Data.dλy

Data Science desde el corazón industrial de MX

14 de noviembre de 2023
Monterrey, México.
https://sg.com.mx/dataday/

UNIT TESTING EN BIG DATA: POTENCIANDO LA CONFIANZA DE NUESTROS ETLS

Ludim Sánchez

Data Engineer @ Spin by OXXO
Ing. de software @ Chica Dev

Economía y decisiones de negocios

Automatizar

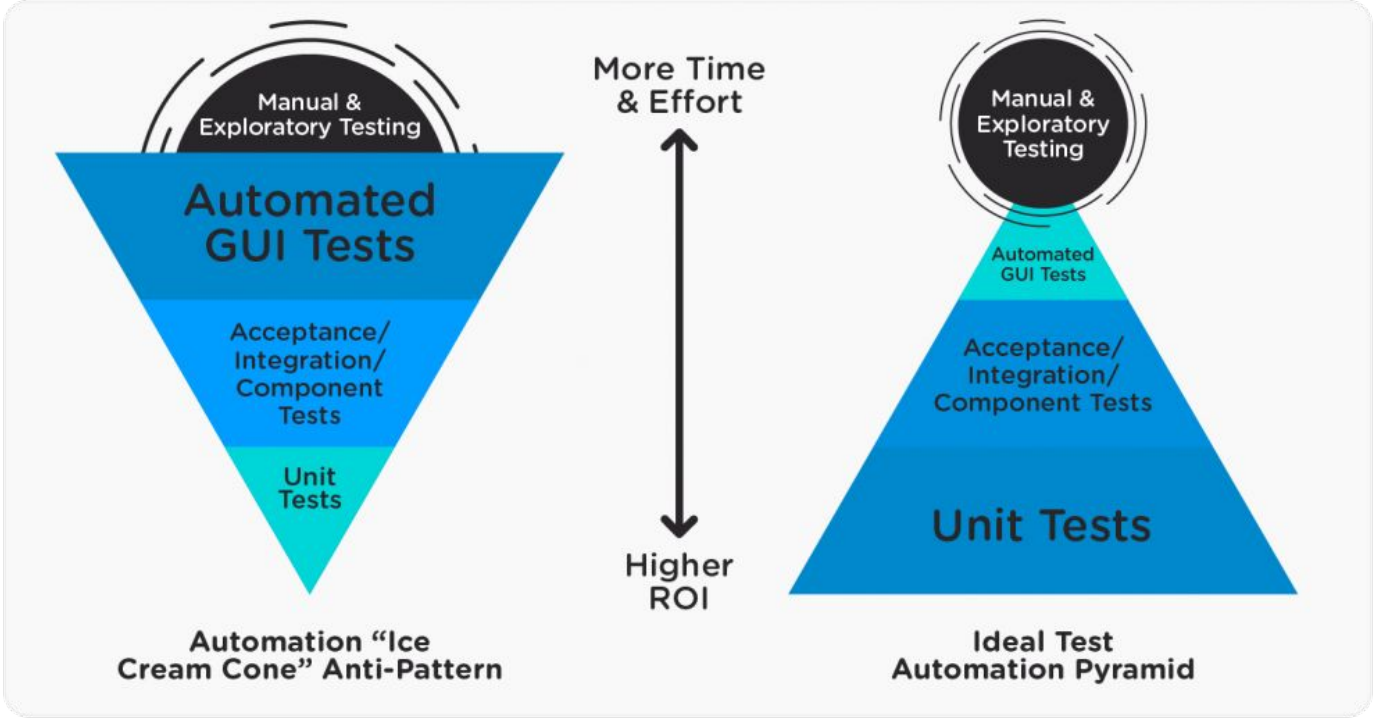
Automatizar

¿Qué es lo primero que propones?

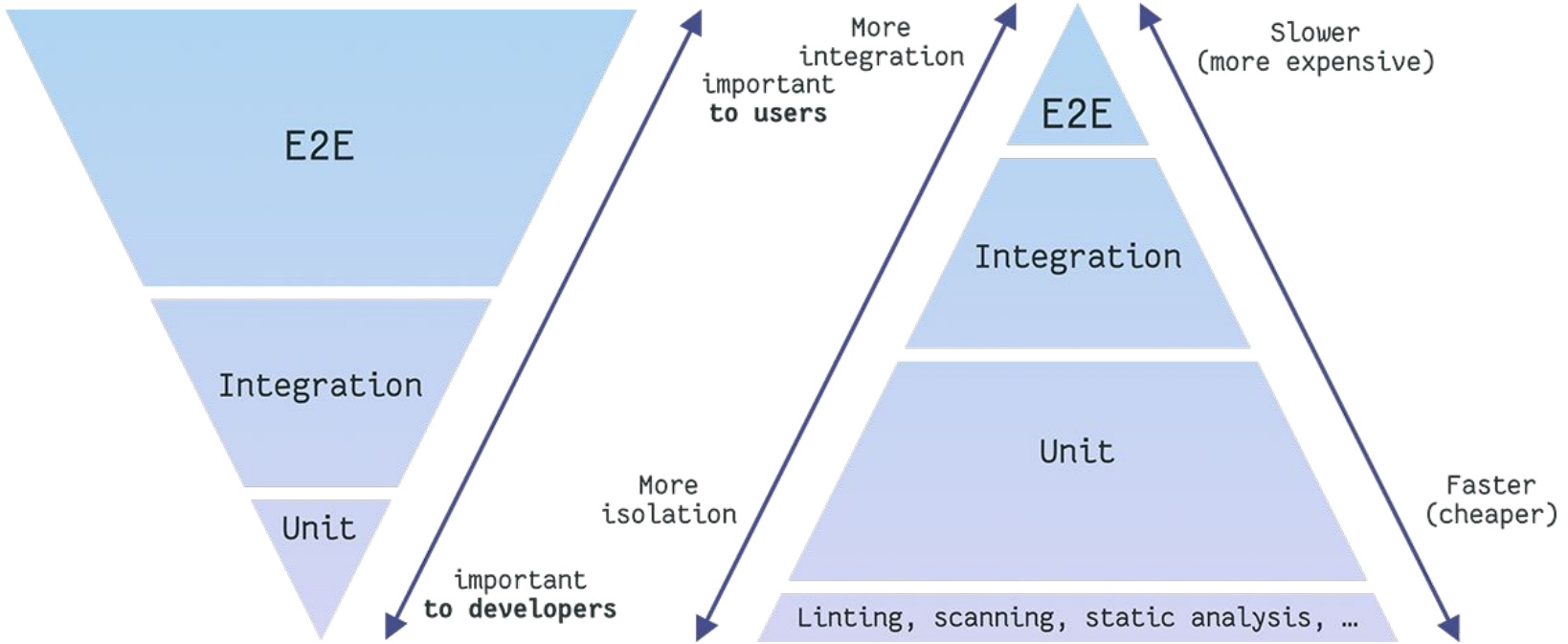
¿Qué es lo que quieren los tomadores de decisiones de la empresa?

- Automatizar todos los escenarios en donde sea posible - ***¿Dónde estamos hoy? ¿Qué tengo y qué necesito?***
- Consolidarse como una empresa *data-driven* - ***¿Dónde estaremos mañana?***
- Generar una forma de trabajo estándar - ***¿Cómo saber si lo estamos logrando?***

Pirámide de Pruebas de Automatización



Visto desde otro ángulo



De: https://buddy.works/tutorials/integration-testing-for-aws-lambda-in-go-with-docker-compose

¿Qué significa *unit*?

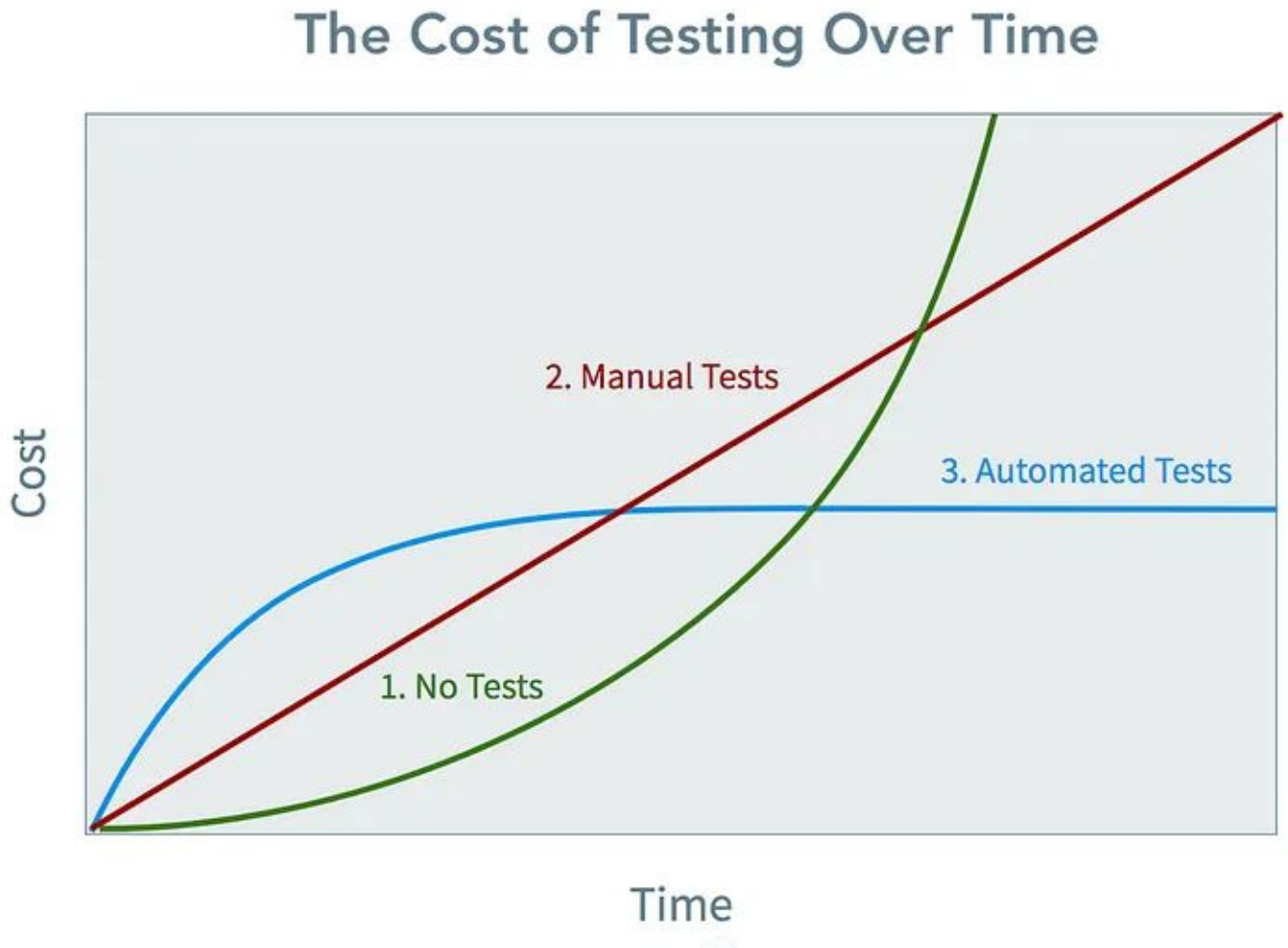
Depende…

Lenguaje funcional, lo más probable es que una unidad sea una única función.

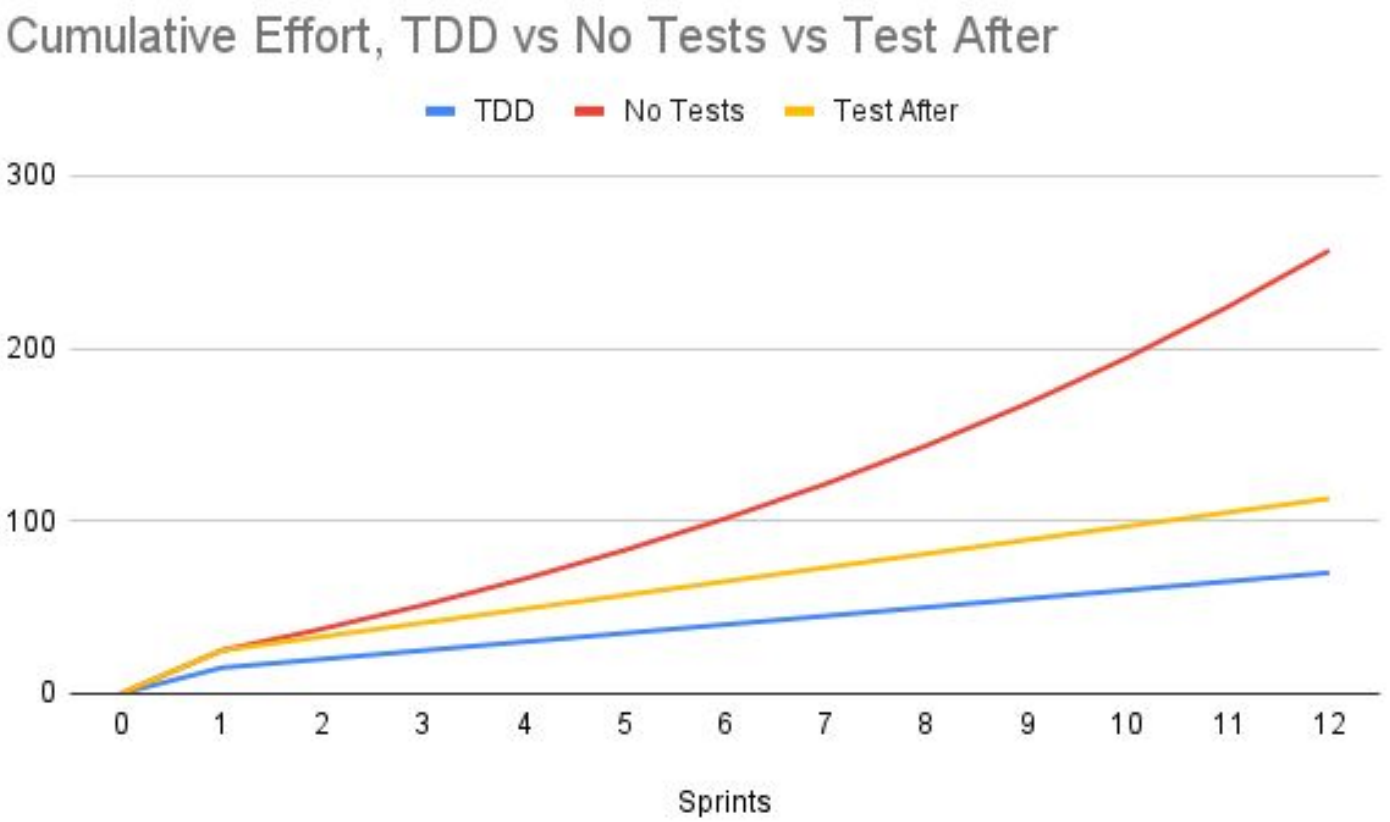
Lenguaje orientado a objetos, una unidad puede ser desde un único método hasta una clase entera.

Una prueba unitaria de software —también conocida como unit testing— es el instrumento utilizado para validar un fragmento de código fuente.

Los desarrolladores aíslan una línea del lenguaje codificado para saber si el sistema está operando correctamente en una función, proceso o actividad específica.



De: https://www.karllhughes.com/posts/testing-matters



De: https://jhall.io/posts/tdd-roi/

Ventajas

Velocidad

En detección de fallos, las modificaciones son sencillas por lo que los tiempos disminuyen.

Reducción de riesgos

Se aplican en etapas de desarrollo con el objetivo de prevenir fallos en etapas posteriores.

Aseguran la calidad del Desarrollo

De las pruebas depende el buen funcionamiento del sistema, satisfacción de usuario y crecimiento de la Empresa.

Integración

Seleccionan las líneas de código en partes pequeñas y facilitan la integración de bloques de código de mayor complejidad y libre de errores.

Diseño intuitivo

Cada línea de código es una pieza del rompecabezas que permite intuir cuál es la siguiente parte del sistema, agilizan la labor de diseño.

¿Y para proyectos de datos?

Las pruebas unitarias de datos son muy útiles para saber **cuándo cambian los datos**, cuándo los datos están **obsoletos** o **almacenados en caché**, y para evitar que los datos erróneos arruinen los modelos de aprendizaje automático o los informes.

Son buena manera de documentar el aspecto que debe tener el conjunto de datos.

Tener código mejor estructurado

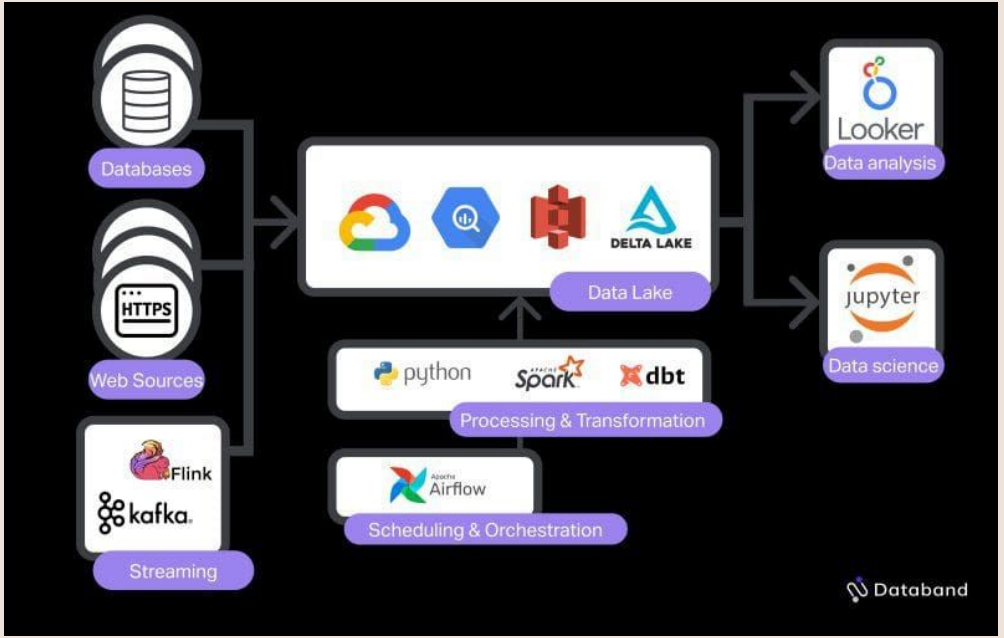
Confianza en los resultados

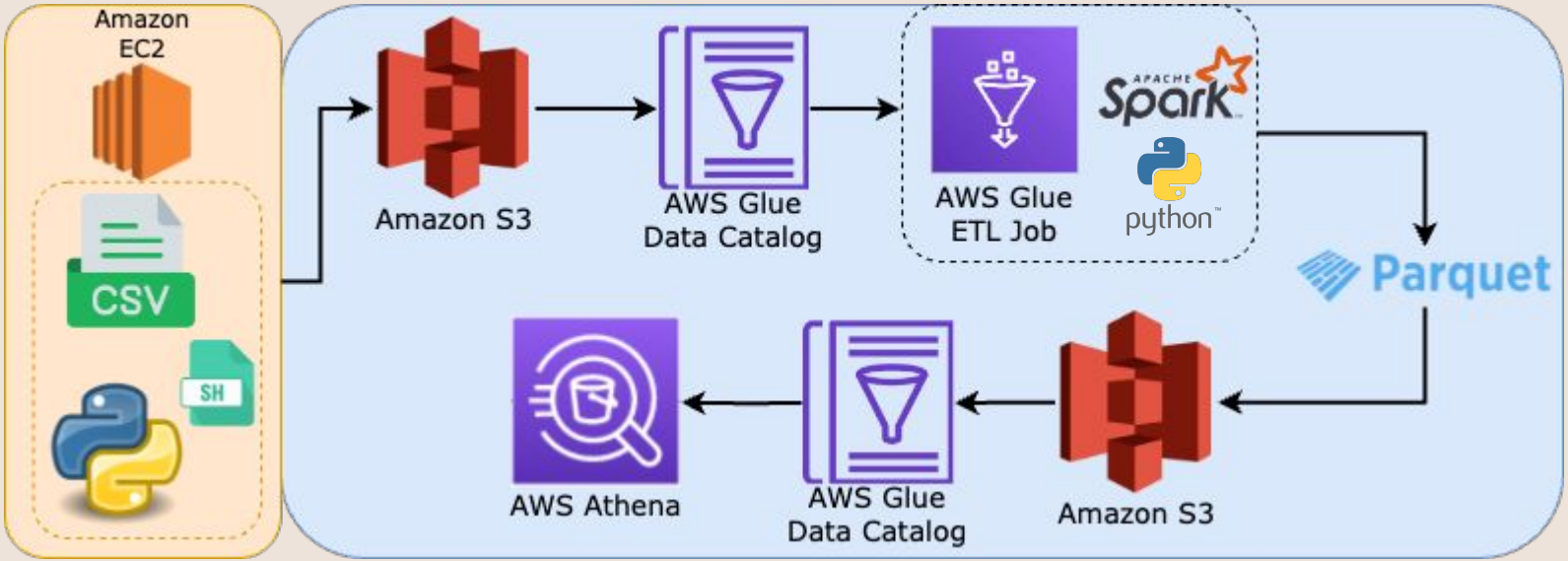
Ejemplos de pruebas

¿Qué tipo de pruebas podemos hacer?

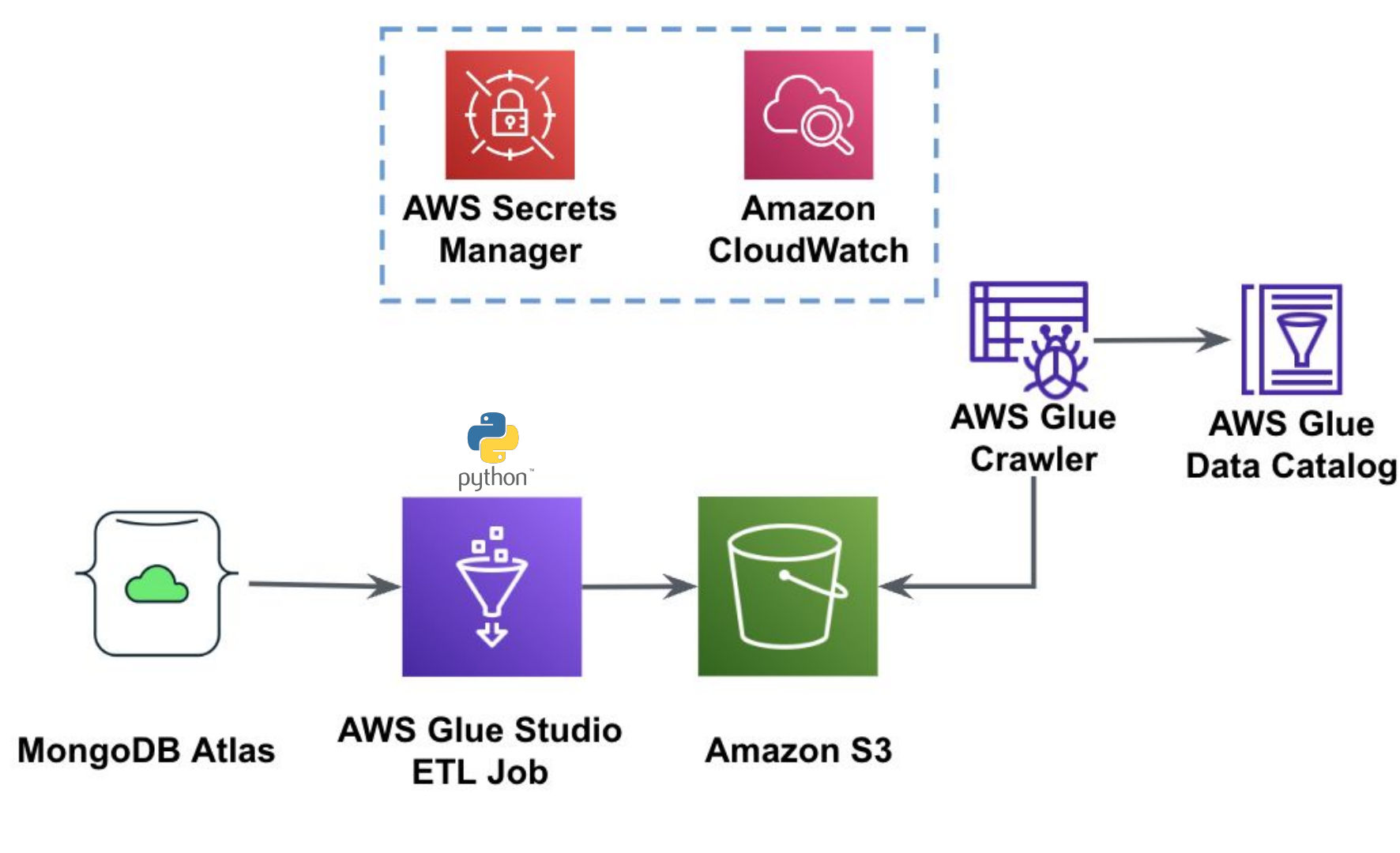
- Comprobación de nuevas columnas.
- Valores únicos para determinada columna
- Valores están dentro de cierto rango
- Comprobación de filtros / cruces

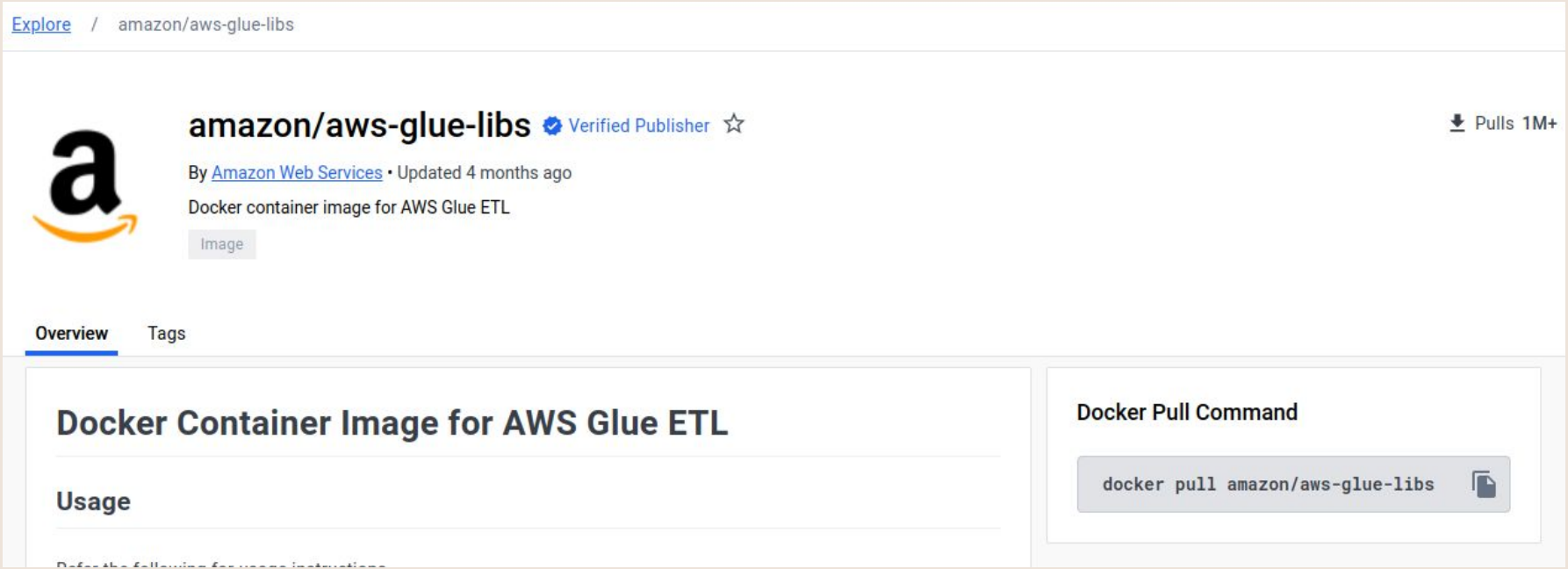
Manos a la obra



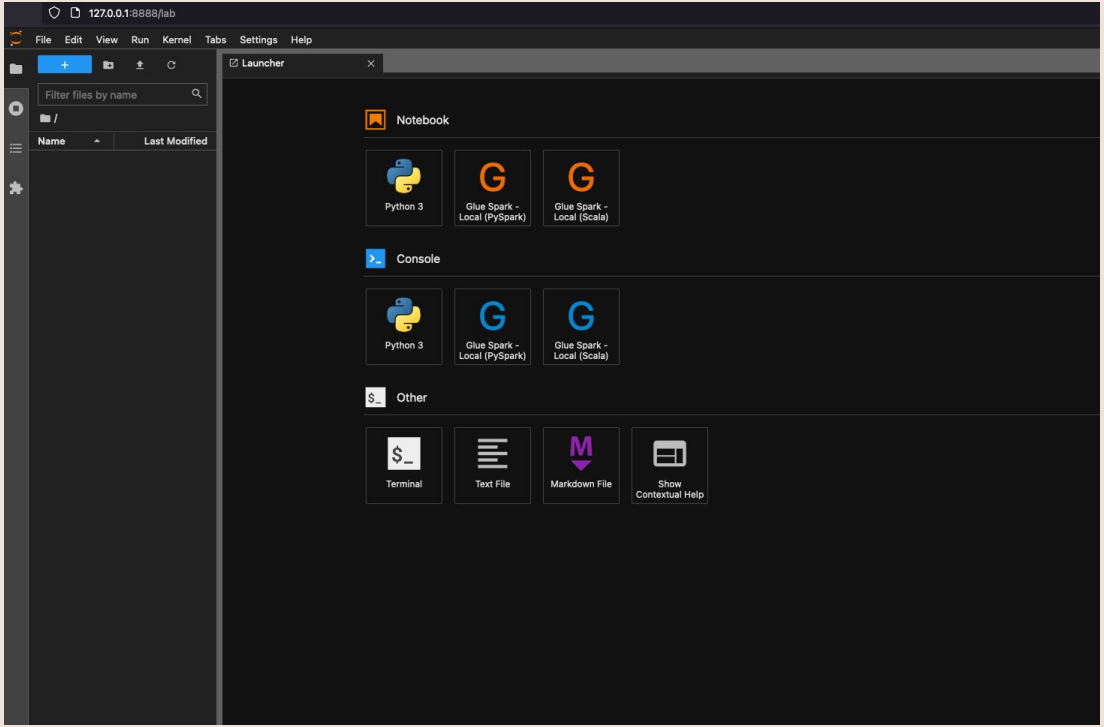


De:https://medium.com/@dogukannulu/aws-cloud-data-engineering-end-to-end-project-aws-glue-etl-job-s3-apache-spark-967d6ebe1d88


AWS Secrets Manager
Amazon CloudWatch
AWS Glue Crawler
AWS Glue Data Catalog
python
MongoDB Atlas
AWS Glue Studio ETL Job
Amazon S3



https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-libraries.html

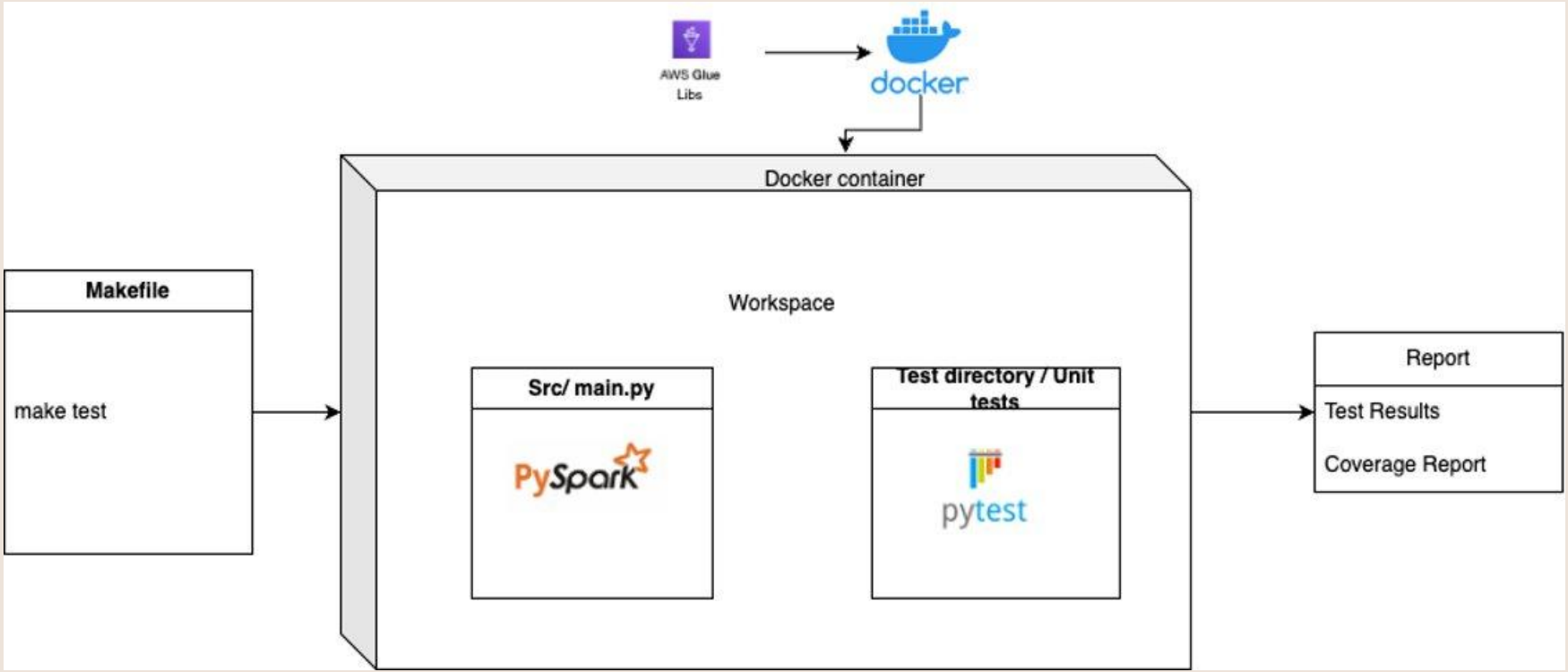



```
$ JUPYTER_WORKSPACE_LOCATION=/local_path_to_workspace/jupyter_workspace/
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v
$JUPYTER_WORKSPACE_LOCATION:/home/glue_user/workspace/jupyter_workspace/ -e
AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 -p
8998:8998 -p 8888:8888 --name glue_jupyter_lab
amazon/aws-glue-libs:glue_libs_4.0.0_image_01 /home/glue_user/jupyter/jupyter_start.sh
```

Problemas...

VPN
MFA
Ambientes

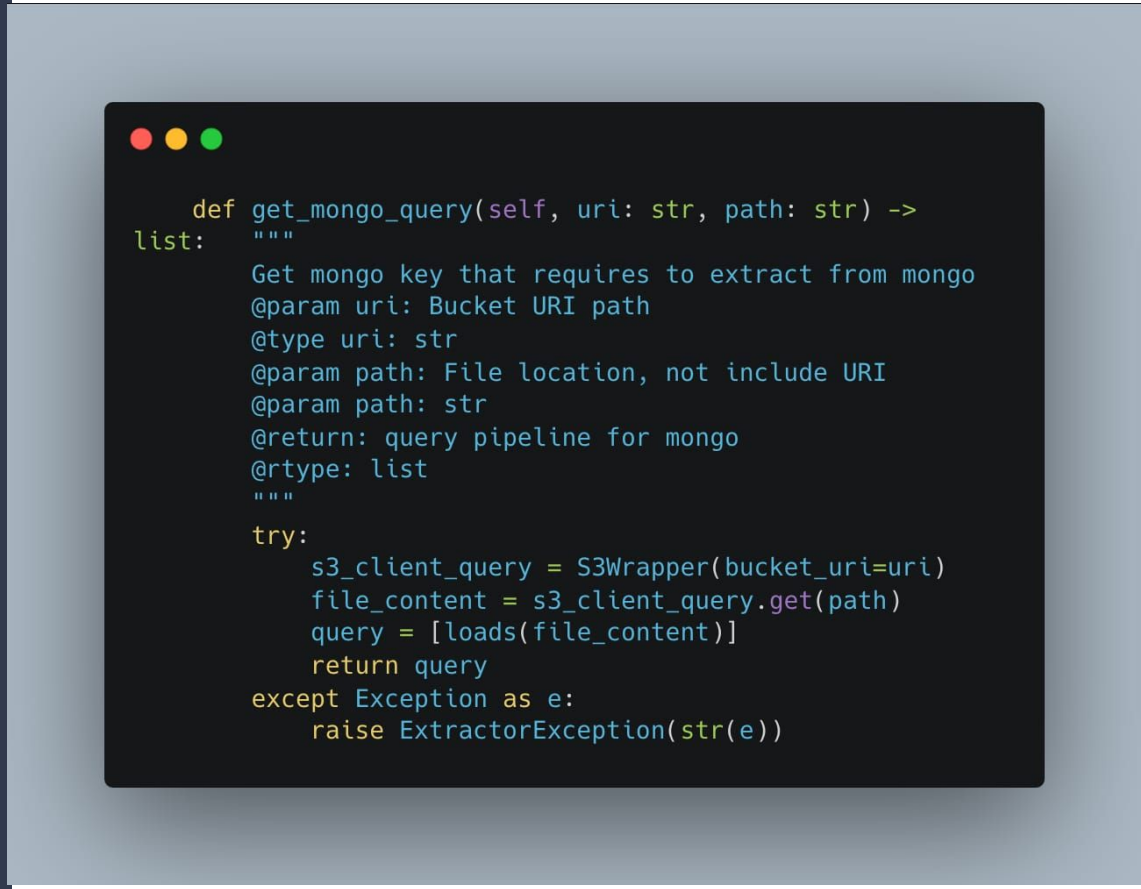
```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v
$WORKSPACE_LOCATION:/home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e
DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_pytest
amazon/aws-glue-libs:glue_libs_4.0.0_image_01 -c "python3 -m pytest"
--AWS_ACCESS_KEY_ID=$ACCESS_KEY_ID --AWS_SECRET_ACCESS_KEY=$SECRET_ACCESS_KEY
--AWS_SESSION_TOKEN=$SESSION_TOKEN
```


AWS Glue
Libs
docker
Docker container
Makefile
make test
Workspace
Src/ main.py
PySpark
Test directory / Unit tests
pytest
Report
Test Results
Coverage Report

Organización de un job

```
|--- Dockerfile
|--- Makefile
|--- README.md
|--- requirements.txt
|--- requirements_test.txt
|--- resources/
     |--- query_mongo.json
     |--- query.json
|--- set_credentials.py
|--- src/
     |--- __init__.py
     |--- main.py
|--- tests/
     |--- conftest.py
     |--- test_main.py
|--- tests_resources/
```

Hello, world!



```
    def get_mongo_query(self, uri: str, path: str) ->
list:   """
        Get mongo key that requires to extract from mongo
        @param uri: Bucket URI path
        @type uri: str
        @param path: File location, not include URI
        @param path: str
        @return: query pipeline for mongo
        @rtype: list
        """
        try:
            s3_client_query = S3Wrapper(bucket_uri=uri)
            file_content = s3_client_query.get(path)
            query = [loads(file_content)]
            return query
        except Exception as e:
            raise ExtractorException(str(e))
```

Mi primera prueba

```python
class TestExtractorMongo(unittest.TestCase):
    def setUp(self):
        env = "dev"
        db = "stores"
        collection = "store"

        self.mongo_extractor = ExtractorMongo(env, db, coll)

        self.bucket_uri = "jobs_assets"

def test_get_mongo_query(self):
        """
        Intentar un archivo
        @return:
        """
        key = "resources/query_mongo.json"
        query = self.mongo_extractor.get_mongo_query(self.bucket_uri,
key)    assert type(query) == list

        expected_result = [{
            "$project": {
                "id": "$_id",
                "name": "$name",
                "location": "$location",
                "postalCode": "$postalCode"
            }
        }]
        assert query == expected_result

        with self.assertRaises(Exception):
            self.mongo_extractor.set_mongo_query(self.bucket_uri, None)
```

Buenas prácticas

- Prueba unitaria por método, bloque de Código
- Nombres de prueba apropiados, se recomienda usar el prefijo test + el nombre del método: test_método (python)
- Pruebas simples, para mejor mantenibilidad y legibilidad.
- Minimizar (Evitar) la dependencia de las pruebas, para que factores externos no interfieran en el resultado de la prueba.
- Apuntar a la máxima cobertura de prueba, validar al menos un caso de éxito y otro de fracaso.
- Diseñar pruebas unitarias para que sean lo más rápidas posibles, para que no interrumpan el proceso y se puedan usar con frecuencia
- Automatice las pruebas

Reportes de cobertura

Coverage report: 87%

Module	*statements*	*missing*	*excluded*	*coverage*
mymath.py	9	3	0	67%
test_mymath.py	14	0	0	100%
Total	**23**	**3**	**0**	**87%**

coverage.py v4.1, created at 2016-07-18 15:04

Primeros pasos en la automatización

`make build`

Una imagen única para todos los devs

`make test`

Ejecución de todas pruebas unitarias en job

`make test-coverage`

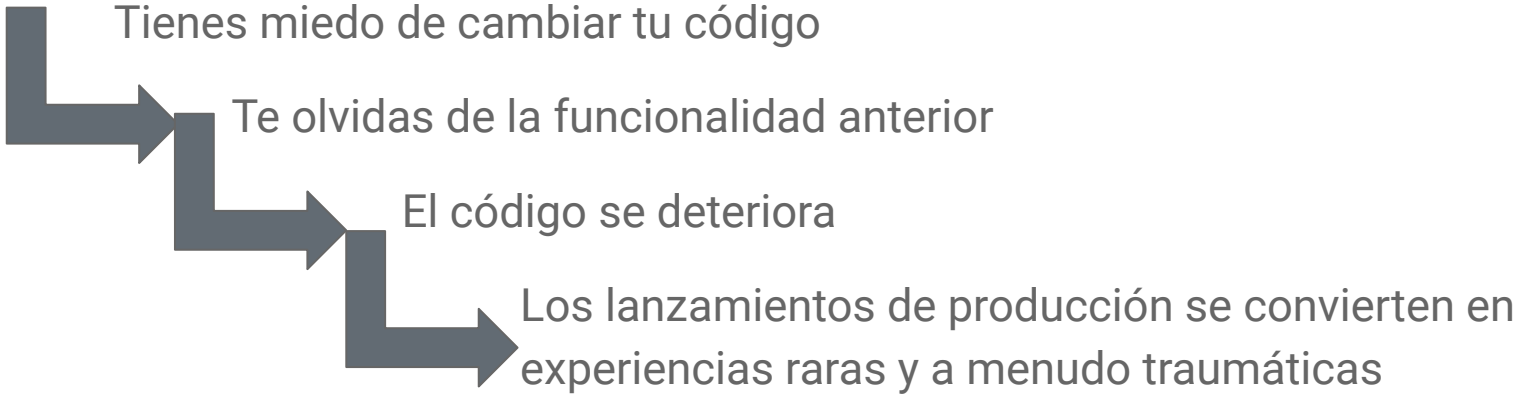
Resultado de cuánto código fue probado

Retos

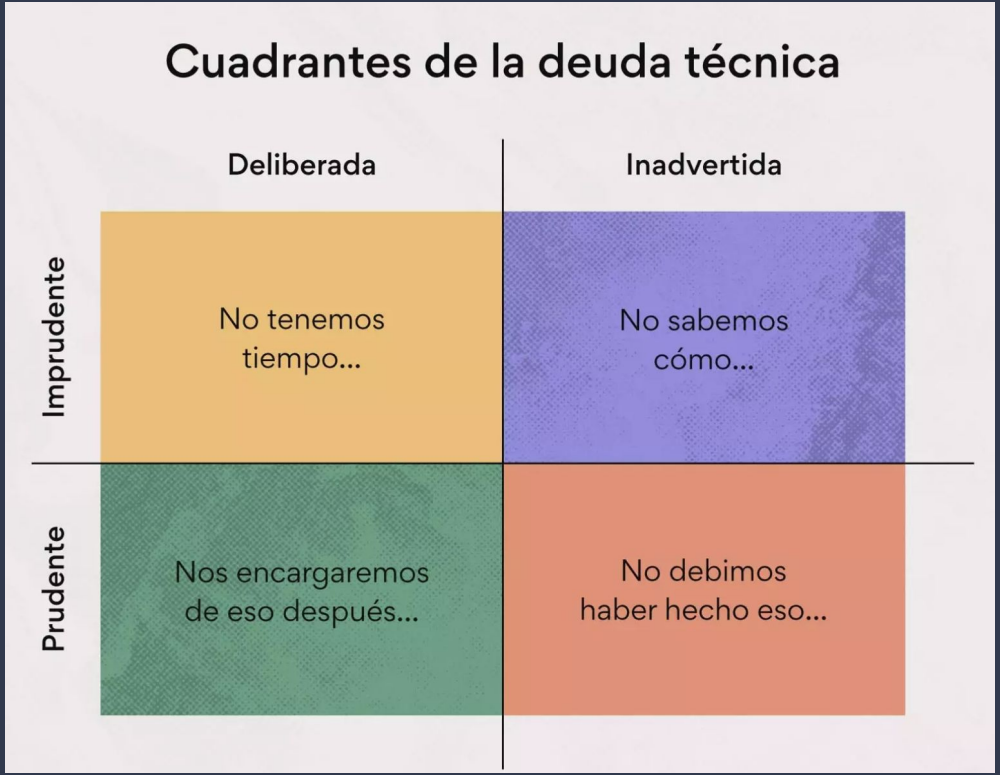
1. La curva de aprendizaje
2. Cambiar en énfasis del desarrollo a las pruebas
3. Obtener el apoyo de la organización (reducción temporal de productividad)
4. Dificultades para adaptar código de sistemas legados
5. Sesgo aplicado cuando los desarrolladores diseña sus propias pruebas
6. Alto costo de errores no identificados

NO
Tiempo, dinero y esfuerzo

¿Qué sucederá inevitablemente?



Sabiendo eso... ¿en qué cuadrante quedamos?



De: https://asana.com/es/resources/technical-debt

Agradecimientos

Víctor Herrera

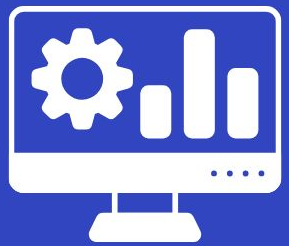
Data QA Engineer de confianza

¿Preguntas?

WOMEN WHO CODE /monterrey

Women in data MTY

Únete a nuestro grupo de estudio de Data science & arquitectura de software



Comming soon 2024.
Grupo de estudio virtual
https://t.ly/_TAr6

https://t.ly/_TAr6

https://www.meetup.com/women-who-code-monterrey/events/297237051