel Régional des Baronnies Provençales
Parc national des Cévennes
Alès
Anduze
Orange
Carpentras
Sisteron
Digne
Allos
Avignon
Nimes
Park Maison Du Parc Naturel Régional du Luberon
Manosque
Wildlife Park Parc naturel régional du Verdon
Montpellier
Parc Naturel Régional de Camargue
Salon-de-Provence
Aix-en-Provence
Le Grau-du-Roi
Sète
Agde
Martigues
A
Marseille
Fréjus
Saint-Tropez
Parc national des Calanques
Toulon
Google
Map data ©2018 Google, Inst. Geogr. Nacional

Méthode 2 : **manual testing**

- A bit better : **in controlled environment**
- **Test circuit**
- Get a precise **list** of **tests to perform**
- **We need to define**
  "a test plan"

Method 3 : **automated integration** tests

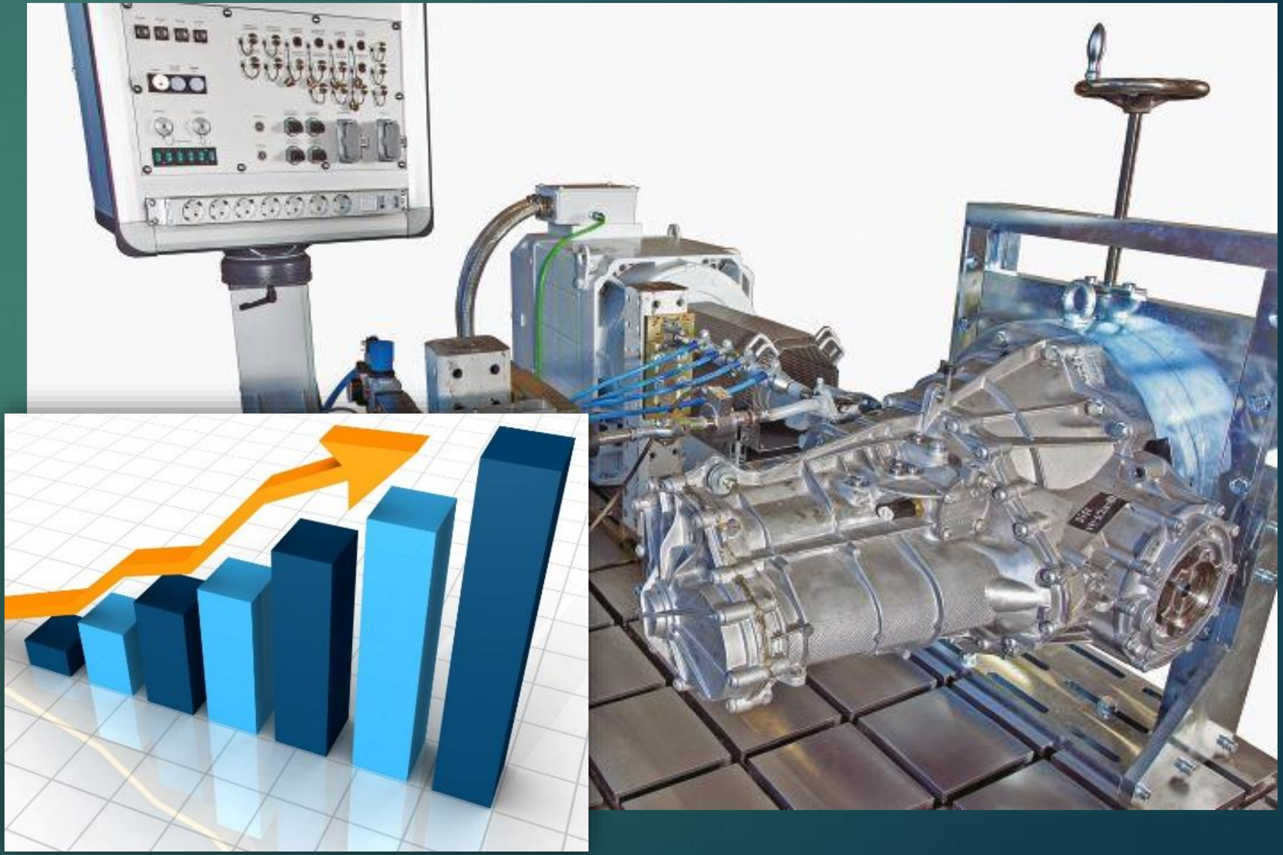
- We **build** a **prototype** and we run the **tests**
- **Each time** you **change** a **gear** in the **gear box** ?



https://es.wikipedia.org/wiki/Veh%C3%ADculo_aut%C3%B3nomo#/media/Archivo:Hands-free_Driving.jpg
https://www.needpix.com/photo/download/367388/accident-auto-crash-car-road-free-pictures-free-photos-free-images-royalty-free
http://maguy69.m.a.pic.centerblog.net/o/969011b4.jpg

Method 4 : **unit** test

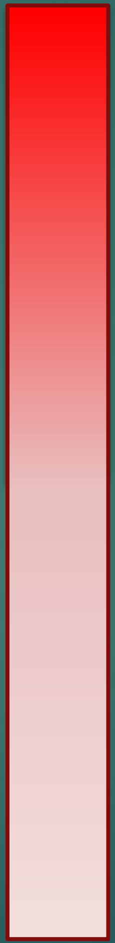
- You use **a test bench**
- Test **only** the **gear box**
- In **controlled situation**
- Can:
  - put **infrared camera**
  - **Probes** to see temperature.
  - **Vibration measurement**

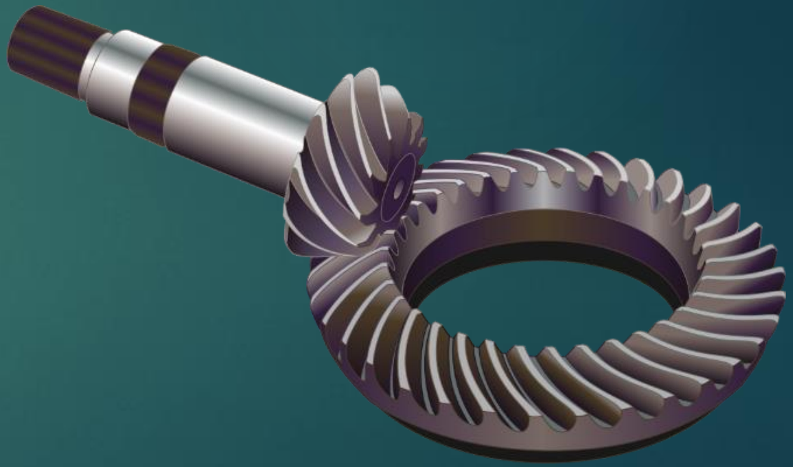


https://www.techbriefs.com/component/content/article/tb/features/application-briefs/13978

Notice contiguous transition….

- There is **unit test**
  - Test **one gear**
- A little bit more, still unit test
  - Test **two gears**
- …
- A little bit more, **integration** test
  - Test the **gear box**
- **End to end**, now **test in the car**.





https://www.indiamart.com/proddetail/automotive-spur-gear-19598784273.html
https://en.wikipedia.org/wiki/Spiral_bevel_gear#/media/File:Gear-kegelzahnrad.svg

Run example - OK

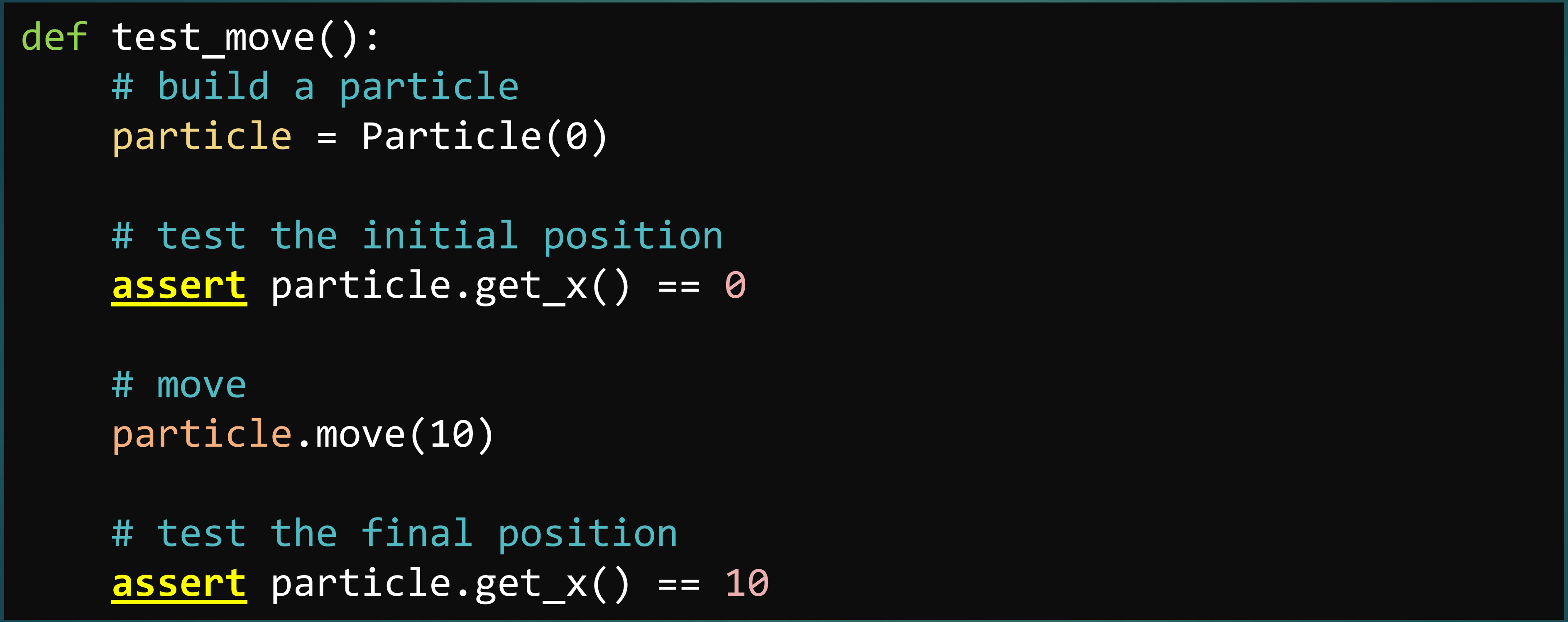
```
sebv@sebv6:~/2022-01-unit-test$ 
```

But how it looks ?

- The simplest test in python :

```
def test_abs_value(self):
    assert abs_value(-10) == 10
    assert abs_value(10) == 10
```

What is a unit test in python ?



```python
def test_move():
    # build a particle
    particle = Particle(0)

    # test the initial position
    assert particle.get_x() == 0

    # move
    particle.move(10)

    # test the final position
    assert particle.get_x() == 10
```

A bit more advanced one

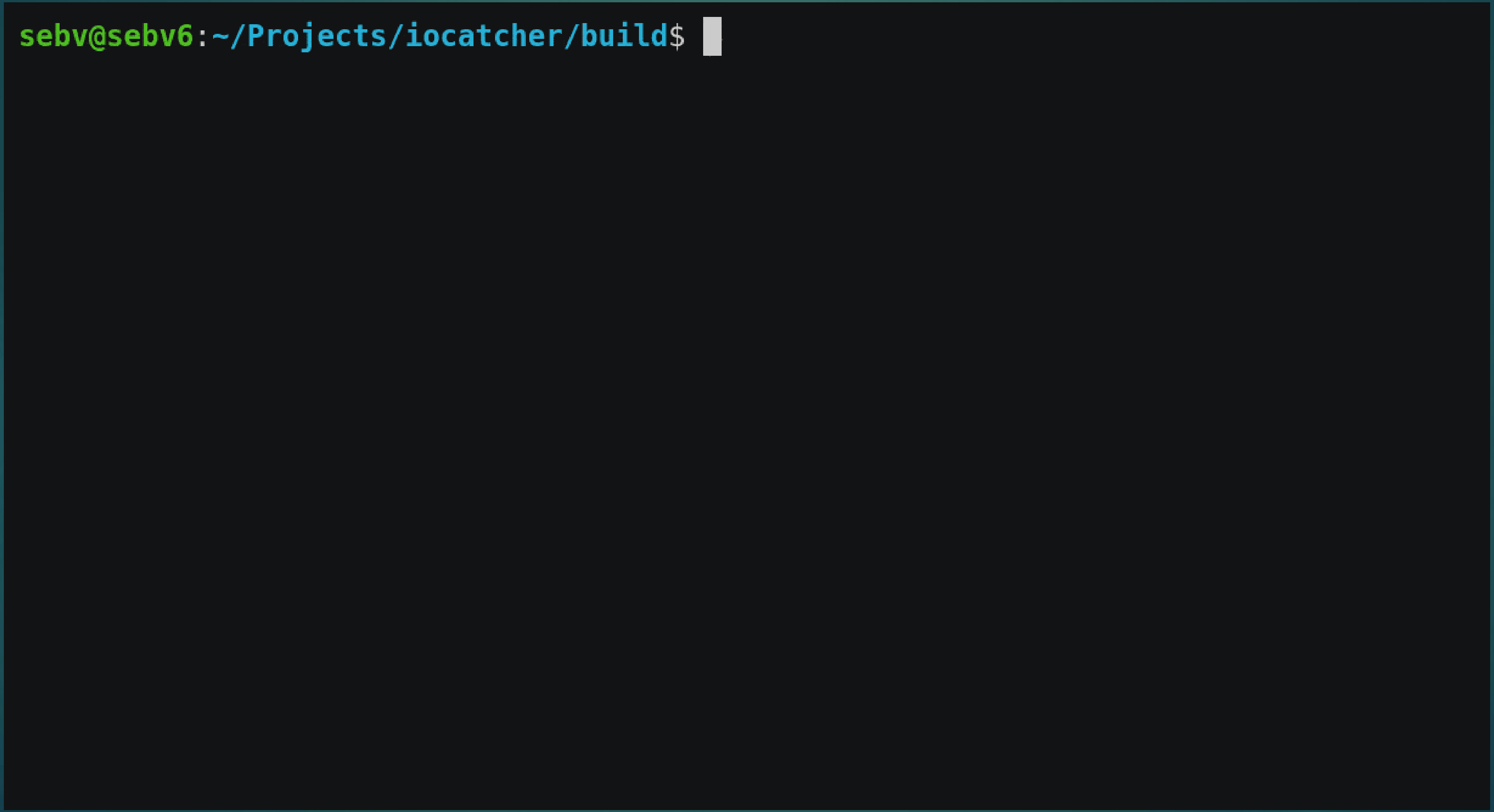
```
def test_collide():
    # build two particles
    particle1 = Particle( 0,5, -1.5)
    particle2 = Particle(-0,5,  1.5)

    # collide particules
    dt = 1.0
    collide = Physics.elastic_collide(particle1, particle2, dt)
    assert collide == True

    # checks
    assert particle1.get_vx() == 1.5
    assert particle2.get_vx() == -1.5
```

Most unit test frameworks relies on:
assert keywords

Run example - failure

```
sebv@sebv6:~/2022-01-unit-test$ 
```

A realistic case



```
sebv@sebv6:~/Projects/iocatcher/build$ 
```

My my own experience, feelings

When trying to push in teams…. [integration]

- **Integration test**
  - Mostly **everybody agree**
  - Not exactly on the way to do it….
  - One dev. already made a **dirty bash script** !
  - Seems easier at first look
- **Quickly cost a lot**
  - Eg. CEA project, **10 000** MPI tests, …. **5000 fails**…
  - **One week** to run everything
  - **Depressing**
  - Harder to debug
  - **Nobody looked** on results except me and another one

When trying to push in teams…. [integration]

- **Another integration case (costs):**
  - Eg. in another team (**scrum**)
  - Only integration test
  - Test suite time : **40 minutes**
  - Day to day **maintainance** : **1.5 dev fully dedicated** to it (team 15 dev)

versus **9.42 seconds** in unit

- If your CI env is not stable:
  - **Lots of issues** to maintain the env running
  - Lots of **non code related issues** (**timeout**, …)
  - Company **migrated the CI env** : ~**5 months consumed to migrate**

versus **1 week** in unit

When trying to push in teams…. [unit tests]

- **Unit tests**
  - Required an investment
  - Initial effort
  - We are slower to start
  - **Hard** to **convince** devs who never made unit tests
  - **Hard to introduce in pre-existing software**

- **Common first kill** :
  - “This one is **too hard** to test” ❌
  - “This one **call many others**” ❌
  - “I’m sure of this function, it is **so simple**” ❌
  - “Hola, **do not touch this part of the code !**” ❌

First time I made unit tests

- I was **not convinced**
  - But **I tried**
- Had the impression to **loose my time**
- It **was hard**
- I **didn't see the benefits**
- **I already had most of my codes**
  - Painfull to unit test for weeks

https://depositphotos.com/fr/vector/exhausted-emoticon-13971204.html

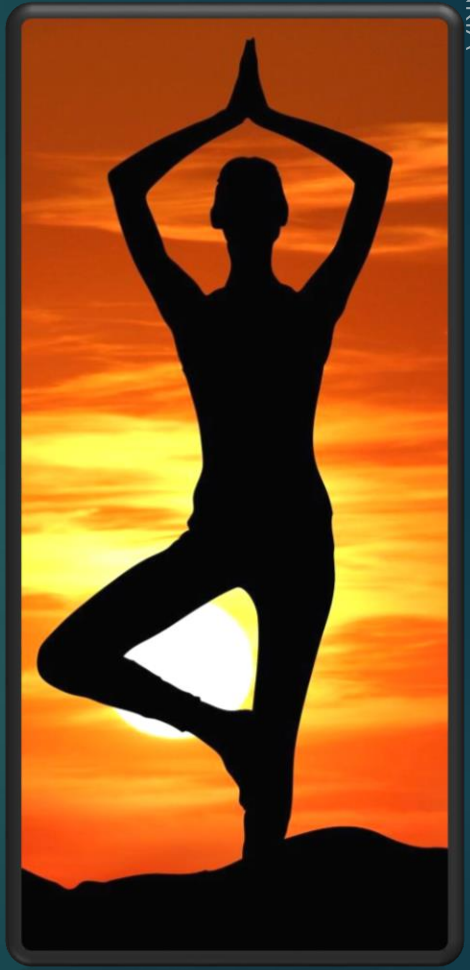
That’s also adequate tools and ways to work

Day to day methodology : **discipline**

- **“This is a POC…. I will make my tests later”** ✖

- **You will never do them later**
  - Because your ***design*** *will* ***not permit***
  - Because you will **want to move** to **other stuff**
  - **Nobody** will be **happy** to write unit **tests for ~4 weeks**
  - **Your boss/commercial manager already sold it to clients**….

- You already **loosed half the benefits** of unit tests
  - **Become** a **more or less useless investment**



Benefits of unit test

- **That’s not only testing** **(~20%)**
- It forces you to **think your design**
- Forbids **global variables**
- Make **spec**, also for **internal APIs**
- Open easy door for **refactoring / rewriting**
- **New developers** are more confident (**you in 6 months**…)

A safety for QA guy

- Quality loss and **rush warnings**.
- Noticed **via a technical channel** not through **quality exigent guy** !

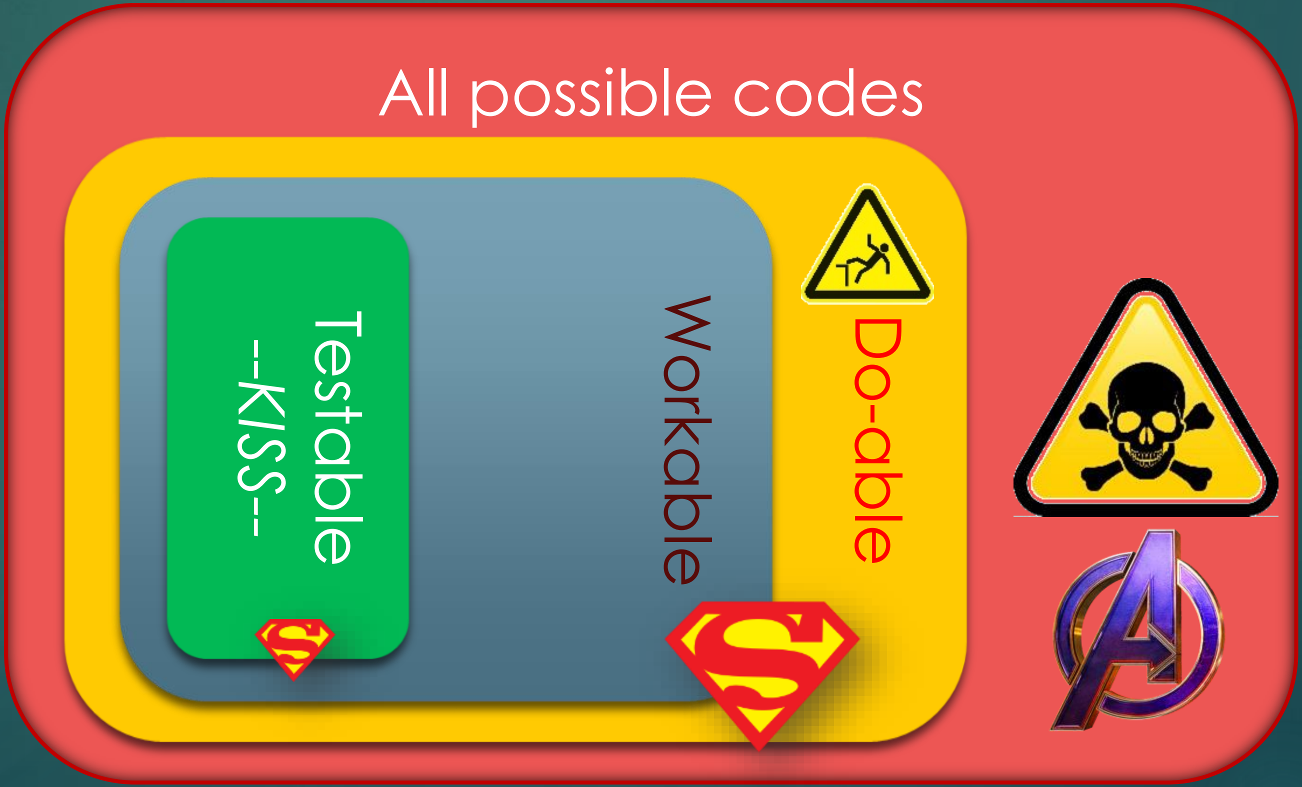
That’s a discret **teacher** !

- You get **feedback by yourself**
- **No** need to get **critics from someone else**
- If you **don’t know how to write** your test :
    - **Your internal API is badly designed** !

https://www.seekpng.com/ima/u2q8i1q8a9q8y3a9/

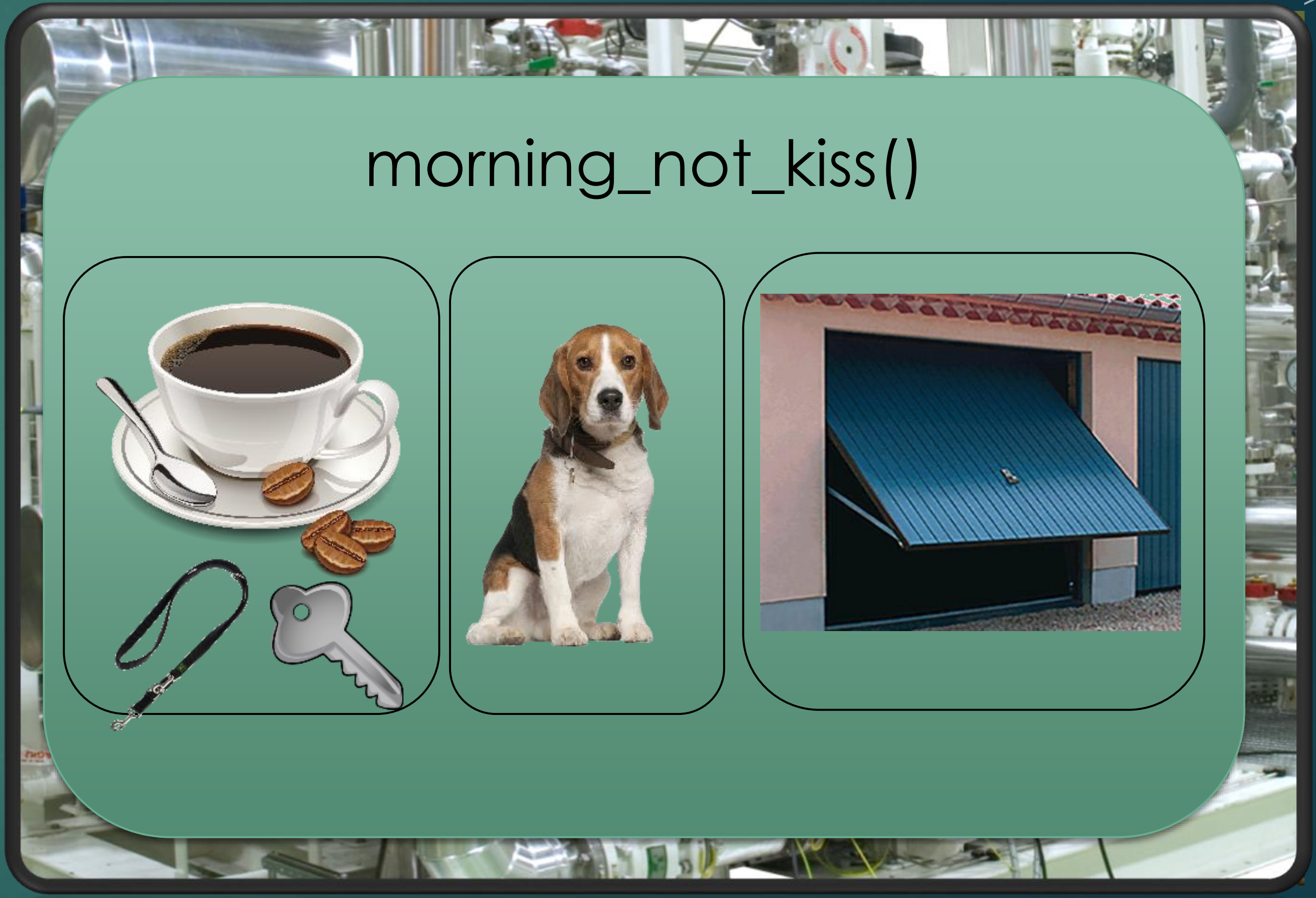
That's also constraints

- **Not all** codes are **unit test-able**

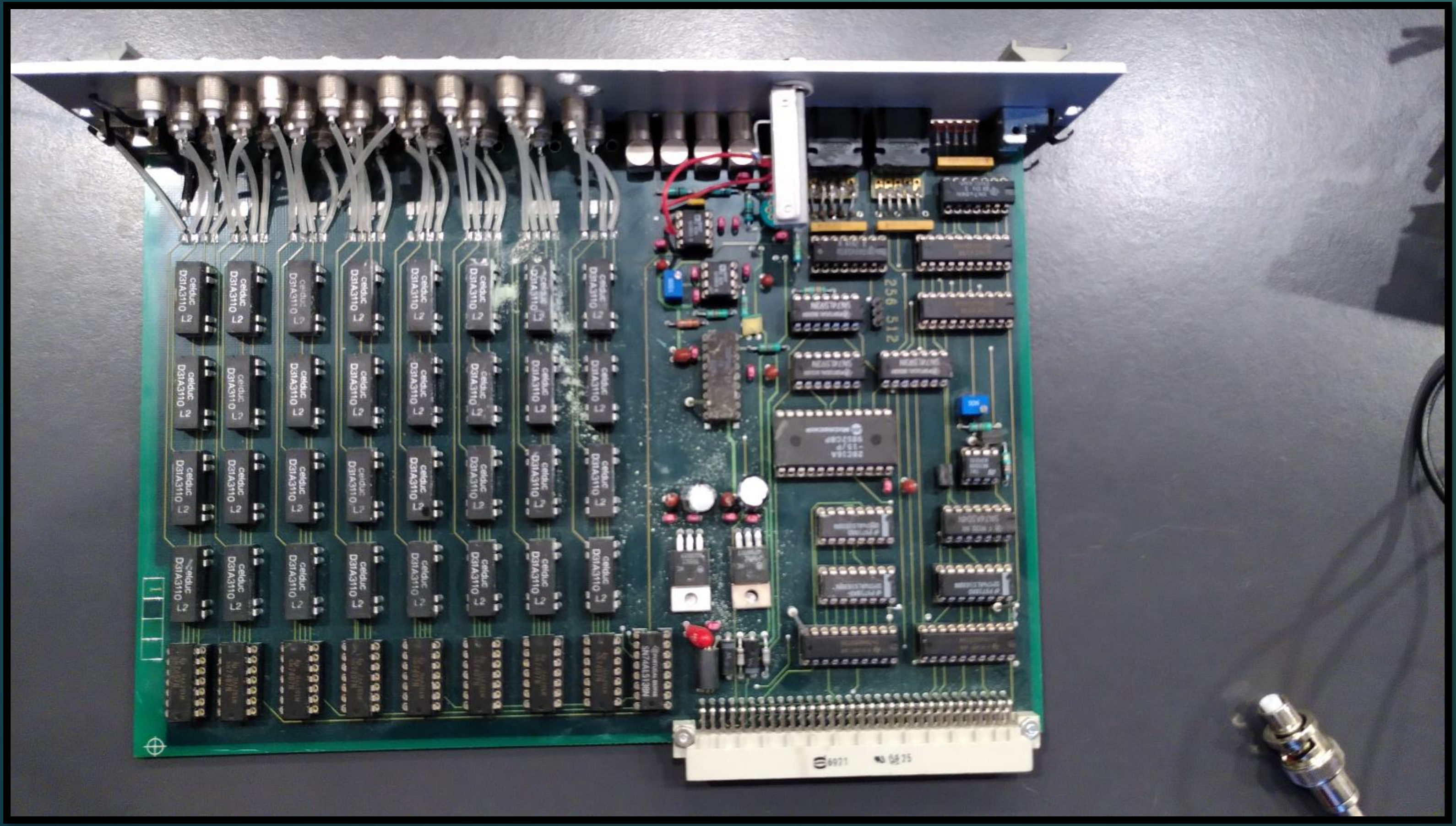


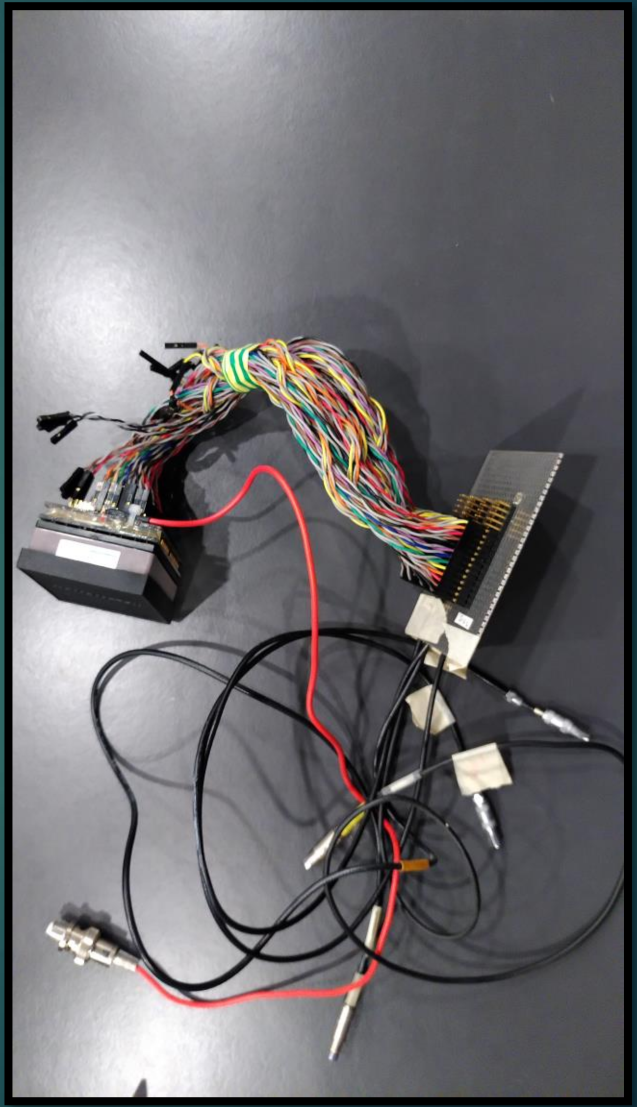
Test a gas machine

- If your **test** become **too complex**
- You are **certainly**
  on the **wrong way**
- **Stop**, **think** and **KISS**

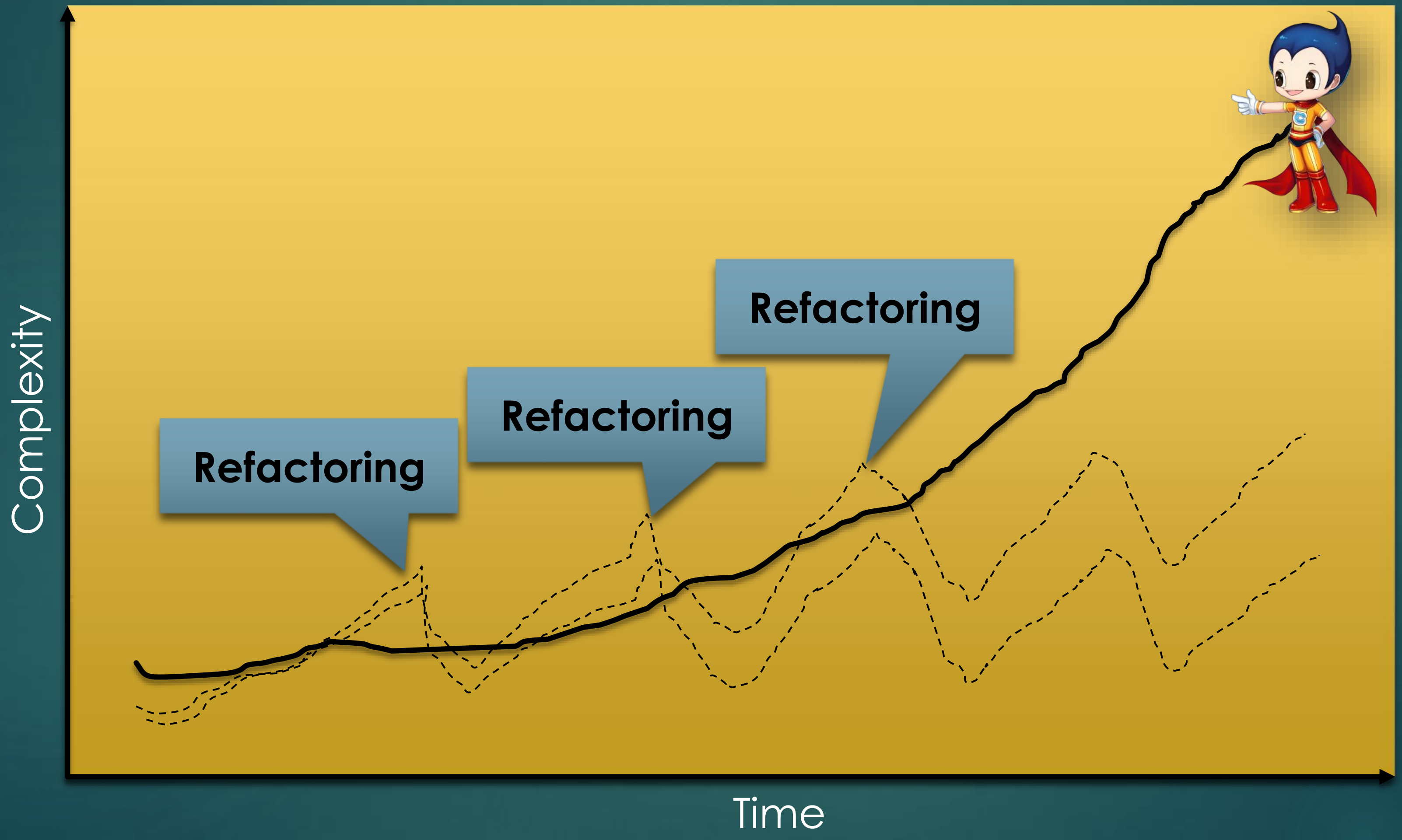


What looks simpler to test ?

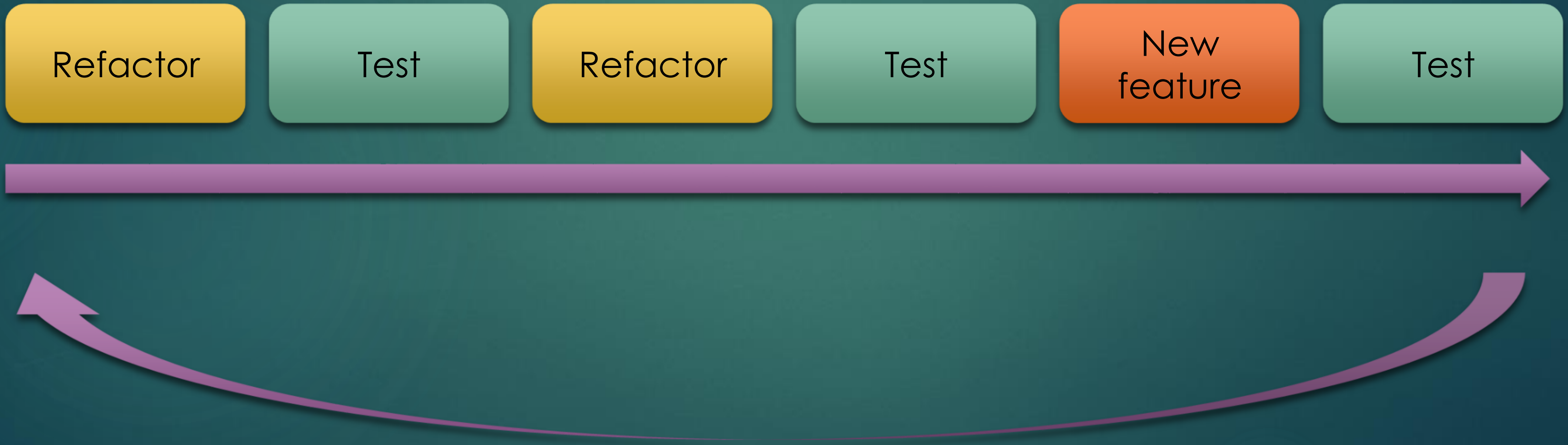




Facing the entropy !



Way of working



Team dynamic - silos

- **Without test** it is **hard to go** on a **part we do not know** !
- **Especially in HPC !**
- So **each dev** will have **his part**
- It **reduces** the **discussions** in the team
- **Favor heroes**

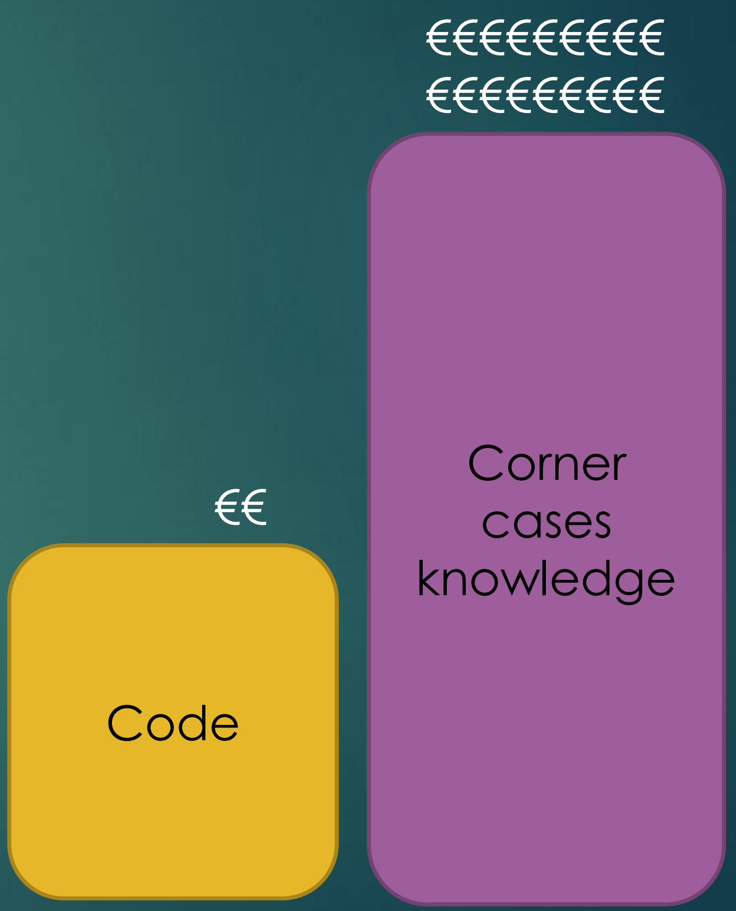
https://www.flickr.com/photos/docsearls/5500714140

Keep the knowledge !

- The hard **corner cases** are **encoded into the tests**

- Useful:
  - On **turnover** or **retirement**
  - Very **usefull** in case of **rewriting a V2**
  - To **translate** in another language

- Eg: **porting my memory allocator**:
  - **C original** implementation : **~1 year**
  - **C++ translation** + new algo **: 1 month**
  - **Rust translation** : **2.5 weeks** for the biggest part



A basement of **agile methods**

Never do **AGILE** or **SCRUM** **without unit test** !

- That's **a REQUIREMENT** for the method, **not an option**
  - For the **technical validity** of the method
  - For the **dynamic of the team**
- In agile you **didn't plan**…
  - If you **cannot refactor** => you are **scrued** **or** **get very lucky** !

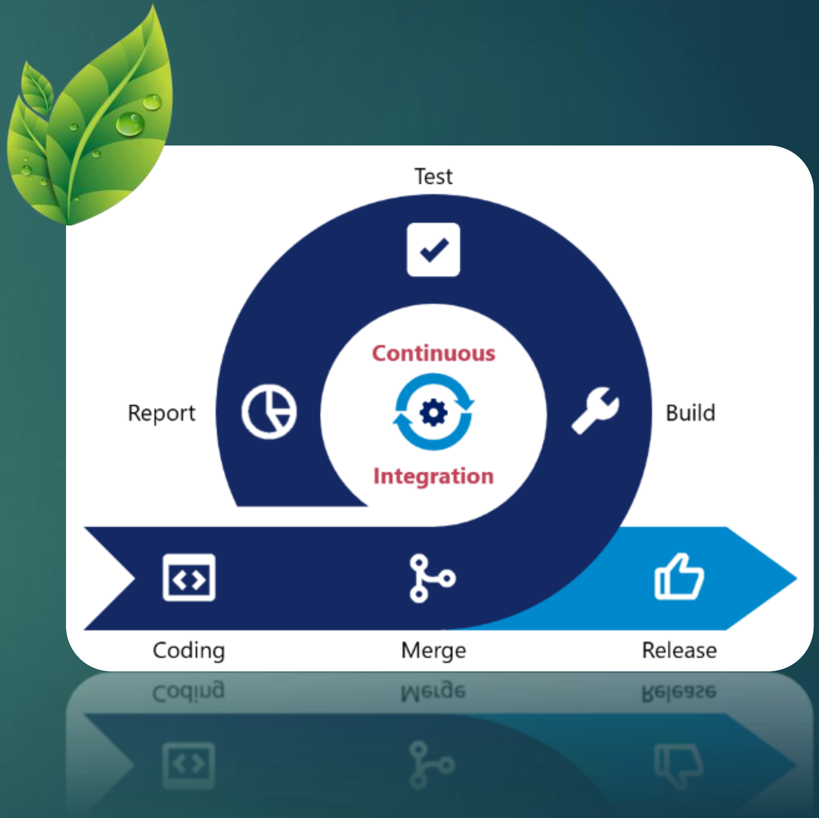
Ecology argument

- You can make the **whole dev localy** on your **laptop**
- **No need** of **a large dev cluster**
- **Once done** and validated with unit tests:
  - Make real **test on cluster**
  - **Once a week** or two weeks
- Not anymore per team cluster
- **Less debugging at scale so less CPU hours !**

https://www.freeiconspng.com/downloadimg/36144
https://www.vigindustries.in/products/server-rack

Ecology argument - 2

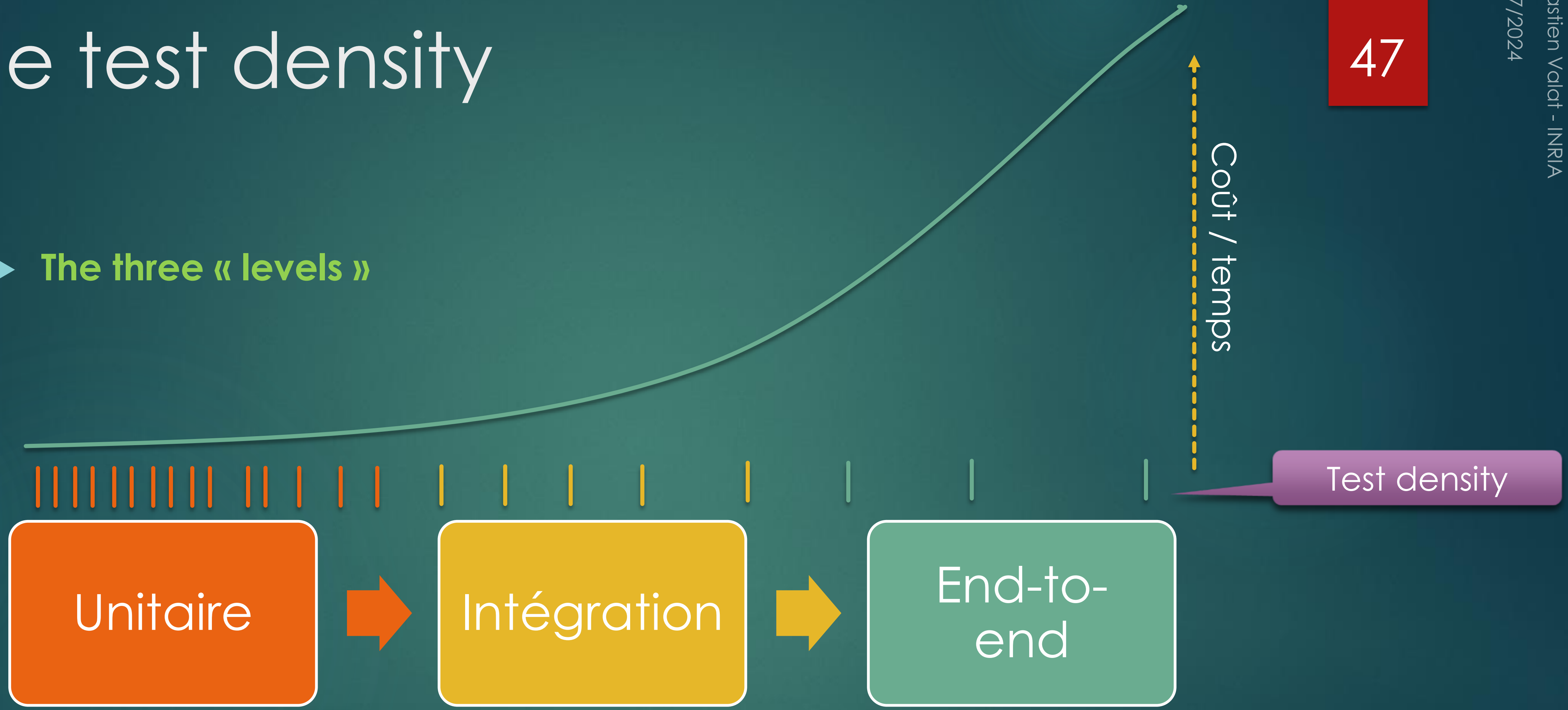
- CI cycle takes **~ a minute**
- Instead of **40 minutes**
  with only **integration tests**
- Require **less CI server** ressources



https://www.qfs.de/fr/blog/article/utiliser-qf-test-dans-des-systemes-continuous-integration.html

The test density

- **The three « levels »**



- **Build** (or use) **dedicated tools** for each level

Reproductibility

- It help **portability**
- **Reproduce** in an **other environnement**
- Because **no one** has the **same**…
- And it **evolve quicly**
- Not **stay stucked** to fixed **old versions**

Research

- You have **tons of ideas** !
- You want to **change path**
  - Refactoring python code ?
- Or **explore** others
- You **don't know where** you **go** !
- Not **loose time** in **debugging**

http://garesettrains.canalblog.com/archives/2020/07/14/38430007.html

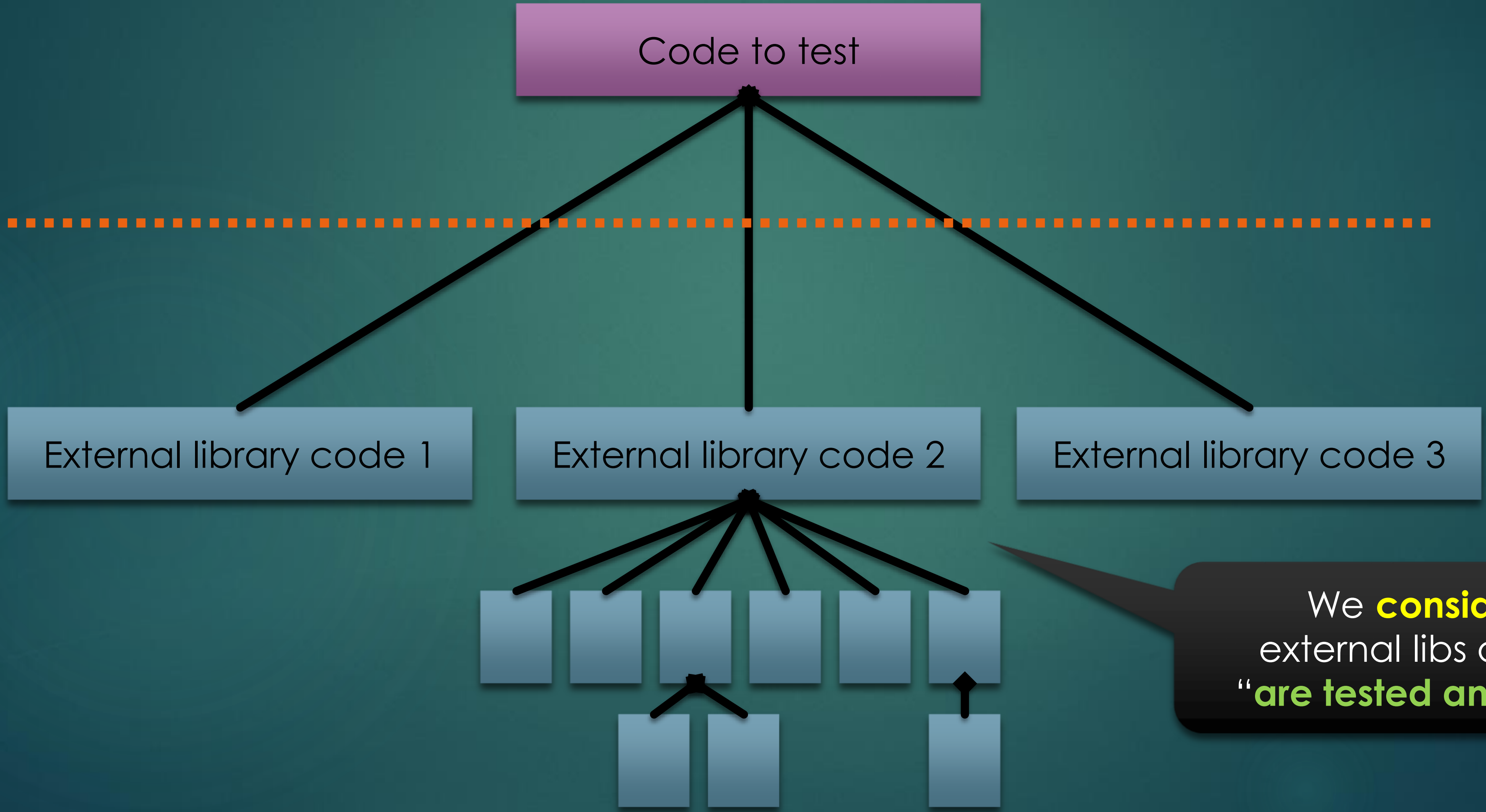
Mocking

The turtle case

- Many **unit test introduction courses** uses:
  - The **turtle** example
- A **turtle**
  - Has a **position**
  - Can **move forward**
- This is **a too simple example**
  - **Not pointing** problems of a real case

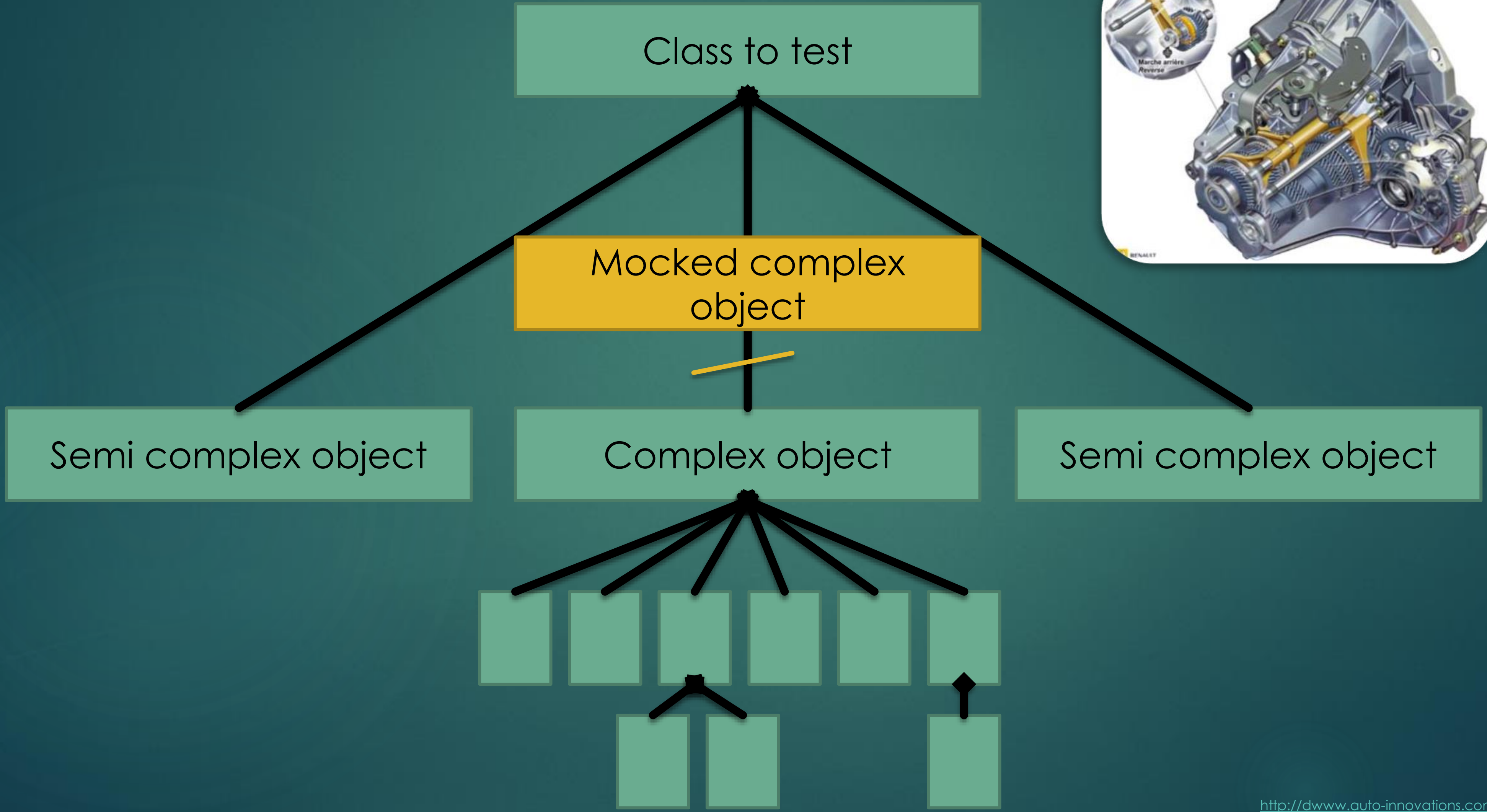
https://snipstock.com/image/png-images-turtle-24-png-17769

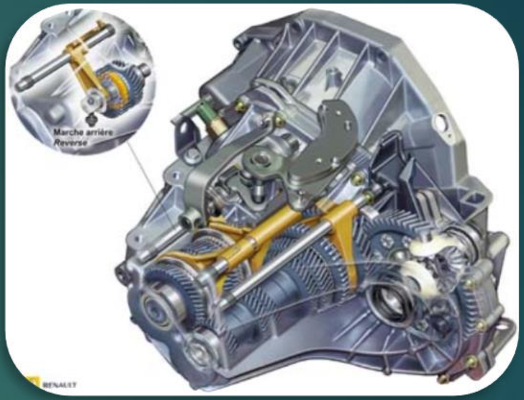
Should we test everythings ?



We **consider** that external libs are tested "**are tested and working**"

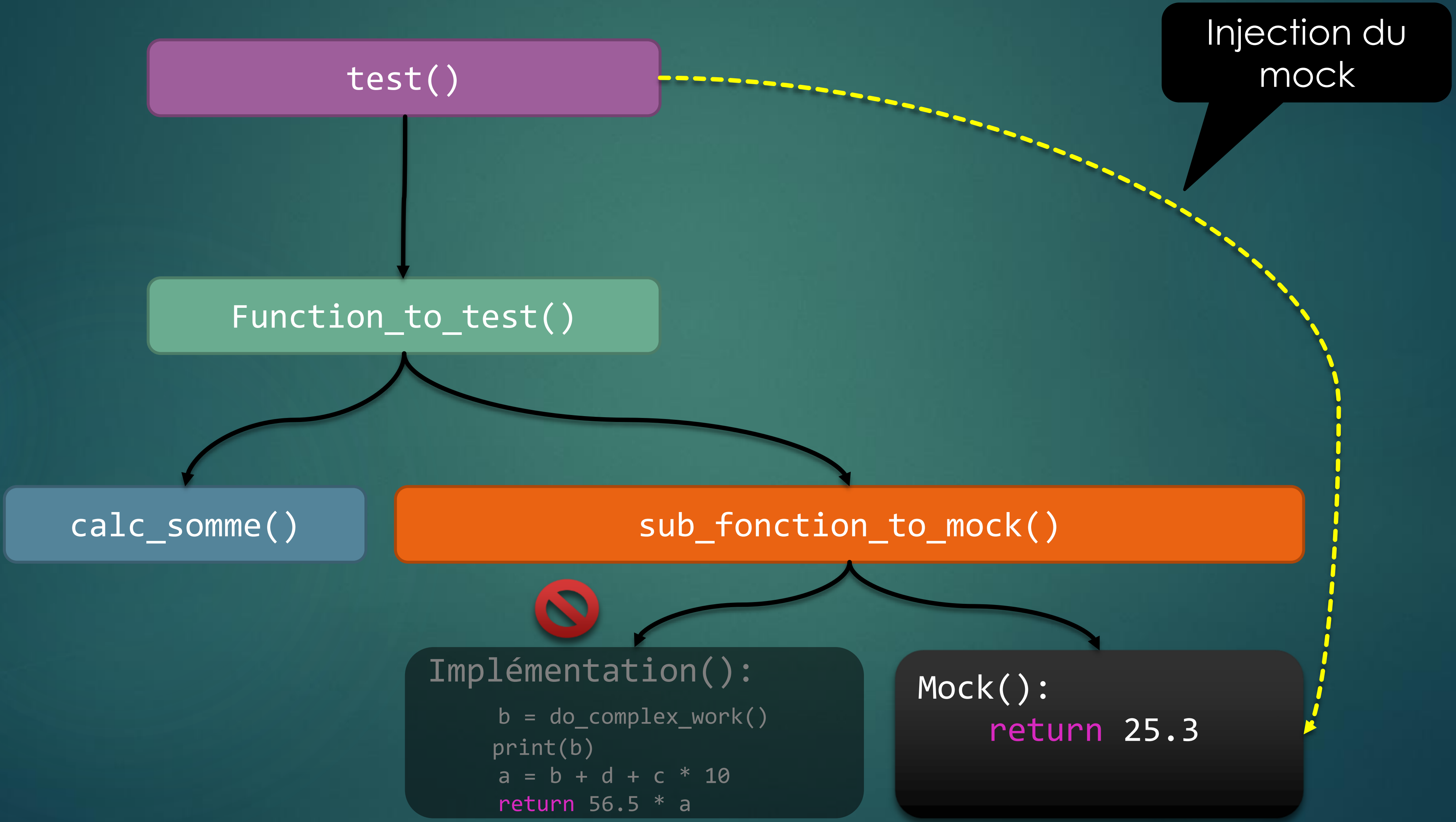
Intriducting **mocking** (factice)





http://dwww.auto-innovations.com/site/images8b/Renault_scenic_TL4.jpg

The mocking



```
Implémentation():
    b = do_complex_work()
    print(b)
    a = b + d + c * 10
    return 56.5 * a
```

```
Mock():
    return 25.3
```

What is a mock in python ?

```
From unittest import mock

def test_collide():
    # build two particles
    particle1 = Particle( 0,5, -1.5)
    particle2 = Particle(-0,5,  1.5)

    # override random function by mock
    with mock.patch(‘random.randrange’, return_value=0.75):
        # collide particules
        dt = 1.0
        collide = Physics.collide(particle1, particle2, dt)
        assert collide == True

        # checks
        assert particle1.get_vx() == 1.5
        assert particle2.get_vx() == -1.5
```

Some framework

Language	Test framework	Mocking
Python	Unittest pytest	unitest.mock
C++	Google test Catch2 Boost test library cppunit …	Google mock FakeIT
C	Google test Criterion	
Bash	bats	
Rust	[native]	*mockall*
Go	[native]	*gomock*

Timings on 1 examples

COSTS AND EXAMPLES

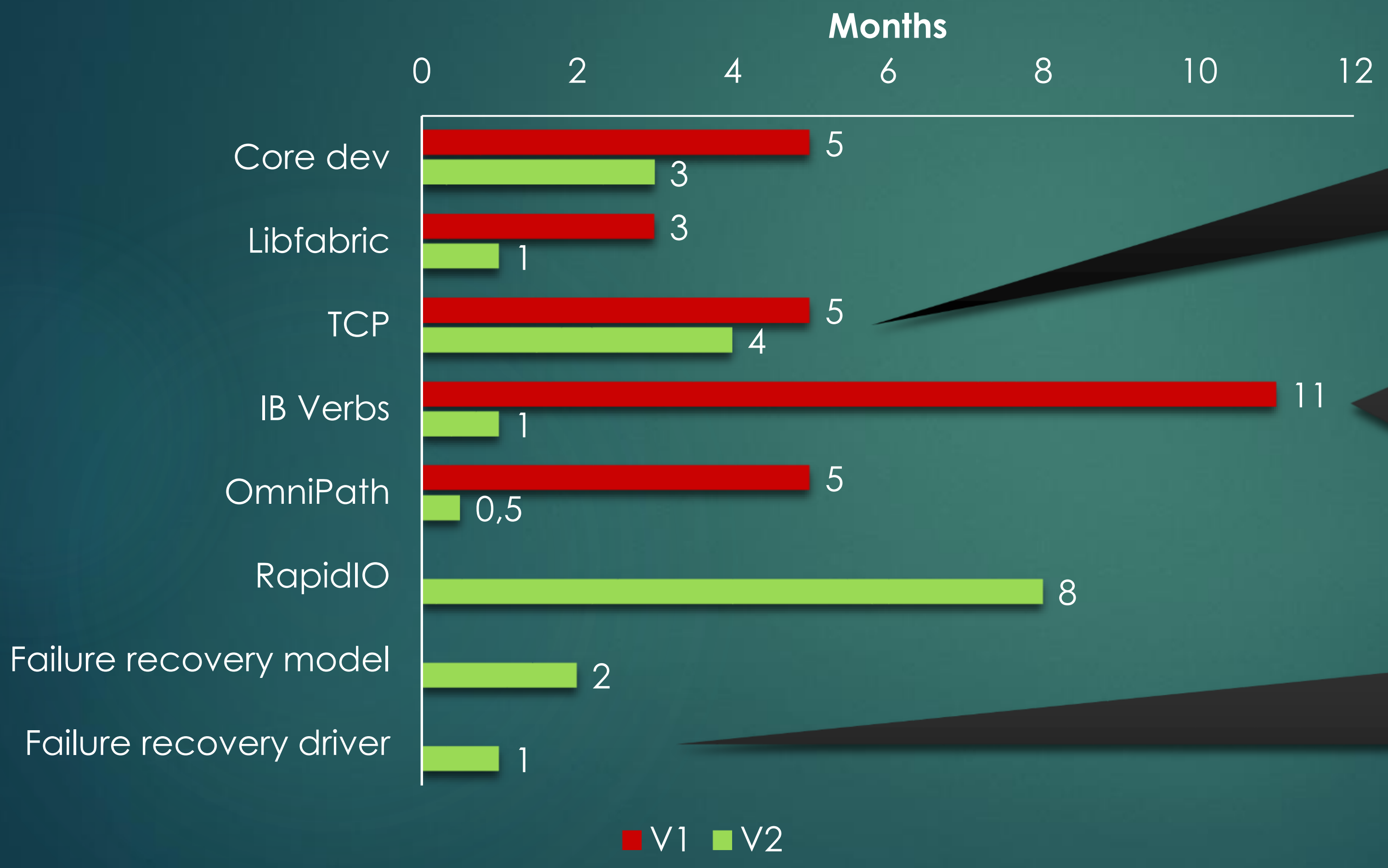
CERN lhcb-daqpipe

- **LHCB** Acquisition **R&D code** for **scaling studies**

- Need to handle **40 Tb/s** on
  - **InfiniBand**
  - **Omni-Path**
  - **100G ethernet**

- Over **500** servers (continuous **80 Gb/s all-to-all**) + send to ~**3000**

Compare costs



TCP driver :

V1 => network **expert**
V2 => **very basic** C/C++ **knowledge**

IB driver :

V1 => student made an **IB simu**.
V2 => **No** MPI or RDMA **knowledge**

Failure recovery:

Restart and **reconnect** in middle of continuous all-to-all communications (**InfiniBand**).

Not gaining only time !

DON’T BE AFRAID BY REAL PROBLEMS

MALT

December

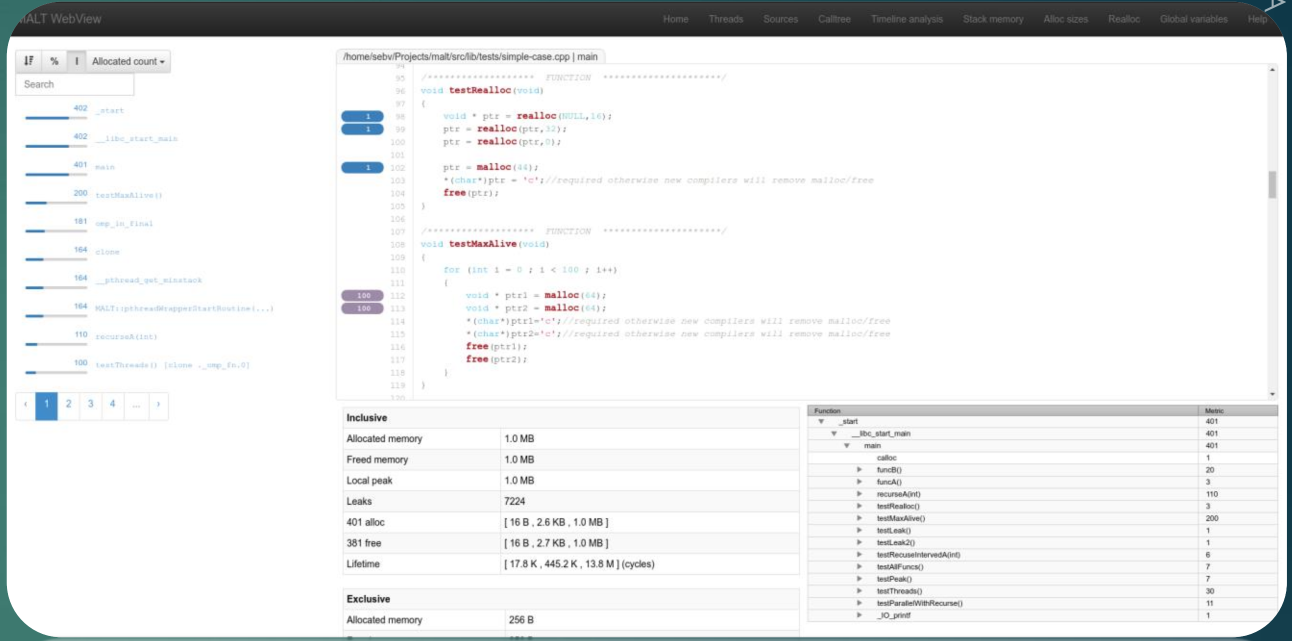
- Working on laptop Core2Duo
- OS: Gentoo / Debian

Mid-Feb

- First run as a POC
- Basic backend + draft GUI

March

- A “real” test at ViHPS
- With a Phd. student stucked with his app



402	_start
402	__libc_start_main
401	main
200	testMaxAlive()
181	omp_in_final
164	clone
164	__pthread_get_minstack
164	MALT::pthreadWrapperStartRoutine(...)
110	recurseA(int)
100	testThreads() [clone ._omp_fn.0]

/home/sebv/Projects/malt/src/lib/tests/simple-case.cpp | main

```
/******************  FUNCTION  ********************/
void testRealloc(void)
{
	void * ptr = realloc(NULL,16);
	ptr = realloc(ptr,32);
	ptr = realloc(ptr,0);

	ptr = malloc(44);
	*(char*)ptr = 'c';//required otherwise new compilers will remove malloc/free
	free(ptr);
}

/******************  FUNCTION  ********************/
void testMaxAlive(void)
{
	for (int i = 0 ; i < 100 ; i++)
	{
		void * ptr1 = malloc(64);
		void * ptr2 = malloc(64);
		*(char*)ptr1='c';//required otherwise new compilers will remove malloc/free
		*(char*)ptr2='c';//required otherwise new compilers will remove malloc/free
		free(ptr1);
		free(ptr2);
	}
}
```

Inclusive	
Allocated memory	1.0 MB
Freed memory	1.0 MB
Local peak	1.0 MB
Leaks	7224
401 alloc	[16 B , 2.6 KB , 1.0 MB]
381 free	[16 B , 2.7 KB , 1.0 MB]
Lifetime	[17.8 K , 445.2 K , 13.8 M] (cycles)

Exclusive	
Allocated memory	256 B

Function	Metric
▼ _start	401
▼ __libc_start_main	401
▼ main	401
calloc	1
▶ funcB()	20
▶ funcA()	3
▶ recurseA(int)	110
▶ testRealloc()	3
▶ testMaxAlive()	200
▶ testLeak()	1
▶ testLeak2()	1
▶ testRecuseIntervedA(int)	6
▶ testAllFuncs()	7
▶ testPeak()	7
▶ testThreads()	30
▶ testParallelWithRecurse()	11
▶ _IO_printf	1

MALT

- The PhD. **student aside me**:
  - **Stuck** with his code **failing on cluster** due to **out-of-memory**
  - **"I develop a tool for this, maybe we can test ?"**

- **"I'm not sure because I started 3 month ago"**
  - Never tested **MPI**
  - Biggest (~uniq) code was 1000 lines, C.
  - His one was **256 tasks**, **~30000 Fortran**, **Ifort**, **Intel MPI**
  - Cluster OS: **Redhat** (I tested Gentoo)

MALT

Install

- Success

Unit test fail

- Due to redhat old feature
- Fixed in **5 minutes**

First run

- In Debug mode + assert
- One too strict assert [comment]
- One fatal error
- Both fix **10 minutes**

He **forgot global variable**

Biggest than what **he thought**

12-20 GB

MALT

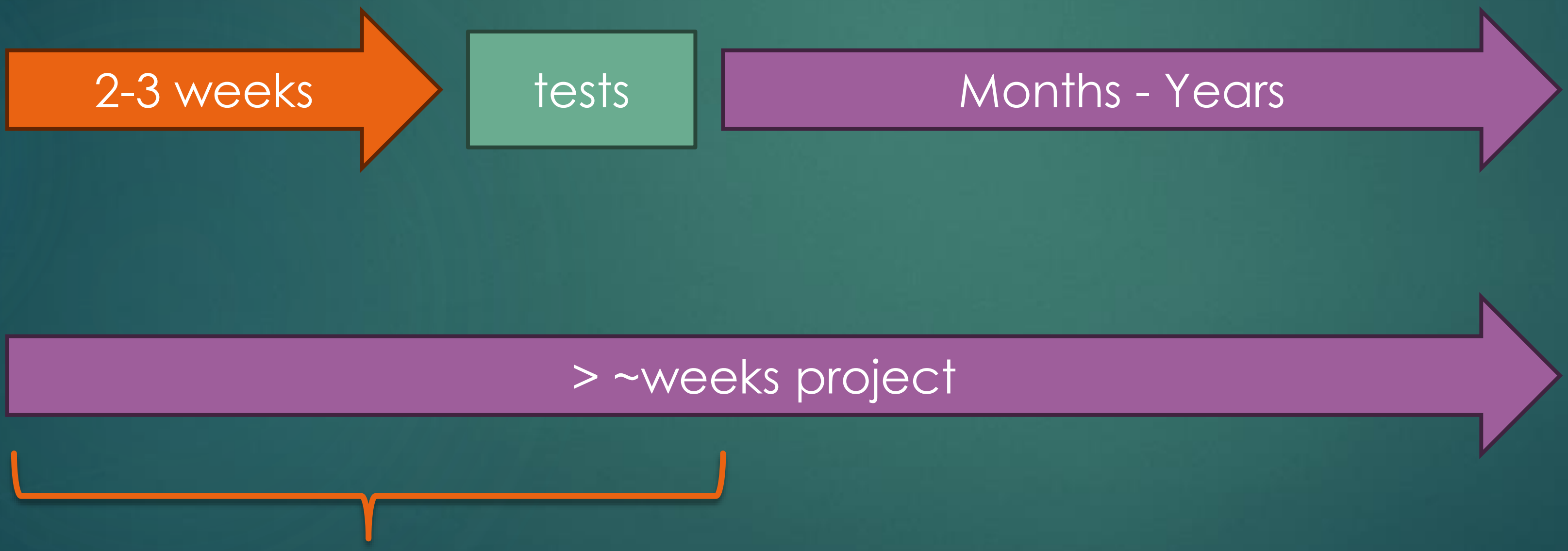
- Total dev : **~8 month at the lab**
- **1.5 year latter** without touching
  - Run at CERN on lhcb-daqpipe (**30000 C++ lines**) => Success
- Run on **Lhcb framework** (**~2 million lines** + XXXX libraries)
  - Backend success
  - NodeJS not loading Json file larger than 600MB => mine 690MB ☹
  - **~1.5 week data reshaping** and **recursive call** stack **compactation**
  - File 250MB => display OK

No fear to **quickly expose to real app** !

Conclusion

My time rules

- Of course, **depend on** language / objectives / complexity



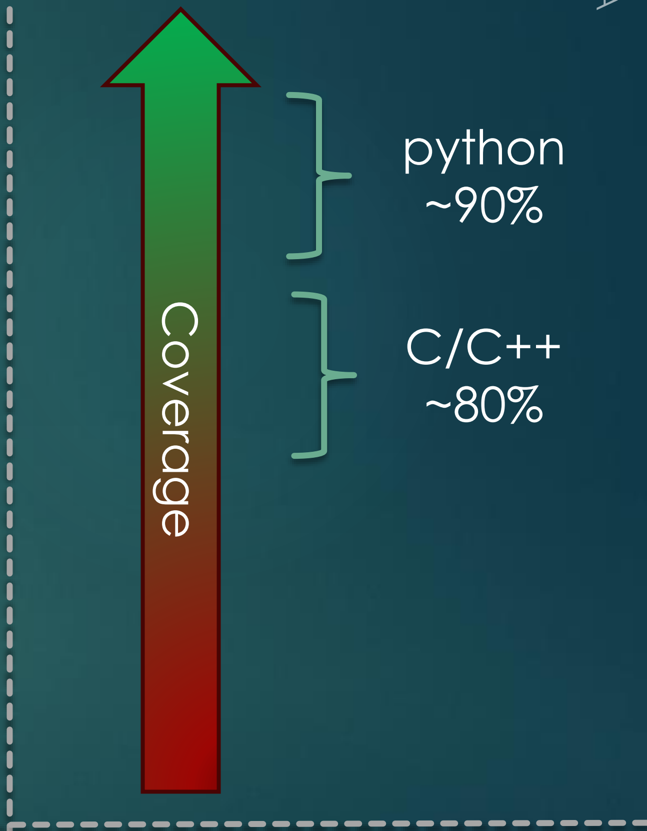
2-3 weeks

tests

Months - Years

> ~weeks project

~**1 or 2 months** slower



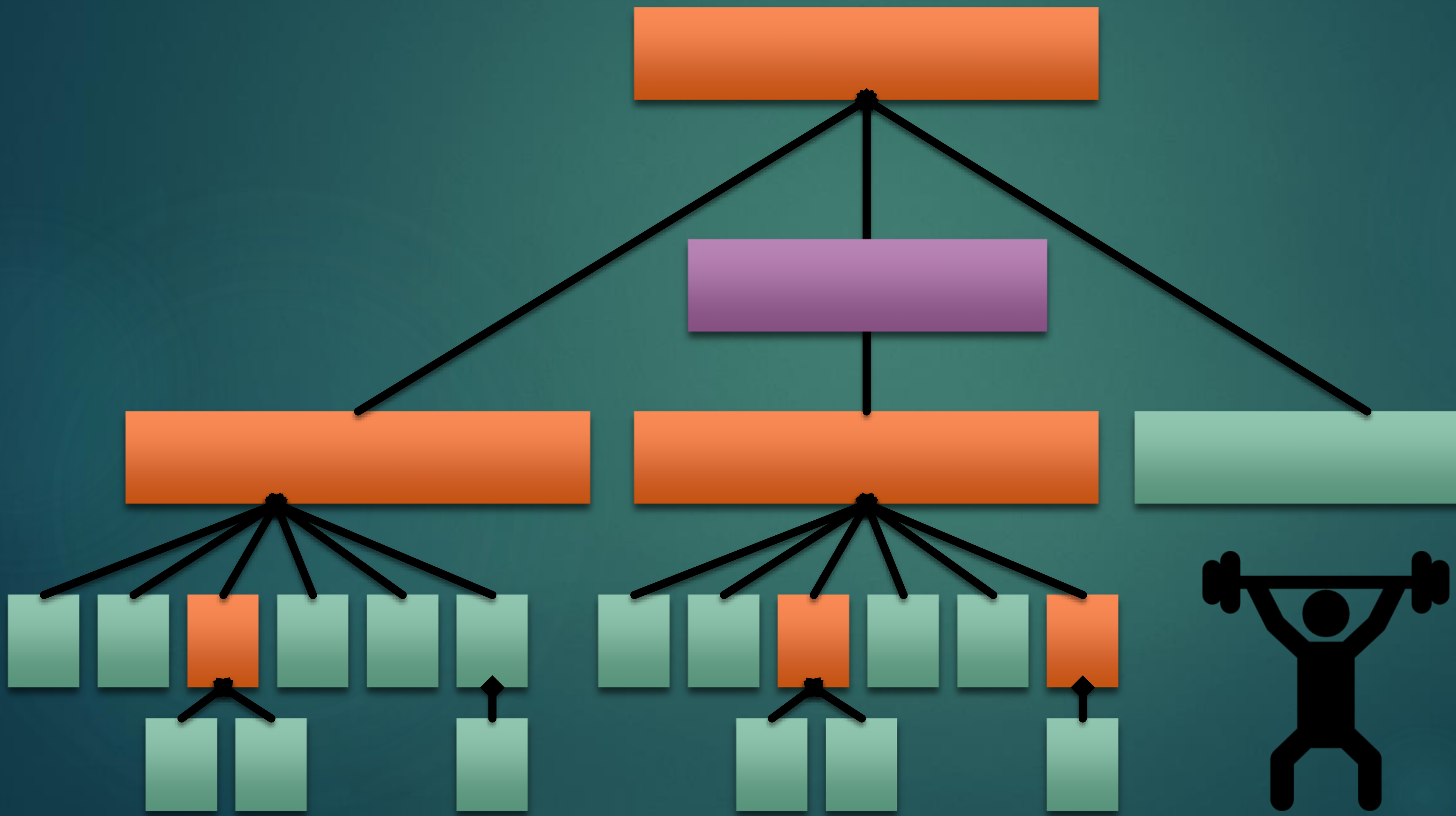
Coverage

python
~90%

C/C++
~80%

A least
1 test per
function / class

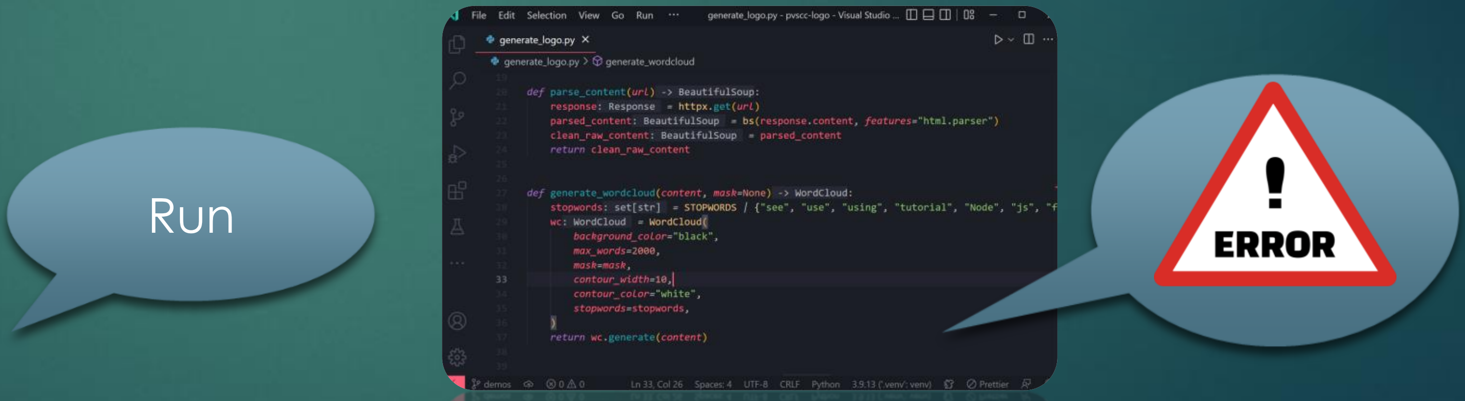
Learning in an exiting software



https://pngimg.com/image/61708

Son code : **votre meilleur ami**

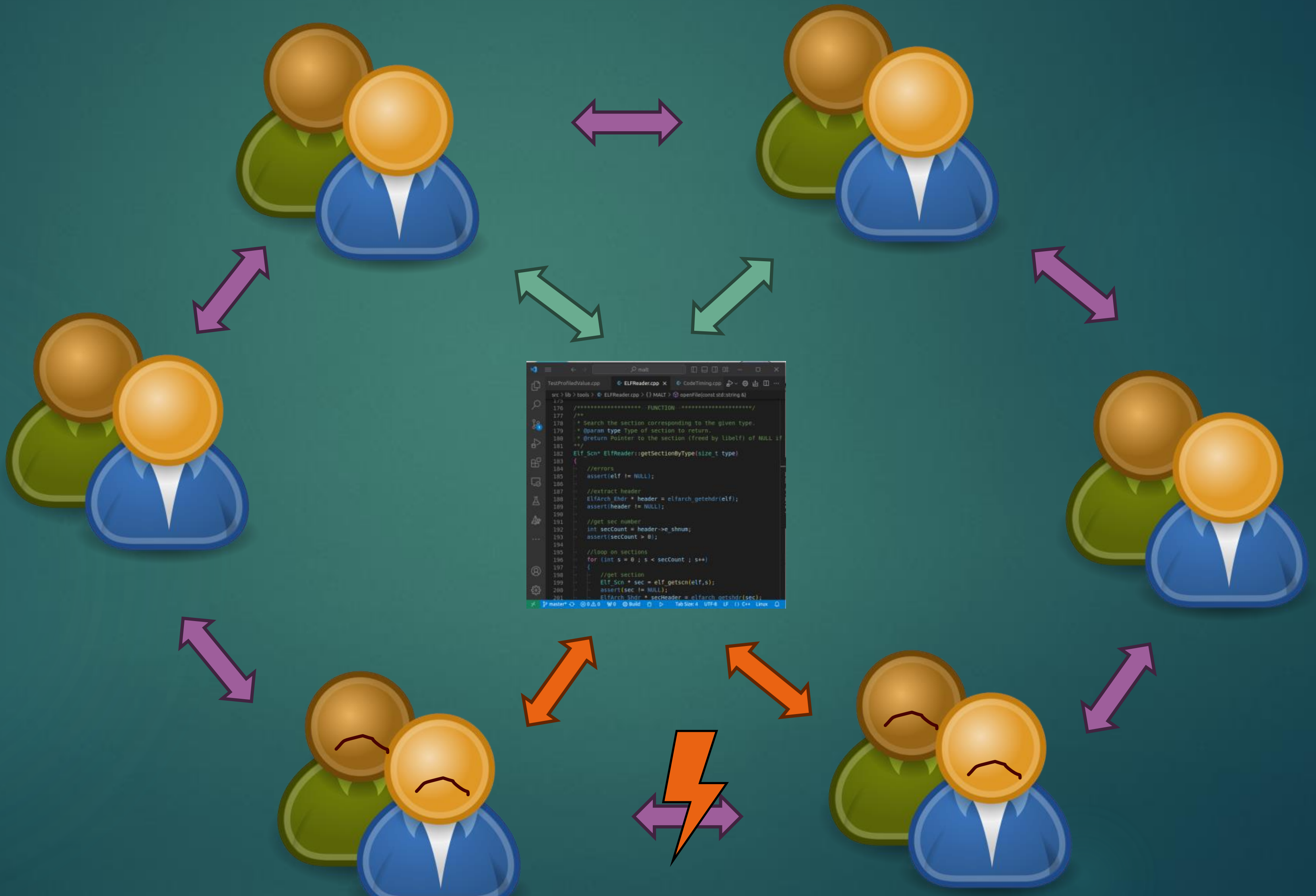
- C'est **avec lui** que vous allez **passer la majorité de vos journées**
- Vous allez **lui parler**
- Il va vous **répondre**……



```python
def parse_content(url) -> BeautifulSoup:
    response: Response  = httpx.get(url)
    parsed_content: BeautifulSoup  = bs(response.content, features="html.parser")
    clean_raw_content: BeautifulSoup  = parsed_content
    return clean_raw_content


def generate_wordcloud(content, mask=None) -> WordCloud:
    stopwords: set[str]  = STOPWORDS | {"see", "use", "using", "tutorial", "Node", "js", "f
    wc: WordCloud  = WordCloud(
        background_color="black",
        max_words=2000,
        mask=mask,
        contour_width=10,
        contour_color="white",
        stopwords=stopwords,
    )
    return wc.generate(content)
```

https://uxwing.com/error-icon/

A bridge in middle of the team



malt
TestProfiledValue.cpp
ELFReader.cpp
CodeTiming.cpp
src > lib > tools > ELFReader.cpp > {} MALT > openFile(const std::string &)
master* ⊗ 0 ⚠ 0 ⓦ 0 Build Tab Size: 4 UTF-8 LF {} C++ Linux

```
/********************  FUNCTION  *********************/
/**
 * Search the section corresponding to the given type.
 * @param type Type of section to return.
 * @return Pointer to the section (freed by libelf) of NULL if
**/
Elf_Scn* ElfReader::getSectionByType(size_t type)
{
	//errors
	assert(elf != NULL);

	//extract header
	ElfArch_Ehdr * header = elfarch_getehdr(elf);
	assert(header != NULL);

	//get sec number
	int secCount = header->e_shnum;
	assert(secCount > 0);

	//loop on sections
	for (int s = 0 ; s < secCount ; s++)
	{
		//get section
		Elf_Scn * sec = elf_getscn(elf,s);
		assert(sec != NULL);
		ElfArch_Shdr * secHeader = elfarch_getshdr(sec);
```

Conclusion

- Always compare with **real world engineering**
- We **tend** to **think** because it is **virtual** it **cost nothing**
  - That's **absolutely wrong** on **long term**
- In research we **want to explore** algos
  - We need to **change the code many times**
  - Hard if we **lose months on debugging**
- There is a **human aspect**
  - the more interesting part for me
  - **In research** => **we let code** to the **next guy** (due to short contracts) !

Be patient,
look the dragon in the eyes

Thanks