Test Driven Development philosophy, feedback in HPC

SÉBASTIEN VALAT – INRIA / GERNOBLE – 24/06/2022

Disclamer

- I see unit testing as a **(philosophical) path**
- We **cannot apply** all **in on week**…
- **You** might also **not agree** with **everything**
- **Please be critic**

Be patient,
look the dragon in the eyes

My source of thinking

- One book on **TDD, conference** video, **research papers**
- But **mostly my own** (home/PhD/post-doc/engineering) **work**
  - I hardly unit test since **12 years**
  - 3 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
- Coverage starting from **43%** to **93%**

Plan

1. A little bit of philosophy & motivation
2. Thinking about testing methods

3. My own experience, feelings
4. Unit test and agile methods

5. Timings on 2 examples

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/

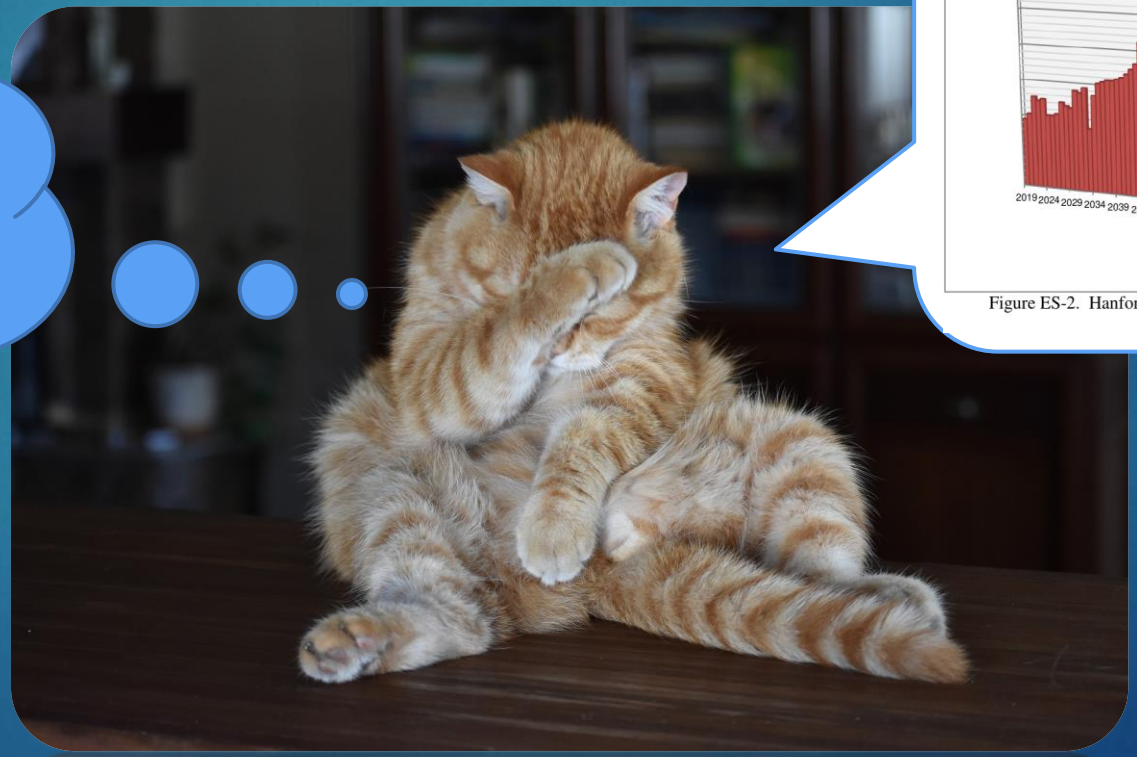
What's inside now

- **No inventory** of the **mixture**
  - Acid
  - Little bit of Actinides
  - …

http://large.stanford.edu/courses/2011/ph241/eason2/

Today: that's **technical debt**

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



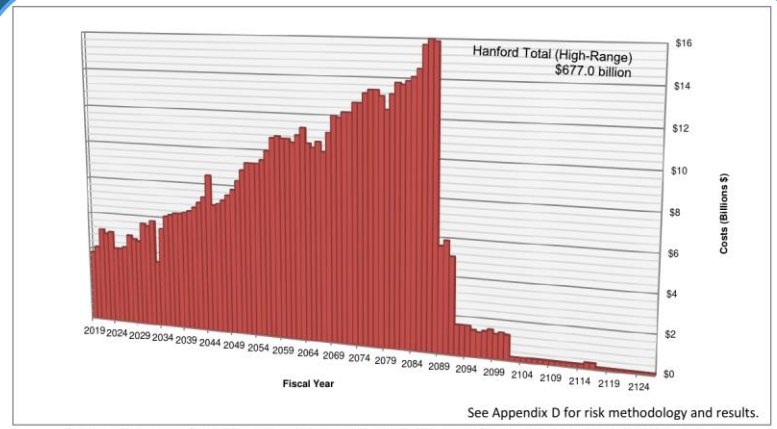


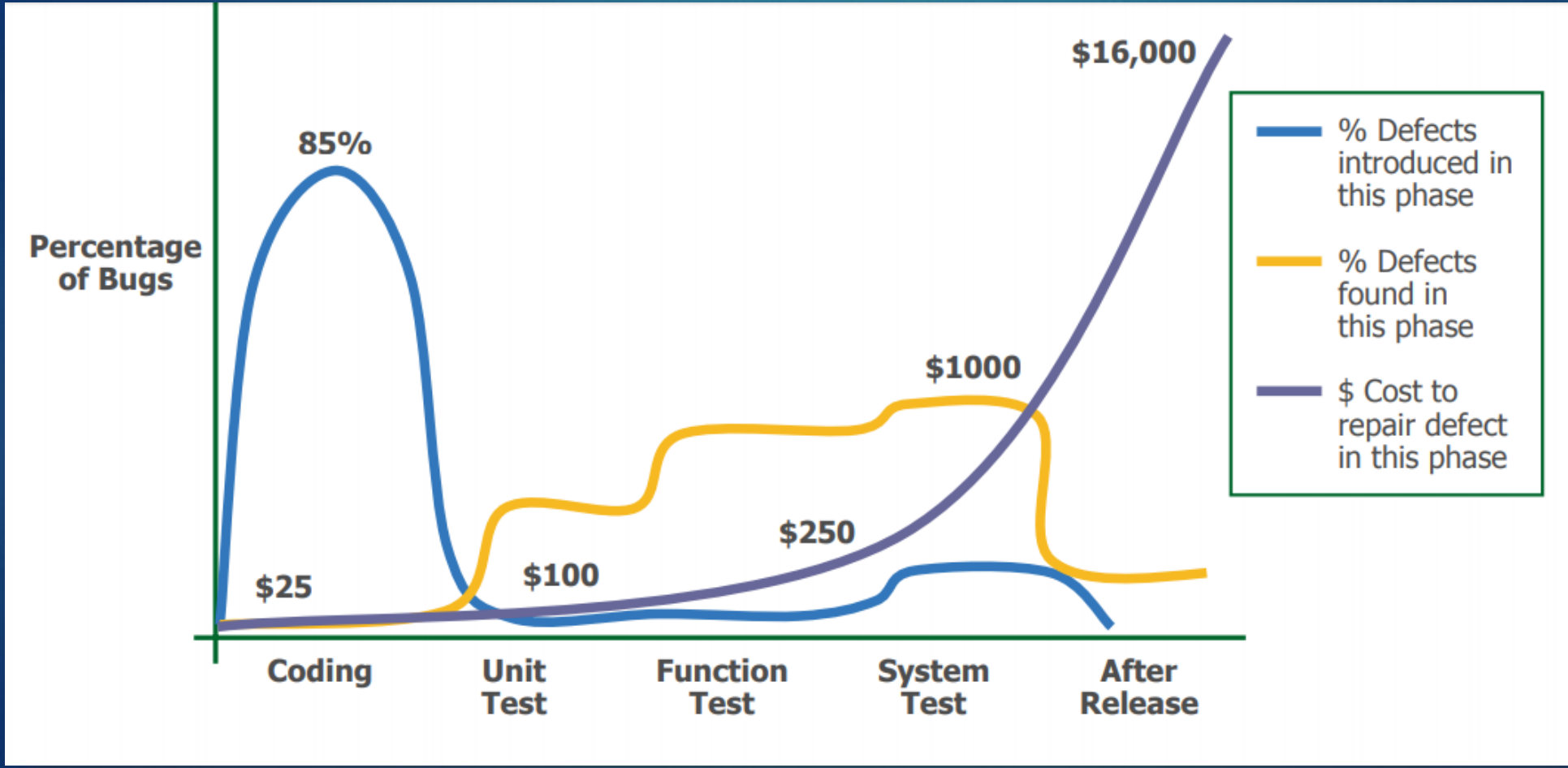
Figure ES-2. Hanford Site Remaining Estimated Cleanup Costs (High-Range) by Fiscal Year (includes both RL and ORP).

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



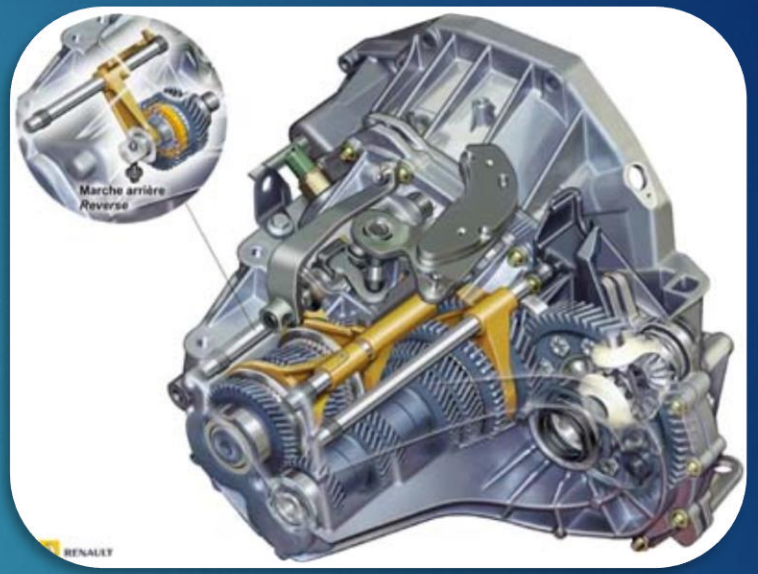
Source: Applied Software Measurement, Capers Jones, 1996

2011 - ref

675 compa.
35 gov/mili.
13500 proj.
24 countr.

Thinking about testing

Lets think you are a car engineer



http://www.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**

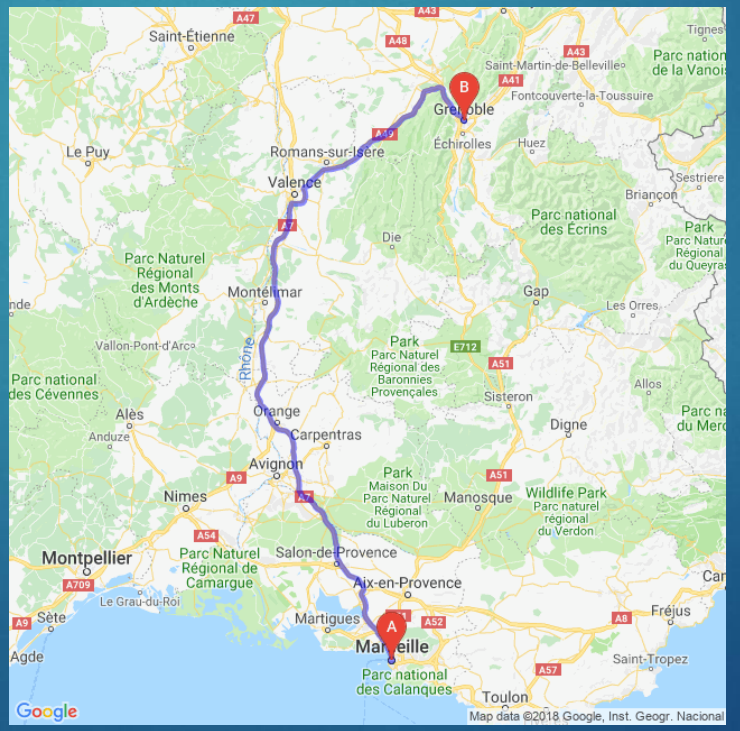
You make no test…

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

- **Would you by ?**

Method 1 : manual test

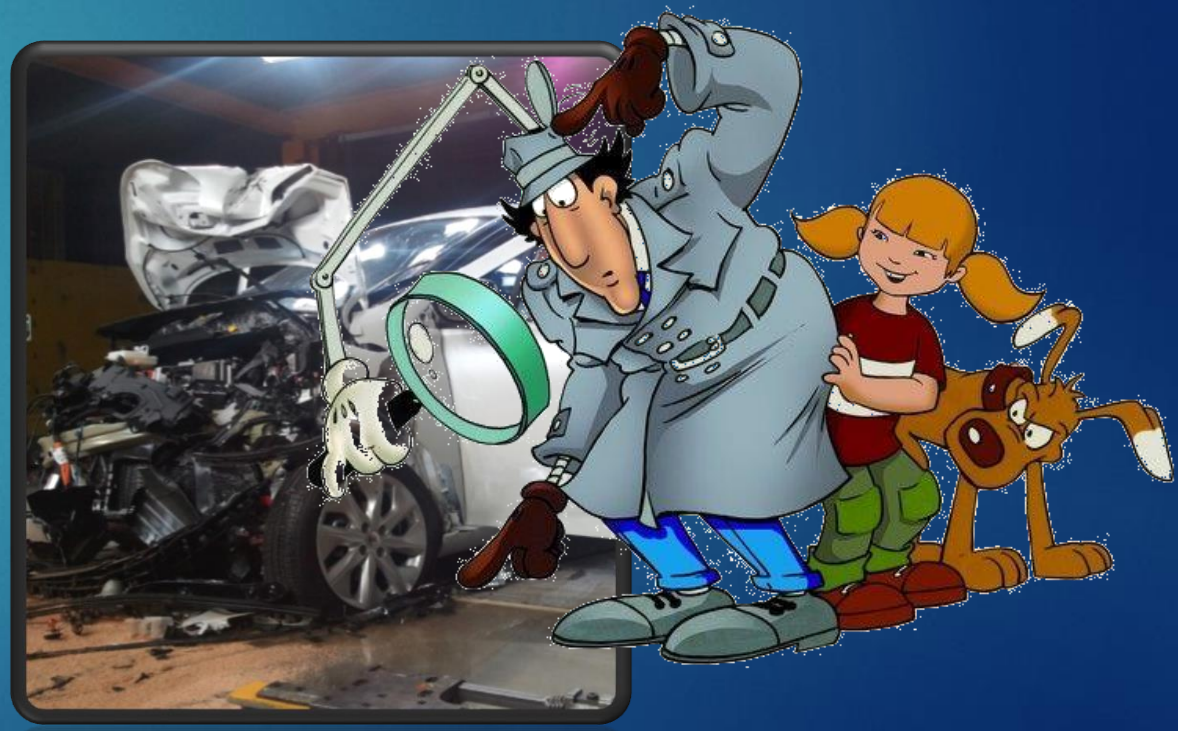
- Way to test a **new gear** we added
- Make a Grenoble – Marseille



Method 2 : **integration** tests

- You **build** a **prototype car** and make **a crash tests**
- **Every time** you **change a gear shape** in the gear box

http://www.thedetroitbureau.com/wp-content/uploads/2016/05/IIHS-Camaro-Crash-Test.jpg



https://www.automobile-propre.com/crash-test-renault-zoe-securite/
http://maguy69.m.a.pic.centerblog.net/o/969011b4.jpg

Method 3 : **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$ pytest Particle.py
========================================= test session starts ==========================================
platform linux -- Python 3.9.7, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /home/sebv/2022-01-unit-test
collected 2 items

Particle.py ..                                                                                  [100%]

========================================== 2 passed in 0.02s ===========================================
```

Run example - failure


```
sebv@sebv6:~/2022-01-unit-test$ pytest Particle.py
========================================= test session starts ==========================================
platform linux -- Python 3.9.7, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /home/sebv/2022-01-unit-test
collected 2 items

Particle.py F.                                                                              [100%]

================================================ FAILURES ================================================
_____________________________________ TestParticle.test_collide ______________________________________

self = <Particle.TestParticle testMethod=test_collide>

    def test_collide(self):
        particle1 = Particle(-0.5, 1.5)
        particle2 = Particle( 0.5, -1.5)
        collide = Physics.elastic_collide(particle1, particle2, 1)
        self.assertTrue(collide)
>       self.assertEqual(-1.5, particle1.get_vx())
E       AssertionError: -1.5 != -3.0

Particle.py:31: AssertionError
======================================= short test summary info ========================================
FAILED Particle.py::TestParticle::test_collide - AssertionError: -1.5 != -3.0
====================================== 1 failed, 1 passed in 0.08s =====================================
```

A realistic case


```
      Start 28: TestWorker
28/34 Test #28: TestWorker ......................   Passed    0.11 sec
      Start 29: TestWorkerManager
29/34 Test #29: TestWorkerManager ...............   Passed    0.16 sec
      Start 30: TestTaskIO
30/34 Test #30: TestTaskIO ......................   Passed    0.11 sec
      Start 31: TestTaskScheduler
31/34 Test #31: TestTaskScheduler ...............   Passed    0.11 sec
      Start 32: TestIORanges
32/34 Test #32: TestIORanges ....................   Passed    0.11 sec
      Start 33: TestTaskRunner
33/34 Test #33: TestTaskRunner ..................   Passed    0.18 sec
      Start 34: TestClientServer
34/34 Test #34: TestClientServer ................   Passed    1.85 sec

100% tests passed, 0 tests failed out of 34

Total Test time (real) =   9.42 sec
sebv@sebv6:~/Projects/iocatcher/build$
exit
```

TDD: Test Driven Development

- The **strict** approach:
  - You **write the tests**
  - **Implement** until it valides the tests
- More **realistic** approach:
  - **Implement** a **function / class**
  - Implement the **tests**
  - **Iterate** to **improve** the **implementation** and **API**

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 MPC 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**
  - Complex to maintain : **1 dev fully dedicated** to it (over 15)

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 **migrate the CI env** : ~**5 months consumed to migrate**

versus **1 week** in unit

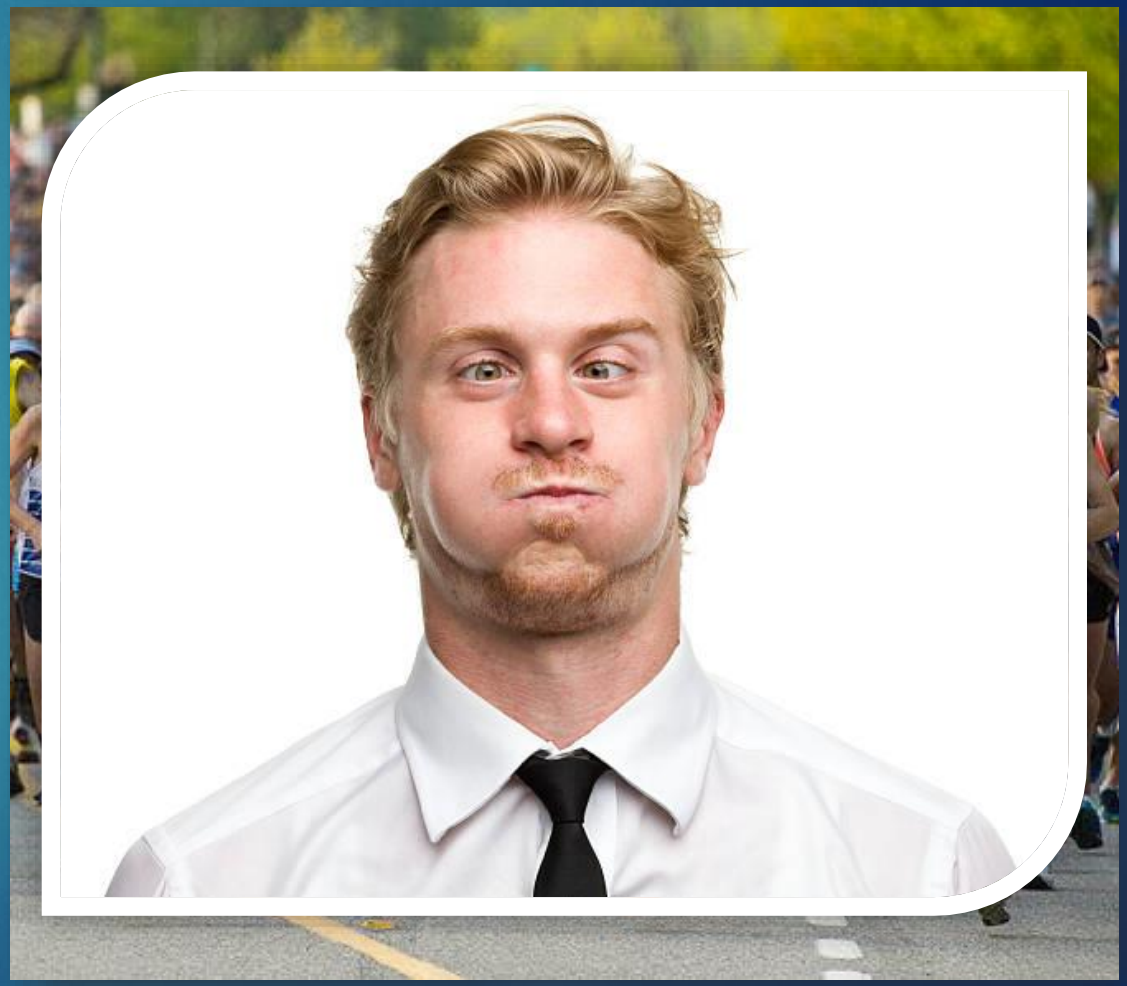
When trying to push in teams…. [unit tests]

- **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

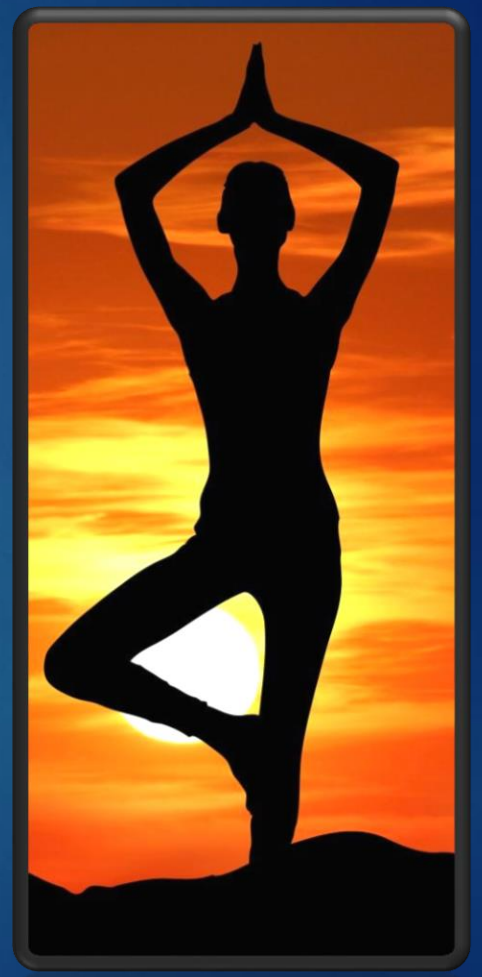


Day to day methodology : **discipline**

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

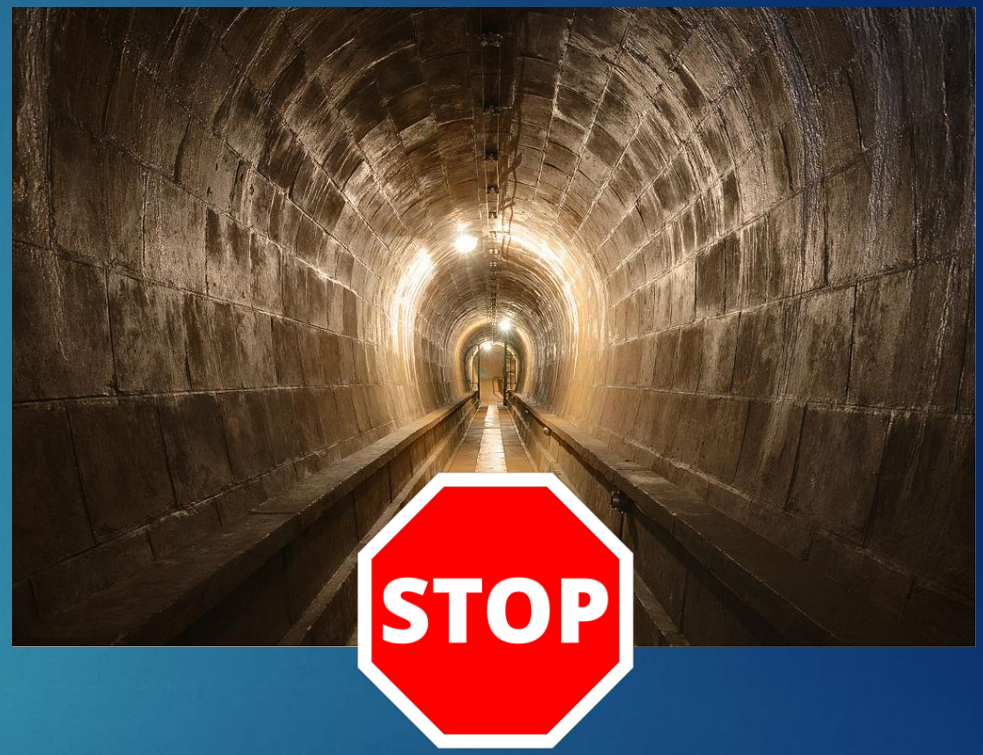
- **You will never do them later**
  - Because your ***design*** *will* ***not permit***
  - Because you will **want to move** to **other stuff**
  - **Nobody** will be **happy** to write unit **tests for ~4 weeks**
  - **Your boss/commercial manager already sold it to clients**….

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



Benefits of unit test

- **That's not only testing**
- 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** !

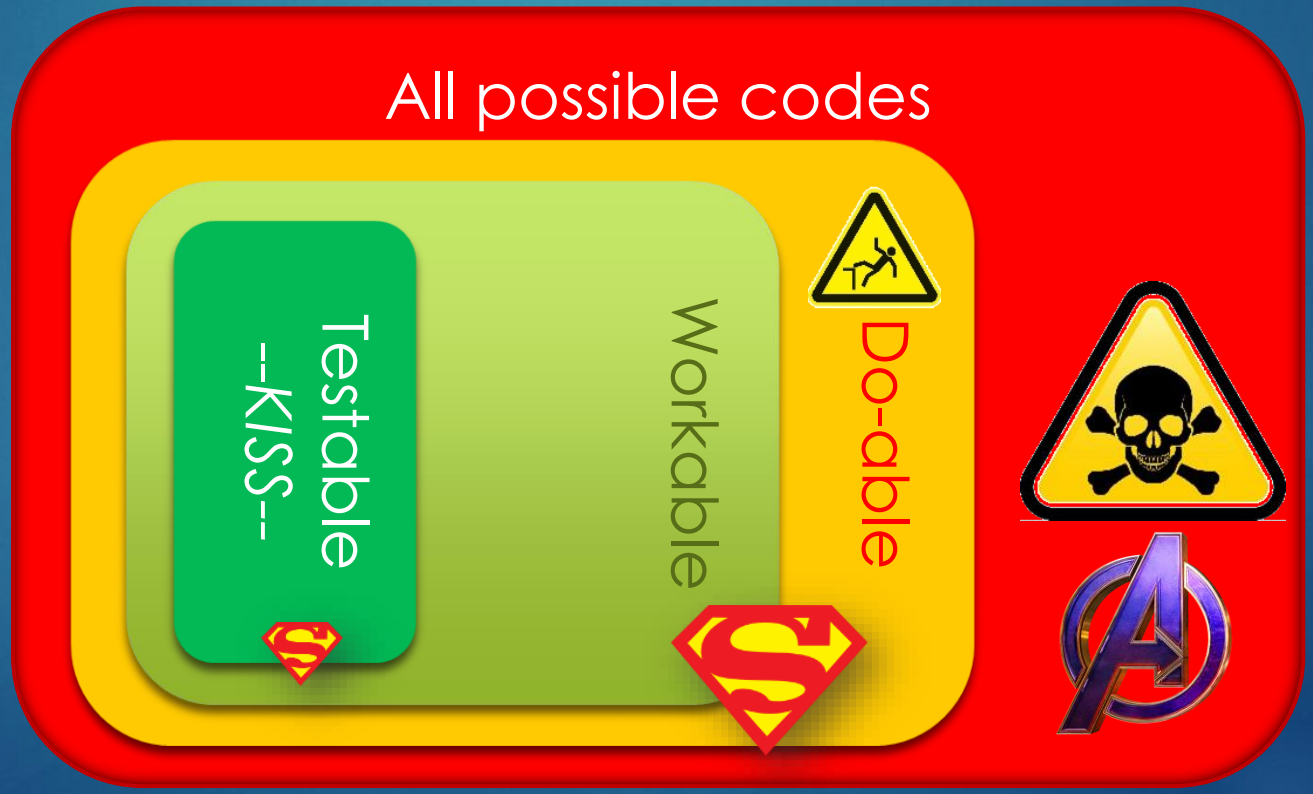
That's a discret **teacher** !

- 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/

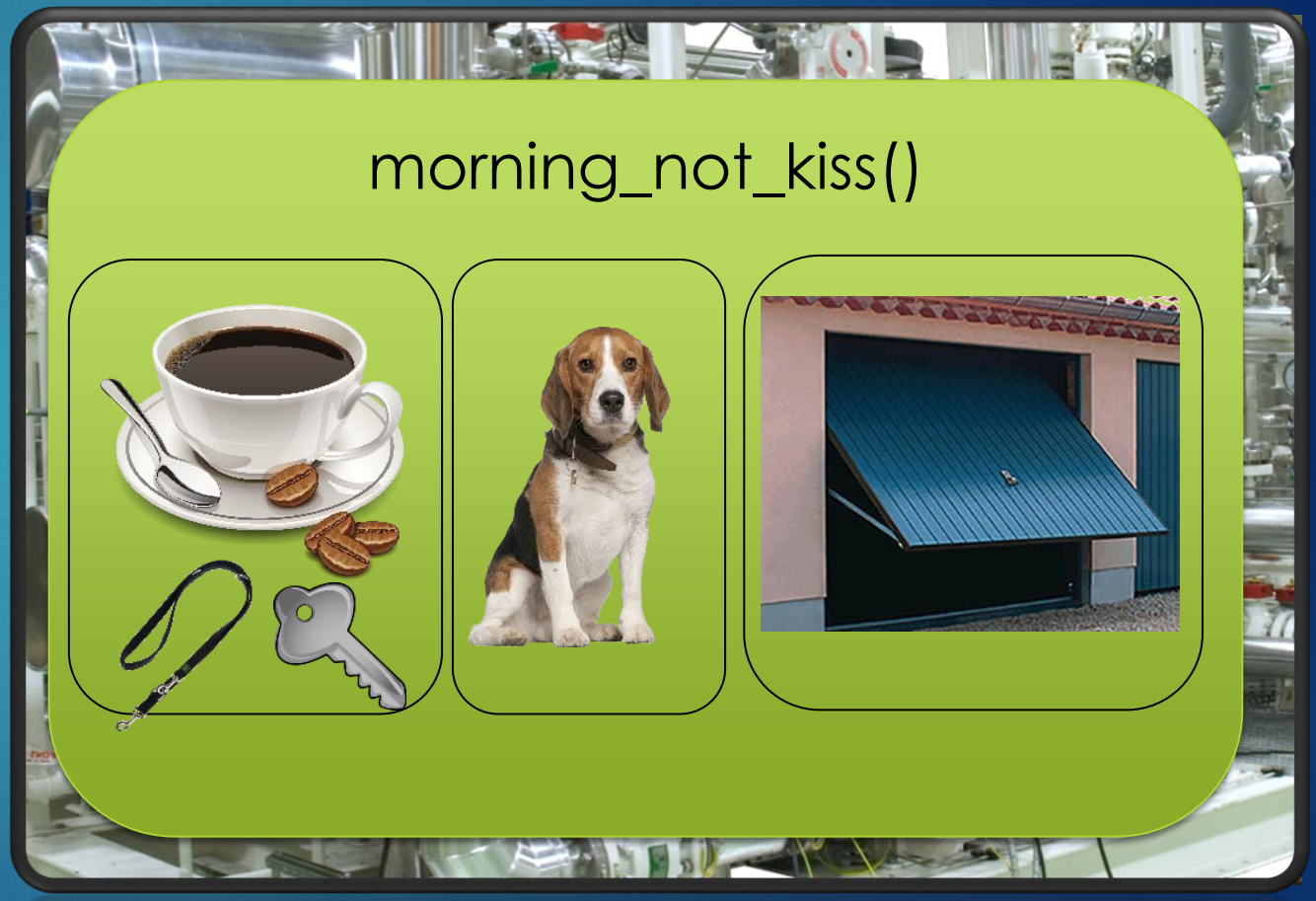
That's also constraints

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



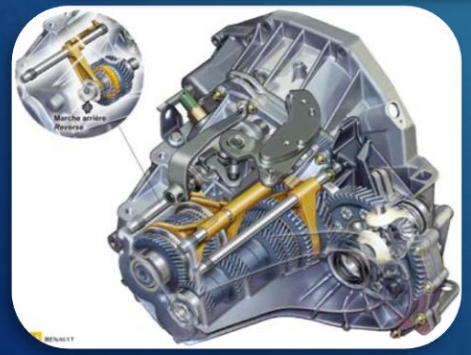
Test a gas machine

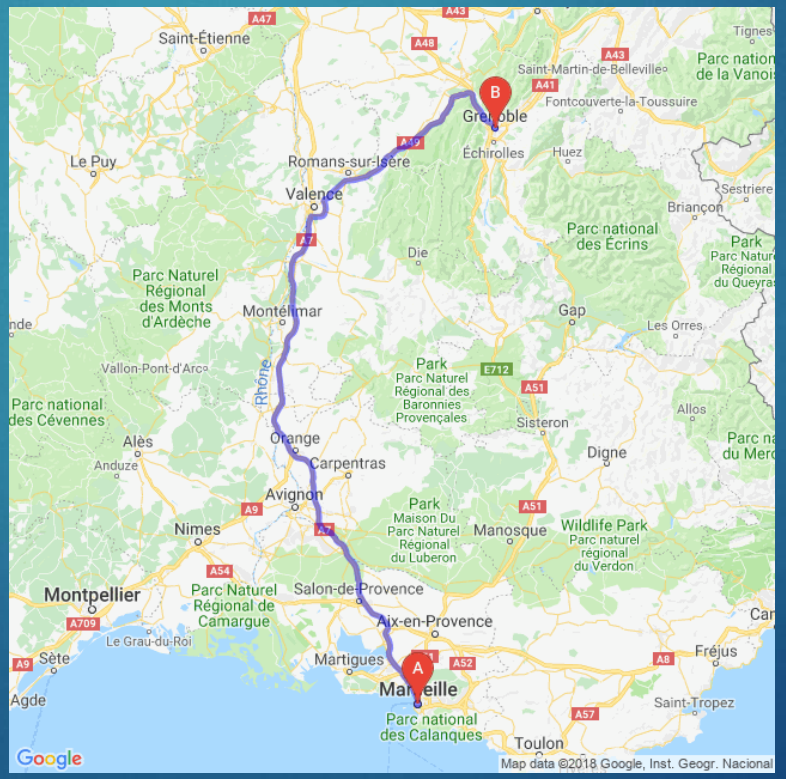
- If your **test** become **too complex**
- You are **certainly** on the **wrong way**
- **Stop**, **think** and **KISS**



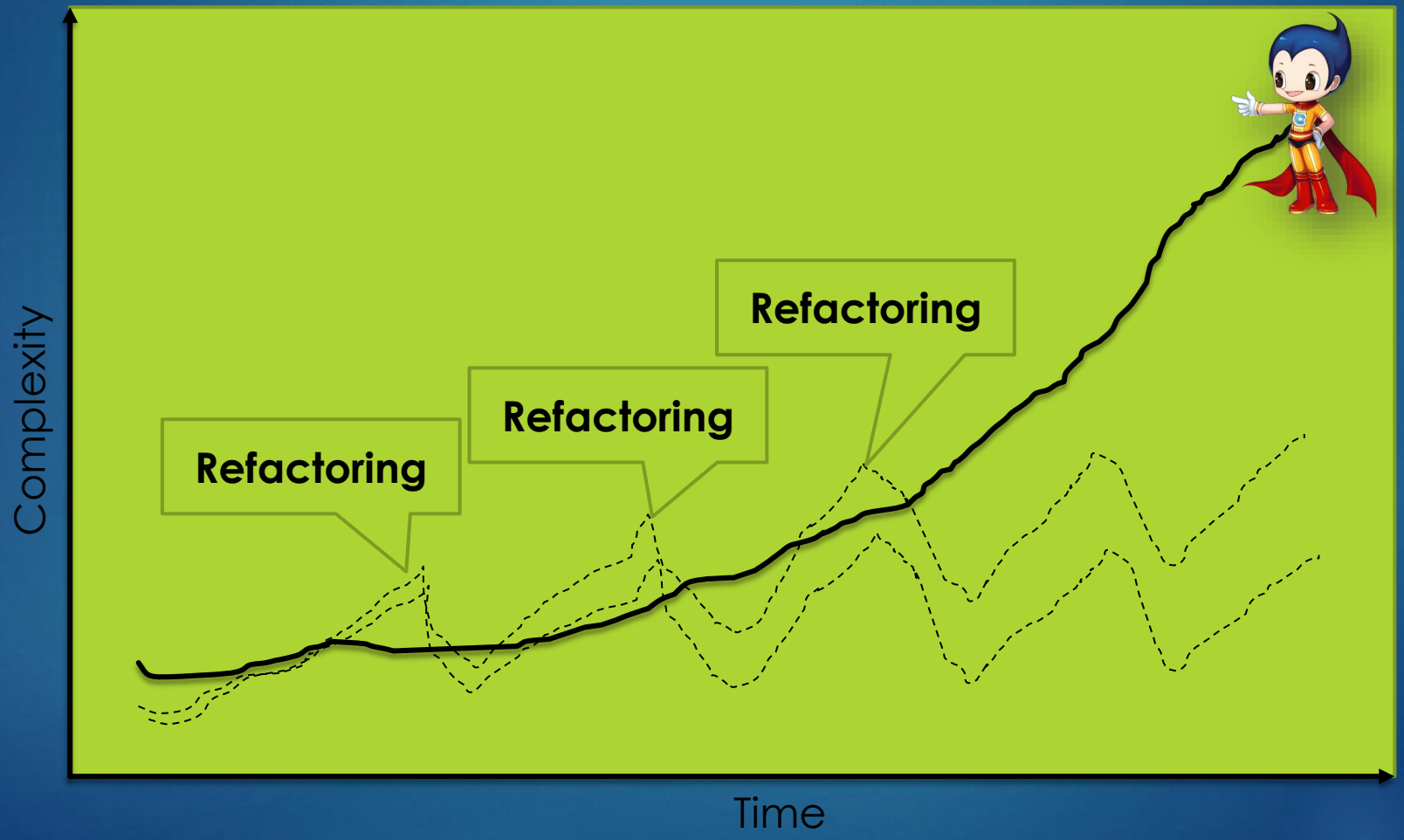
New comer

- How long it take to enter in the code ?
- Make a Grenoble - Marseille





Facing the entropy !

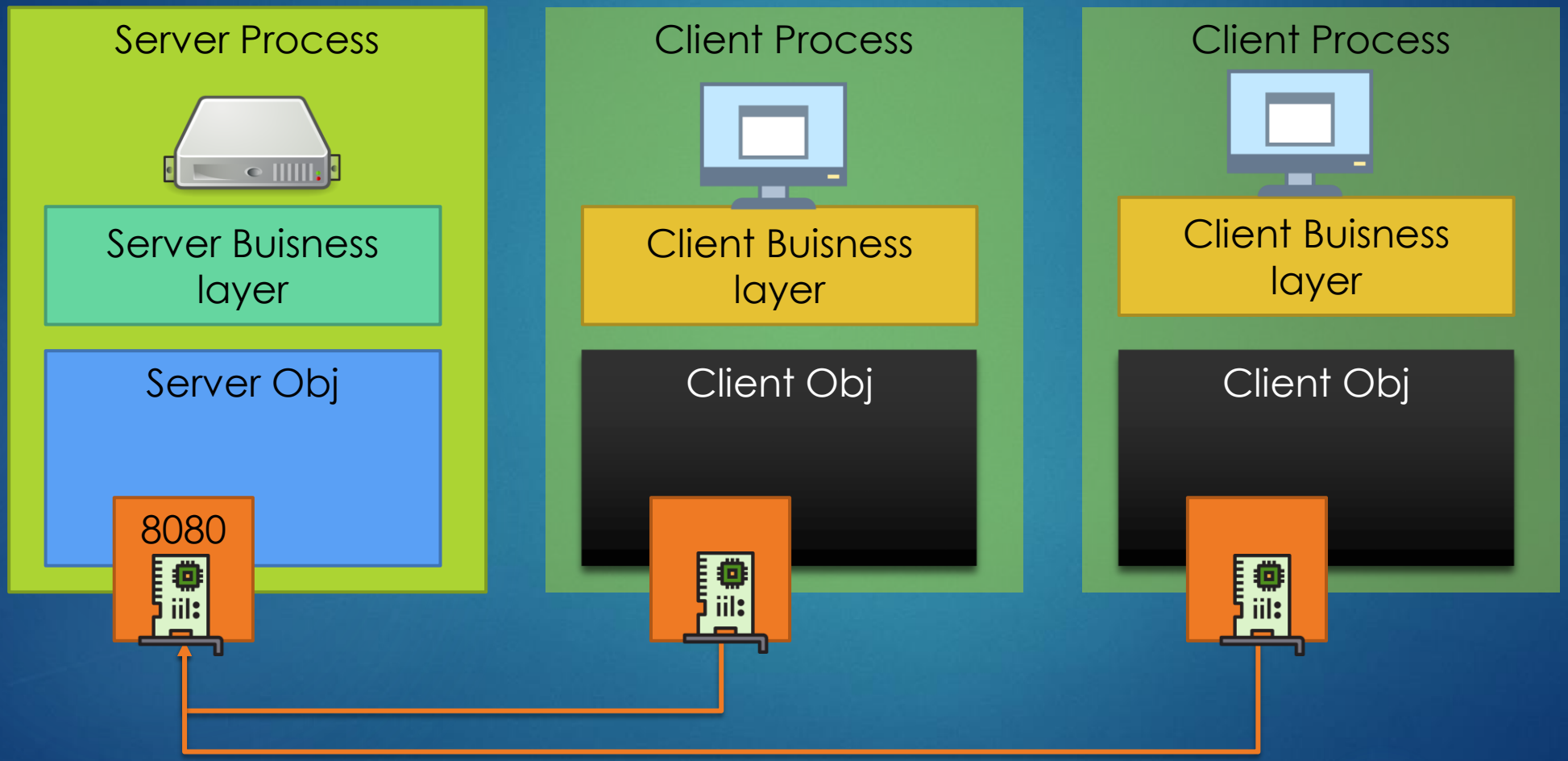


The turtle problem

- Most of **UT introduction courses** use:
  - The **turtle example**
- A **turtle**
  - Has a **position**
  - Can **move forward**
- That's a **too simple example**
  - **Absolutly not representativ** of a real test case

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

Example: client / server

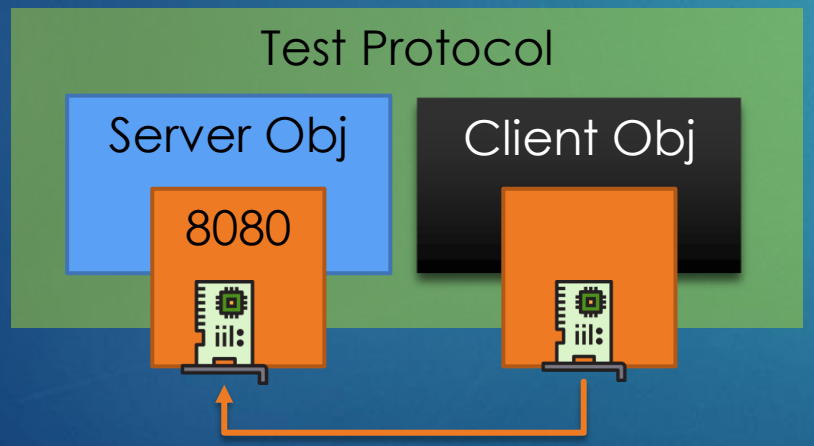


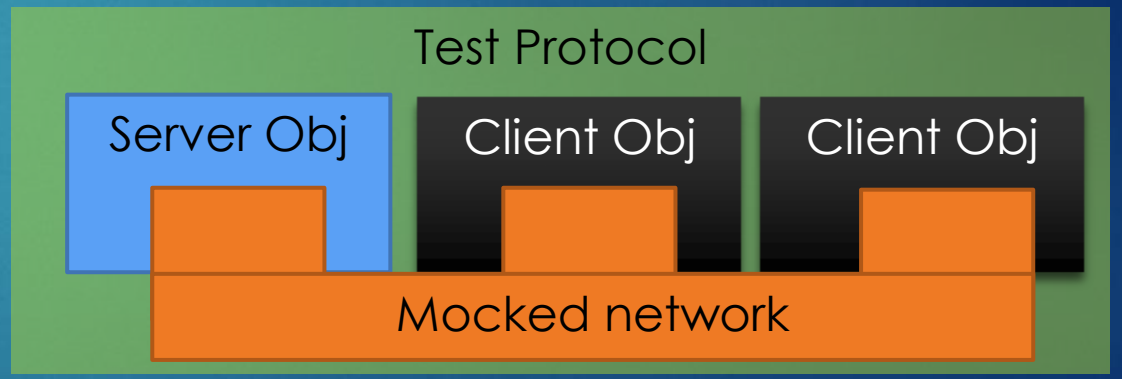
https://kissclipart.com/virtual-machine-icon-png-clipart-computer-icons-co-w8kyyr/
https://www.freeiconspng.com/img/2306

Example: client / server

Server Buisness layer

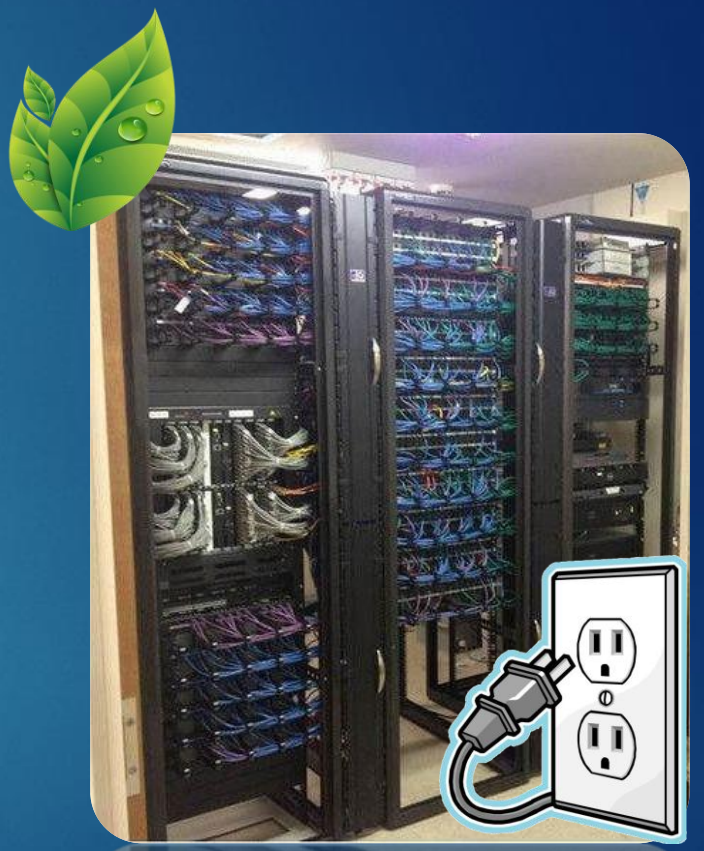
Client Buisness layer





Ecology argument

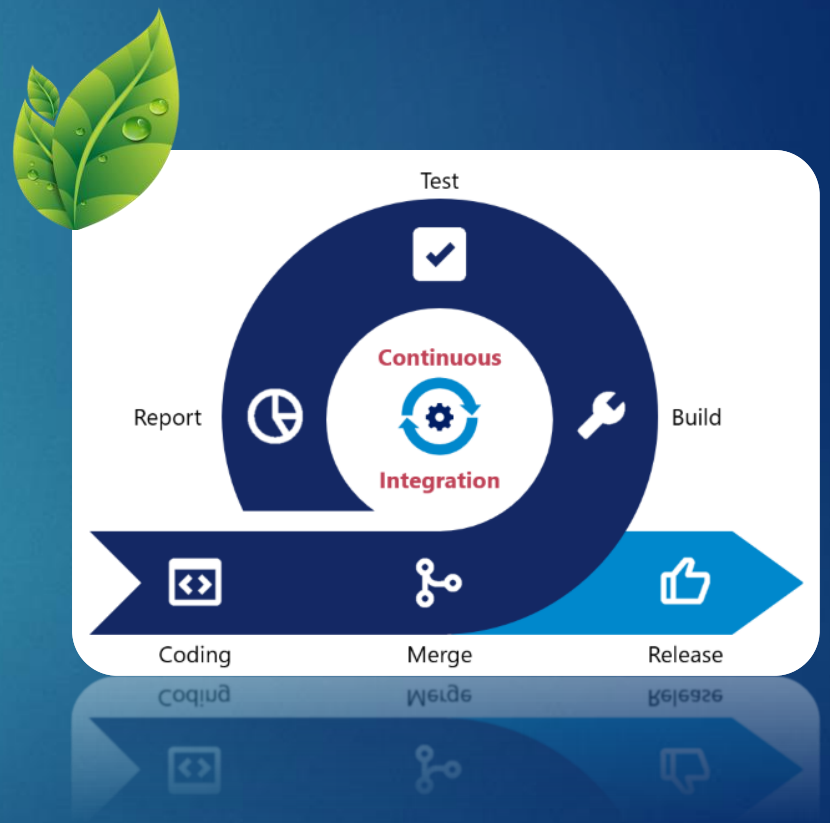
- You can make the **whole dev localy** on your **laptop**
- **No need** of **a large dev cluster**
- **Once done** and validated with unit tests:
  - Make real **test on cluster**
  - **Once a week** or two weeks
- Not anymore per team cluster
- **Less debugging at scale** !



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

Ecology argument - 2

- CI cycle take **~ a minute**
- Intead of **45 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

Keep the knowledge !

- The hard **corner cases** are **encoded in the tests**
- Very **usefull** in case of **rewriting a V2**
- Also to **translate** in another languages
- 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

Summary of interest

- Code design
- Possibility to refactor
- Team dynamic
- Inclusion
- Internal component spec
- Validating the code

About unit tests and agile methods

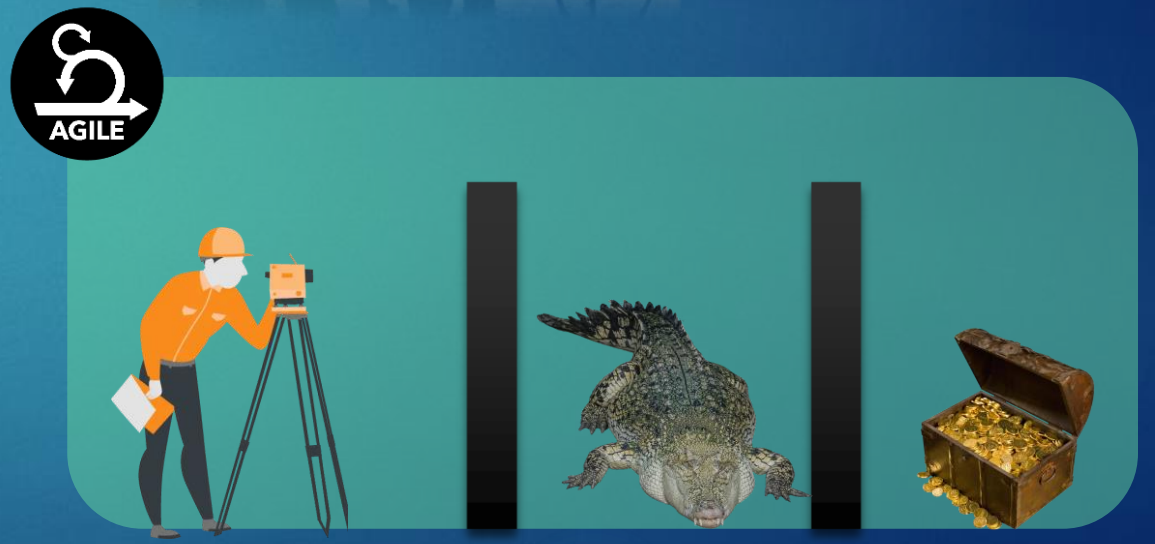
A basement of agile methods

- Agile methods are **built from extrem programming**
  - One feature **is testing**

- That's **a REQUIREMENT** for the method, **not an option**
  - For the **technical validity** of the method
  - For the **dynamic of the team**

Technical aspect

- **V cycle** : evaluate every details
- **Agile**: **just** plan the **next sprint**
- **Refactoring** will be needed
- **Without test**
  - you will **introduce bugs**
  - So you will **pospone the refactoring**
  - You will let you **code under pressure**.



https://parlonssciences.ca/sites/default/files/2019-09/adult_and_children_looking_through_telescope.jpg
https://www.pngmart.com/fr/image/67293
https://www.freepng.fr/png-b6k1n7/
https://www.freepng.fr/png-5jew6x/
http://sebastienbourguignon.com/2015/11/28/agile-et-lautonomie-dans-tout-ca-via-oyomy-infoqfr/agile-icon/

Team dynamic – code review

- A **new feature** requires **changing many parts**
- **Without unit tests**, you need to **patch everything** to **know it is correct**
- Terminate with **large patches**
  - **~2 weeks** of **dev** (or more)
  - **~1 week** of **review**
  - You make **1 or 2 story / sprint / dev** !
  - **Agility is gone** !

https://www.freepng.fr/png-50jeyp/

Team dynamic – code review

- **With unit test**
- **Split** each steps in **small sub patches**
  - Force to **discsuss** the splitting in the team
- One patch take
  - **~1 day** to **dev**
  - **~1 day** to **review**
- You **review** the **global structure** of the patch **not line per line**
  - It **escalate** the **discussion** at a more **design level** **instead** of **deep details**

https://www.flaticon.com/free-icon/boost_2405255?related_id=2405255&origin=search#

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

Timings on 2 examples

COSTS AND EXAMPLES

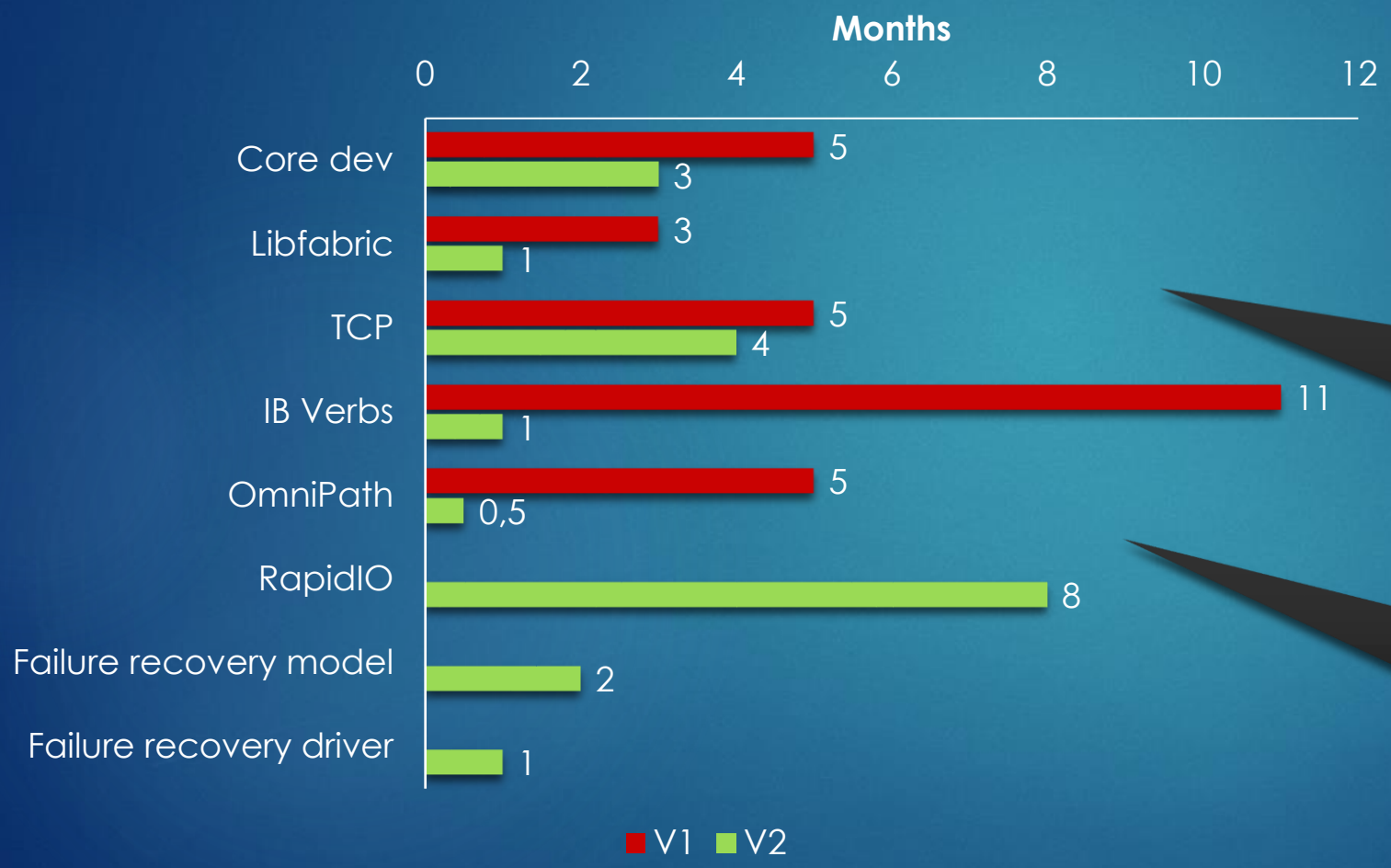
A short example in a previous team

- **First version** (**no unit test**):
  - **~3 - 4 devs** + **2 years**
  - **38000 lines** of code
  - **Loosed** several new **developers** failing to enter in the code (not beginners)
  - Adding feature « open » **: 2 weeks + 2 weeks for patch review**
- **Second version 85% coverage unit test**
  - **1 dev + ~1.5 year** (~6 months half time)
  - **12000 lines** of code
  - **Perf**: small IOs **=> x10**, **1 thread versus ~20**
  - Adding feature « open » => **30 minutes**

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 **~4000**

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**

CERN lhcb-daqpipe V2 IB story

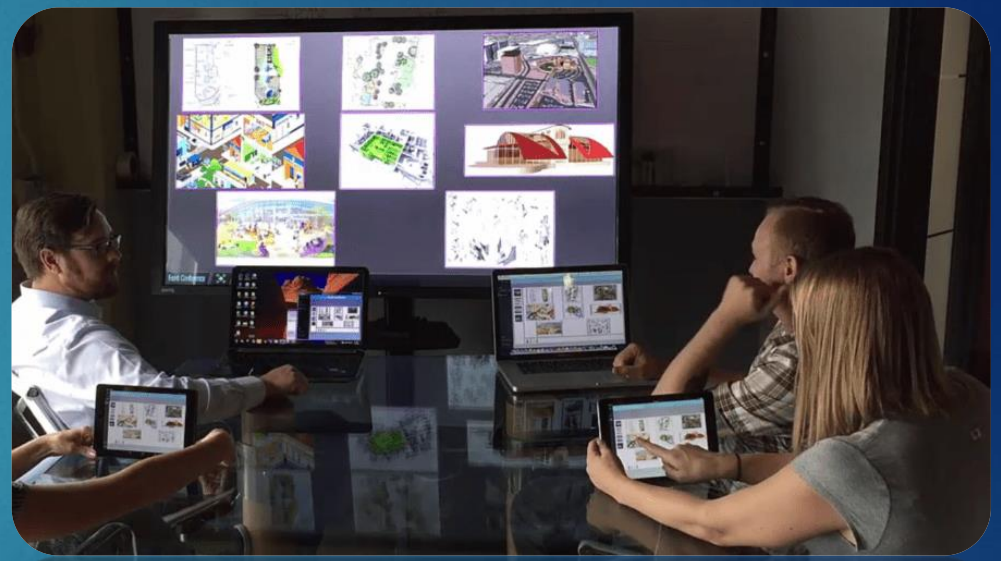
- Polish **PhD. student** coming **1 month for collaboration**
  - **CUDA**/GPU algos for **ITER (1 server)**
  - **No** pre knowledge on **MPI/RDMA** networks
  - **No** pre knowledge on our requirement of code
- **I was vacation last week** when he made tests on more than 2 nodes
- Made **1 month later 2 days of tests** on our 8 nodes
  - To prepare next test on Dell cluster.
- I made a try **2 month latter on 256 nodes**
  - ~perf than MPI aster 1 day of parameter tuning + 1 minor fix.
  - 2 days later => little better than ou MPI driver

Macroni Adventure

- Start dev on V2 => **July**
- Marconi supercomputer test **May**
  - Date and agreement known **2 weeks before**
  - Intel negotiated us **2 days full cluster** before prod
  - That's a **1.7 PFlops** cluster, **1500 server**, **1.3 MWatts**
  - **First real cluster with OPA**
- **Our test bed** was **8 servers**, **target** is **500**

Marconi Adventure

- Present at CINECA
  - **Me + my boss** arrived by plane morning
  - **4 Cineca admins** (~cannot work due to test)
  - **2 INFN people** implied
- Cluster status
  - Still **instabilities on OPA** network
  - **Luster not available**
  - **PBS not working** since 2 days
- **Plug my laptop on projector**, then I work !
  - We didn't pay but I would like to know how much it costs !!!!!!!
  - **Demo effect ?**



Marconi adventure

- First **2 hours** to **find workarounds** because no **PBS + Lustre**
  - **Find adequate hosts** to run on based on **network topology**
- **1 hour to debug** issues pointed by the tests and asserts
- **at 11h**, start to **run up to 32 nodes**
  - Get first nice results with MPI and first pre-try directory with PSM2
- **Afternoon** scaling up to **256 nodes**
  - **One try** asked on **512** but failed

Marcon adventure

- Result **up to 64** where **really fine** for a first run
  - **1-2 hours** or tuning parameters for **every scale**
- **Harder** on **128 to 512** to get best tuning
  - At least thanks to unit tests
  - I was sure it worked at scale because I tested via UT
  - on ….. **my workstation 4 cores** !!!!
- End of day 2, **got all results**
  - Post analysis via simulation showed due to cabling + routing issues we cannot got more **(~10-15 % theorical margins)**

Portability, scalability, dev hardware reduced costs !

Ecology ?

Conclusion

My time rules

- Of course **depend on** language / objectives / complexity
- **2 weeks** of coding
  - Ok without unit test
- Start **for longer**
  - Immediate unit test
- Extend **after 2 weeks**
  - Take 1-2 weeks to **refactor + unit test**
  - **Before continuing** progressing
- For **~1 year project**
  - **1st/2nd month**(s) loosed with "slow" progress

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**
- **I hope I motivated you** to look on UT
- **Even more true** to target **performance**
  - Always **need to change** design to follow architectures
  - Make tons of **performance mistake** we need to fix
- That's a **vital component** of **agile methods**

Do not forget **it is a long road** !
Mine took **~4**+5=9 **years**

BACKUP

Some framework

Language	Test framework	Mocking
Python	unittest	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*