Automation Infrastructures Do’s and Dont’s

Yossi Rosenberg
Gett

About Me: *www*

- **Who** am I?
- **Why** am I here?
- **Why** am I developing automation and not a software developer ?

During this journey we will answer some questions

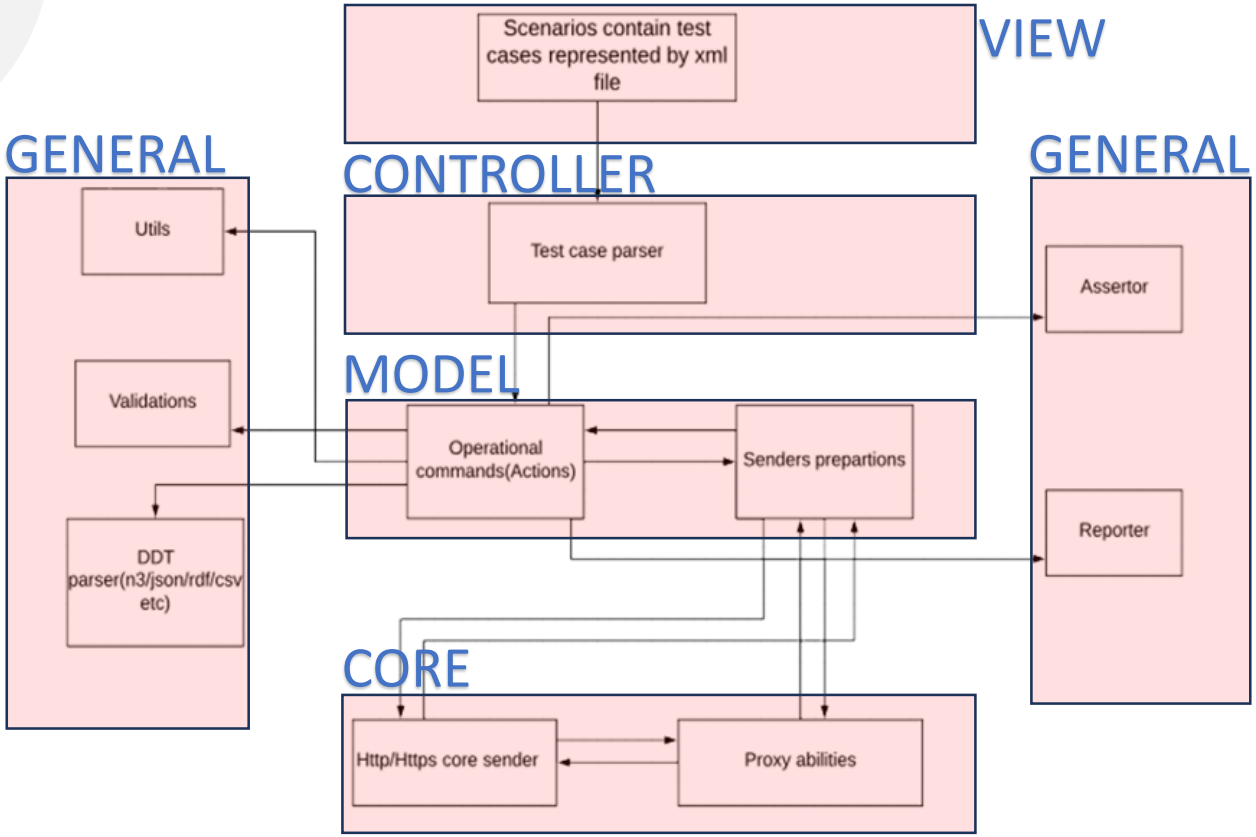
How much time spent using a “bad” infra on development time + maintenance efforts ?

What is the right formula to develop a successful infrastructure ?

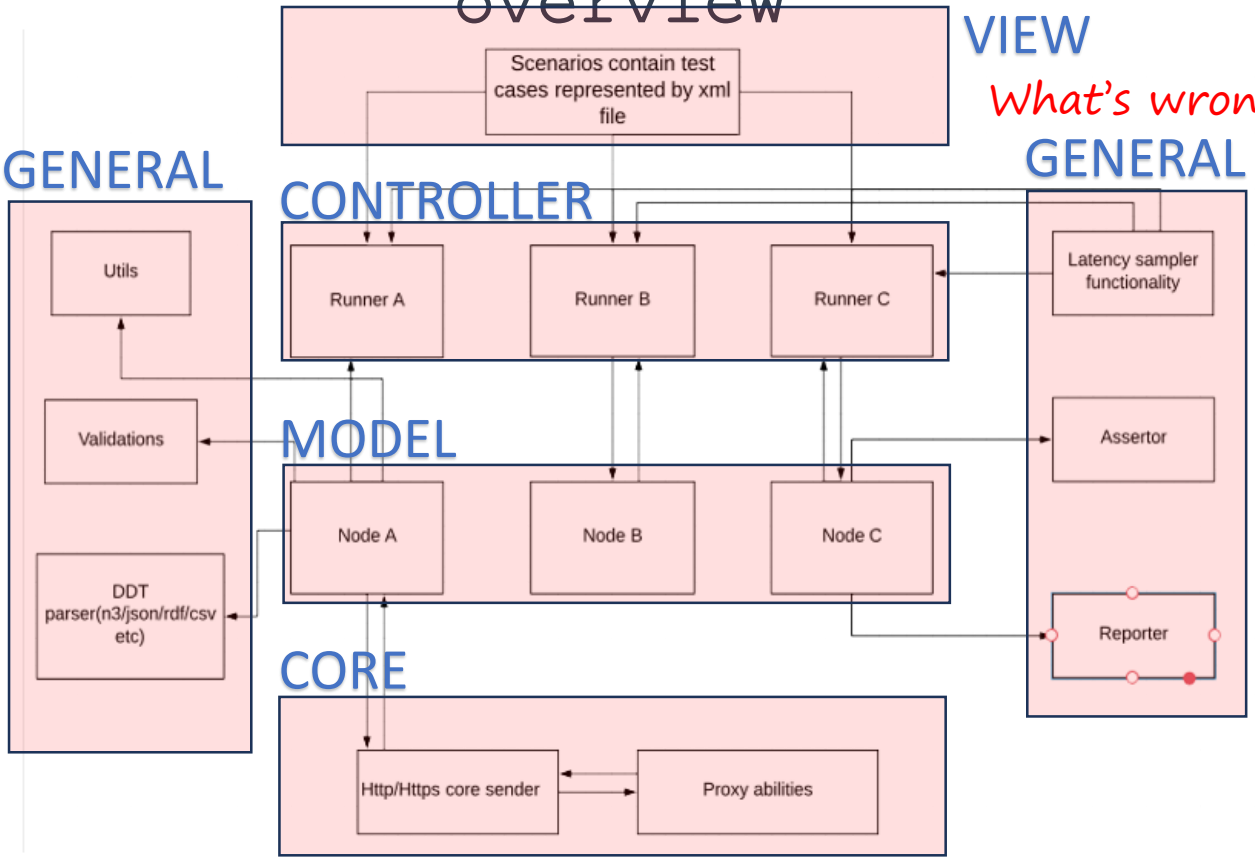
Divide and Conquer

- A very big mass multinational mass media company.
- Tagging & processing a massive amount of data each minute.
- Automation infrastructures were develop separately for each testing field.

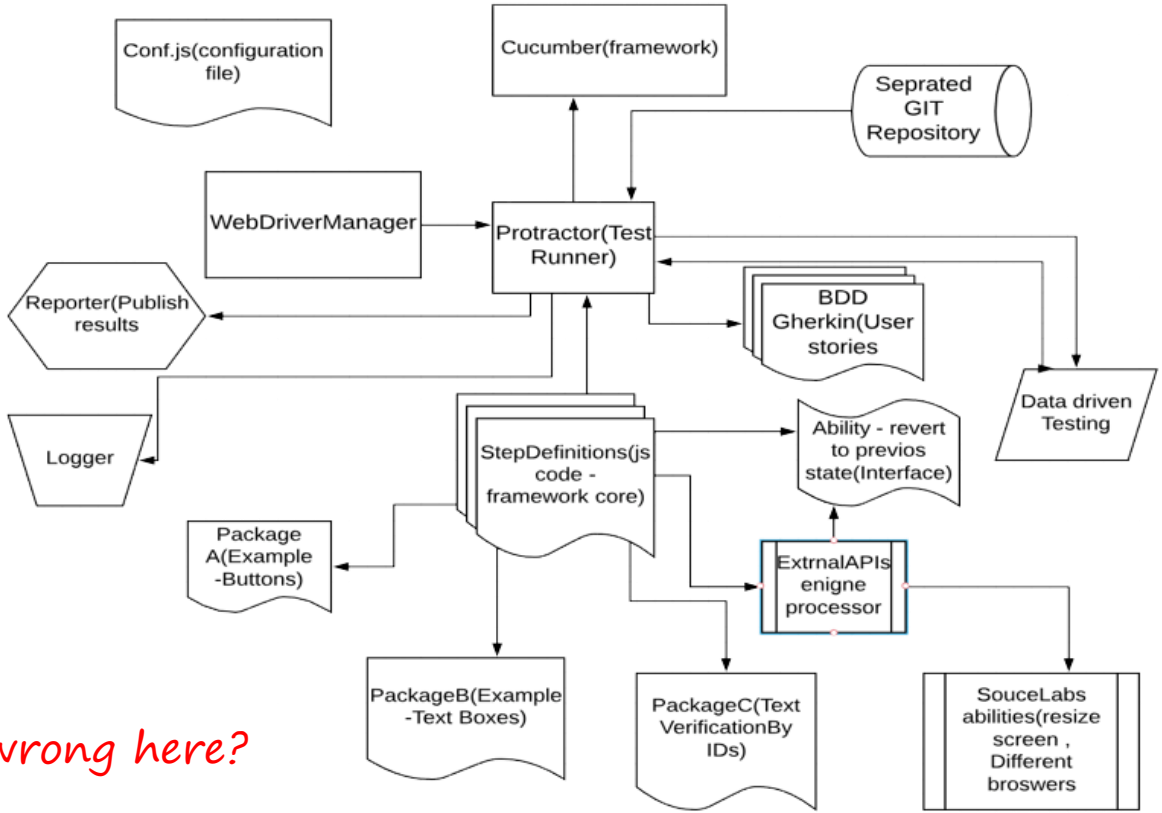
API Automation infrastructure – short overview



API Automation infrastructure – short overview



UI INFRASTRCTURE EXAMPLE *(BDD based)*



And What's wrong here?

MVC – C – G

- **Model** – The automation infrastructure “heart”.
- **View** – The infrastructure’s input layer.
- **Controller** – The “runner”, basically runs and control the whole infrastructure.
- **Core** – Separated from the traditional model – why?
- **General** – General utilities

Pros

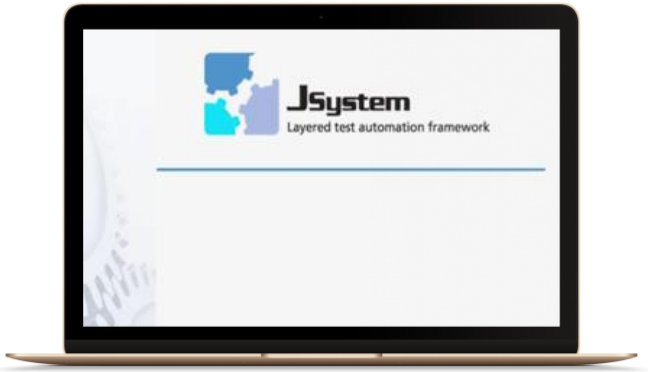
- Possible to develop each automation on another programming language.
- Build failures due to an infrastructure core/model issues of one infrastructure doesn’t affect other infrastructures(we don’t put our all eggs into 1 basket).
- Fast coding(no need to consider on other domains).

Cons

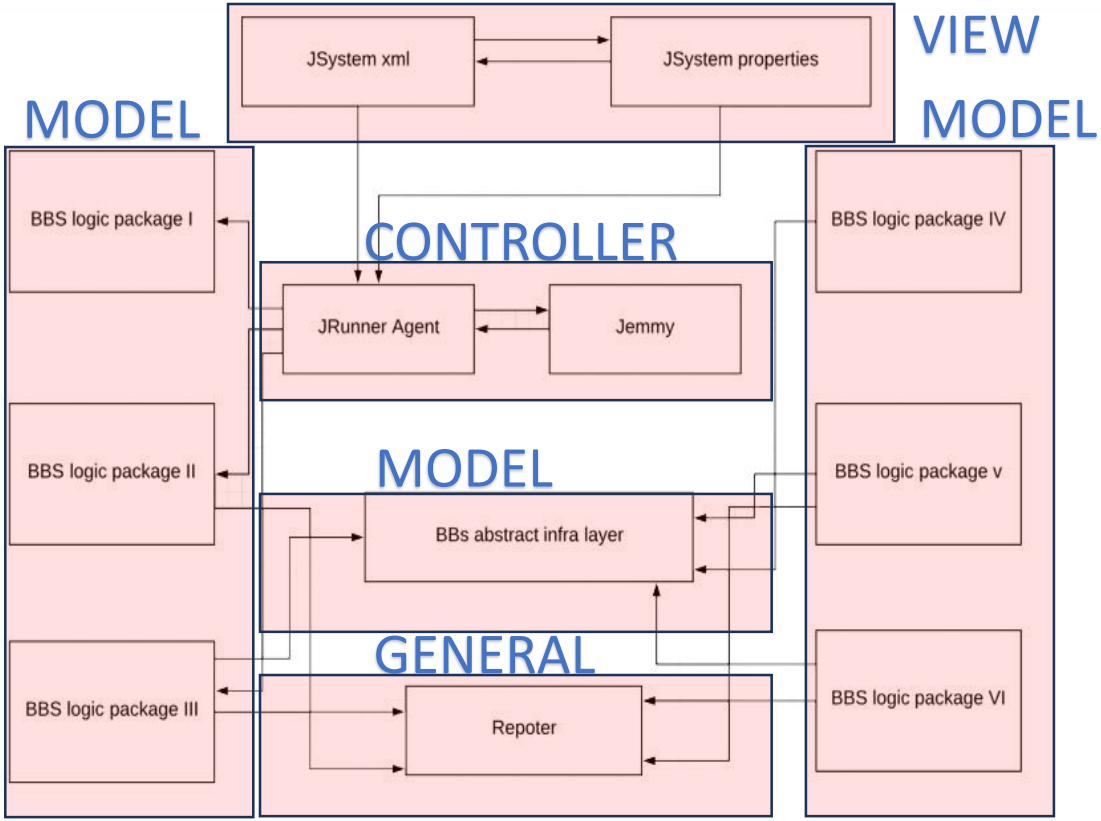
- Falling into the “weak runner” trap makes it harder to fit the MVC-C-G Model.
- Difficult to have a proper knowledge sharing while working on 4 infrastructures.
- Between these infrastructures a standalone utilities have to be developed to fill in some gaps that falling between the cracks.
- Hard to differentiate between Model and General layers.

The Many and The Few

- Big semiconductors industry company
- Testing on a multi disciplinary systems. The optics ,hardware and software need to run together and be synchronized.
- The real challenge is to run the automation on both simulator and a real “breathing” machines.
- Desktop GUI testing Java swing based.



Many Testers – Few Developers



Pros

- Fast development process.
- Easy to develop - manual qa testers can compose a test just by drag&drop Building Blocks of code without even know how to code at all.
- Easy to maintain – fixing the code on the building block layer.

Cons

- Very shallow way of testing, can’t dive too deep.
- Very hard to run automation on real machines.
- Never hard code anything and if you do don’t dare to push it.
- Core layer embedded inside the model layer.
- Test relevancy issue, need to monitor test relevancy all the time in order to save space on the nightly shards and tester’s time analyzing the failures. **Can you think of any solution for that?**

Tests Relevancy solution

- Using sonar to catch duplications which IDE missed to spot.
- Hard code review. (pass mr/pr to each other).
- Documentation – Documentation – Documentation.

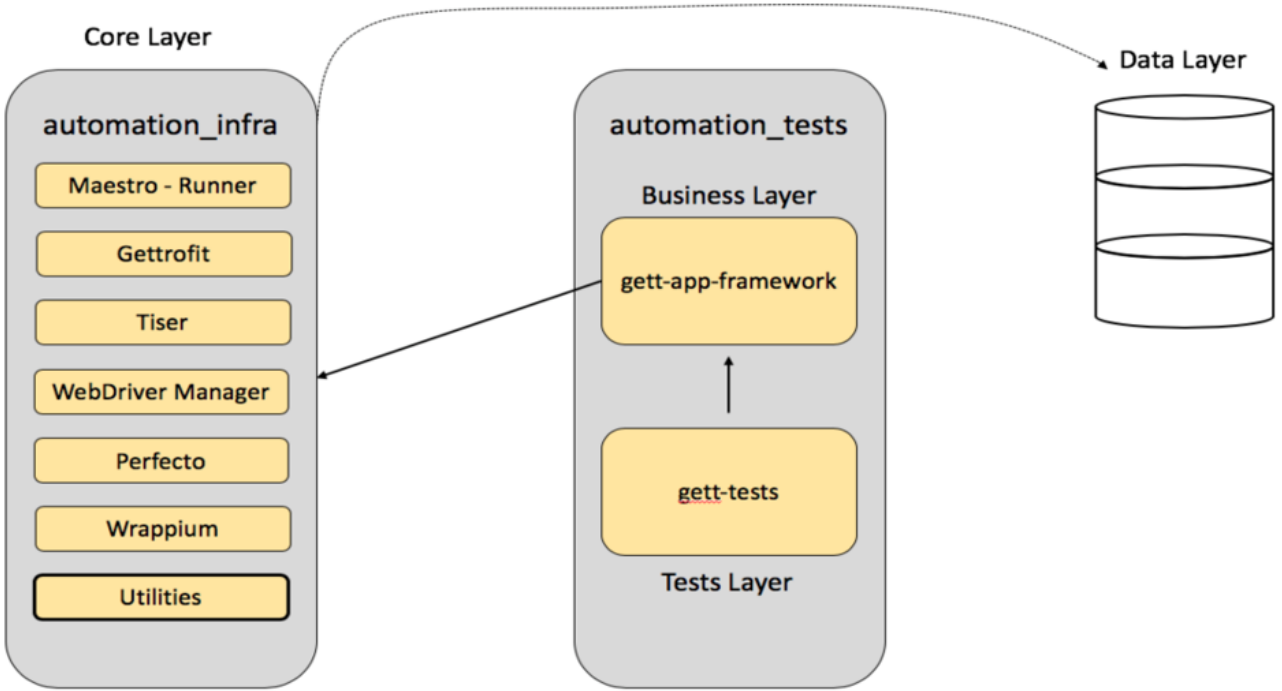
TESTING STAGE

Anyone recognize?

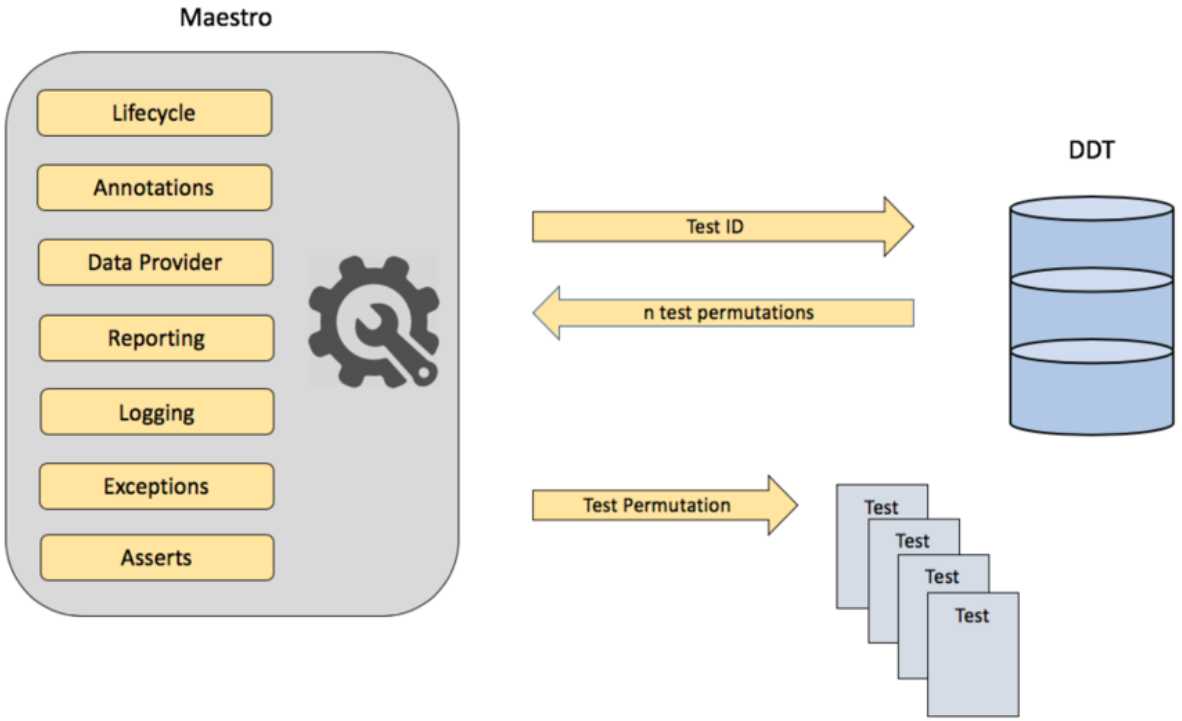
One infrastructure to rule them all

- Moving from an old infra into a new one, switching from Junit + JSystem to our own infrastructure testNG based.
- Dedicated infrastructure team developing the infrastructure.
- Have to support all kind of testing : mobile + web + API.

0. Gett infrastructure to rule them all

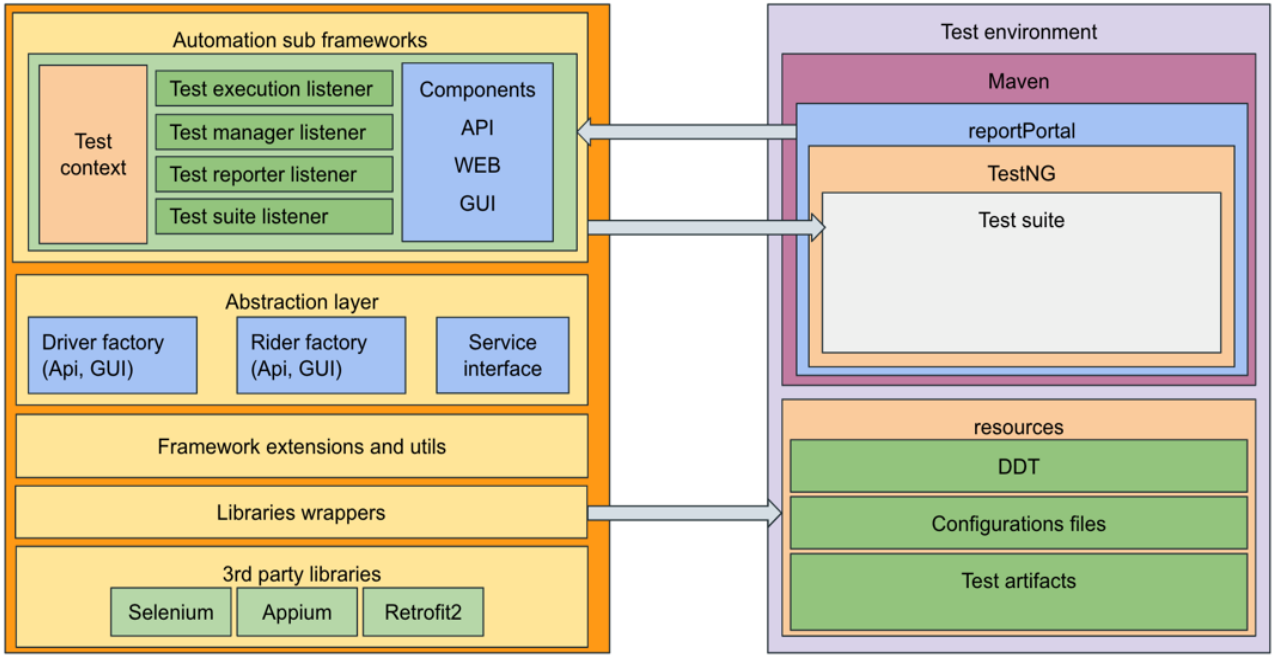


One infrastructure to rule them all



One infrastructure to rule them all

Architecture



Pros

- Full Collaboration and knowledge sharing between all automation teams (cross scrum development).
- General utilities, DB business and other logics are mutual.
- Possibility to develop an e2e flow. On separated infrastructures a Jenkins multijob/bash files have to run in order to run an e2e flow.
- High Code standards (java format, sonar, ci).

Pros

- Possible to run with only one runner job on the ci/cd.
- Generics! Generics! Generics! it makes the infra 10 times more powerful.
- Easy reporting mechanism.
- It is the right move to admit that time has come and we need to build a new infra.

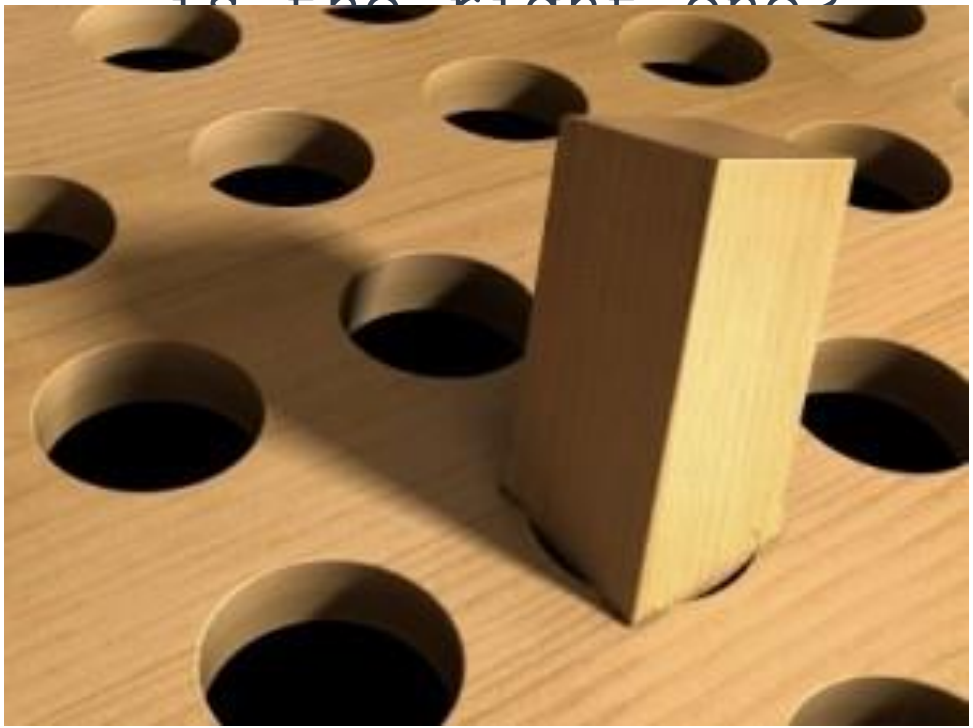
Cons

- From the testers side – big dependency on the infra team.
- Might be hard to maintain , need to consider all domains on every fix.
- Long development process time while developing the infra itself
- Don’t use more than one layer of abstract.

So, which method of infra developing is the right one?

There is no absolute Truth

So, which method of infra developing is the right one?



TESTING STAGE

Thank you

Please stay in touch

Yossi.Rosenberg@gett.com

LinkedIn: Yossi Rosenberg

Yossi Rosenberg

Gett