Git

, לראות מה חדש התווסף,, ואיזה שינויים חלו מתעדת שינויים בקודים

מאפשרת שיתוף פעולה בו זמנית מבלי לפגוע בקוד של השני

.....מאפשרת שינויים מבלי להרוס את הקוד המאסטר.

מאפשרת קביעת יעדים וניהול פרויקט עם כלים מיוחדים.

היא מזהה את הקובץ לפי התוכן ולא לפי השם.

Unit test:

בודקים פונקציה ספציפית בclass

בשאר הבדיקות הן מערך של פעולות

unit test should be a class

נכתב על ידי המפתח עצמו

Mocking

לדמות התנהגות של אובייקט מסויים

דוגמא: בדיקה של פונקציה שמעלה משתמש ממשתמש רגיל לאדמין...לא עושים כל תהליך ההרשמה..נכין כמות משתמשים מראש ומשתמשים בו בפונקציה.

מכינים את data לפני הפעולה בתוך הפונקציה

Assertion

מאפשר להבין מה התוצאה הצפויה והאם התוצאה נכשלה או לא ואז בודקים איפה הכשל בשביל לתקן.

Test driven development

יודעים מראש מה הפונקציות שרוצים לכתוב ..מצפים את התוצאה כי יודעים את הבדיקות ואז כותבים את הפונקציה

Python testing libraries:

unittest: most popular in python, built in

pytset: need to be installed, more advanced

module: a python file that contains what should be tested

unit test: a class named Test\_Module\_Name

methods inside the class are named: test\_<name> of function from module

מכינים שטח

מבצעים פעולה

עושים assert

לכסות את כל האפשריות

לדאוג לכתב נקי וקריא ולתחזק אותו

no input in tests because that may stop the automatic process.

CICD: תהליך בעולם התוכנה לאחר תהליך הבדיקות יש את השלב הזה הוא לוקח את כל האפליקציה ובודק אותה בשלבים

Automatic test structure

לא עושים טסט שבודק כמה דברים, כל טסט בודק משהו אחד,

טסט אוטונומי: עצמאי וממקוד , דוגמא: הטסט עצמו בוחר מוצר מהאתר ומכניס לעגלה, הוא דואג למידע שלו, לא צריך שטסט אחר יתערב.

טסט אטומי: בודק דברים ספציפיים שמגדיר אותם.

Complexity:

מורכבות של טסט... הרבה צעדים מסבכים אותי, דורש תחזוקה יותר

Execution time:

לקצר זמנים כמה שיותר

Fragility:

ייתכן שהטסט יישבר תוך כדי, להכין מראש דברים

Issue identification:

אם הטסט קצר קל לאתר איפה הבעיה, בטסטים ארוכים של 50 צעד למשל יהיה קשה...אז צריך לחלק אותו.

Limited isolation:

בטסט קטן קל מאוד לבודד את הבעיה כי הוא בודק דבר אחד.

Feedback delay:

כל לזהות את הבעיות כשיש טסטים קטנים לכן הפידבק יהיה קל יותר.

טסטים מבודדים:

יתרונות

זמן ריצה קצר

קל לתחזוקה

קל למצוא בגים

זיהוי בעיות מוקדם

חסרונות:

קשה להבין כמה כיסוי יש לנו

הרבה טסטים

לא מכסה הרבה מקרים

דורש זמן ומשאבים

**העיקר : טסטים קטנין ועצמאיים**

AAA: Arrange, act, assert

מחלקים את הטסט לשלושה חלקים

Arrange

מסדרים את ה data, סתם דוגמא לבחור מוצר

Act

עושים את הפעולה, דוגמה מוסיפים לעגלה את המוצר

Assert

פעולה השוואה, איזה תוצאה אני מקבל

Act: includes the action that affects the assert

Setup & Teardown

Setup

להכין מידע פעם אחת ואז הטסטים משתמשים במידע הזה,

Before all:

לפני כל הטסטים מגדיר מידע

Before each:

לפני כל טסט מגדיר מידע

Teardown

לנקות את המידע שהכנסנו, להחזיר סביבה נקיה.

After each

אחרי כל פעולה מבצעים ניקוי

After all

אחרי כל הפעולות מנקים ומחזירים סביבה נקיה

In QA automation, the concepts of autonomous tests and atomic tests are crucial for creating efficient and effective test suites. Here’s a breakdown of what each term means:

**Autonomous Tests**

**Autonomous tests** are designed to be independent of one another. They do not rely on the outcome or state of other tests to function correctly. This independence is crucial for several reasons:

1. **Parallel Execution**: Autonomous tests can be executed in parallel, which significantly reduces the total time required to run the entire test suite.
2. **Reliability**: If a test fails, it doesn't impact the execution or outcome of other tests. This makes it easier to identify and isolate issues.
3. **Maintenance**: Autonomous tests are easier to maintain since changes in one test do not necessitate changes in other tests.
4. **Scalability**: The test suite can grow without the risk of tests interfering with one another.

To achieve autonomy, each test must:

* Set up its own preconditions.
* Clean up after itself to avoid side effects on subsequent tests.
* Be self-contained, meaning it should not depend on the result of another test.

**Atomic Tests**

**Atomic tests** refer to tests that verify a single functionality or behavior in isolation. The principle behind atomic tests is to focus on one specific aspect of the application at a time, ensuring that each small part of the application works as expected. Key characteristics of atomic tests include:

1. **Simplicity**: Each test has a clear and simple purpose, testing one piece of functionality.
2. **Isolation**: Atomic tests run independently of other tests and do not require complex setup or teardown processes.
3. **Clarity**: They are easy to understand and maintain because they focus on one aspect of the application.
4. **Precision**: Failures in atomic tests provide precise information about which part of the application is not working correctly.

**Differences and Overlaps**

* **Scope**: While autonomous tests focus on independence from other tests, atomic tests focus on the granularity of what is being tested.
* **Interdependency**: An atomic test can be autonomous, but not all autonomous tests are necessarily atomic. For example, an end-to-end test might be autonomous but not atomic, as it could test multiple functionalities together.
* **Design Philosophy**: Autonomous tests are designed to avoid dependencies, whereas atomic tests are designed to isolate a single piece of functionality.

**Best Practices**

* **Combine Both Principles**: Aim to design tests that are both autonomous and atomic where possible. This ensures that your tests are robust, reliable, and easy to maintain.
* **Modular Setup**: Use setup and teardown methods to ensure that tests do not leave behind state that could affect other tests.
* **Clear Purpose**: Each test should have a clear and focused purpose, making it easier to identify issues when a test fails.

By adhering to the principles of autonomous and atomic tests, QA automation can become more effective, leading to faster, more reliable, and maintainable test suites.

הבדלים בין טסטים ידניים לטסטים אוטומטיים:

בטסטים ידניים יש גמישות יותר,

ככל שהתמקד בהכנת טסטים אטומים ואנטומים יותר נגיע לתוצאה יותר גבוהה.

לכל בדיקות "יו-איי" ייתכן שמשהו עשה להם בדיקה ידני.

בדיקות אוטומטיות מתבצעות על ידי מחשב, בבדיקה ידנית יש שינוי כי מתבצע על ידי מישהו.

ממקדים את הבדיקה שתהיה ספציפית, עושים טסטים בצורה קצרה, כל צעד נחשב טסט.

מה שלא כתוב לא נבדק, מה שמבטיח כיסוי מבטיח לתרחישים.

--------------------------------------------------------------------------------

כשלים:

כשל טסט

כשל קוד

כישלון בהגדרה (ניסוח הטסט) או כישלון באימות (תוצאה סופית לא תואמת למה שמצפים = לפעמים באג)

לכן מחלקים את השלבים לטסטים בודדים

--------------------------------------------------------------------------------

False Flag

מצב גרוע, לקבל הודעת שגיאה אבל ההודעה עצמה שגויה, **אז לא בודקים האם תופיע הודעה אלה בודקים האם ההודעה תואמת למה שאני מצפה.**

מונעים את הבעיה הזו בלהיות ספציפי בדרישות.

מראה שהטסט עבר אבל לא עבר בצורה הנדרשת.

--------------------------------------------------------------------------------

False positive

הטסט עבר בהצלחה אבל ההודעה שקיבלתי לא כפי שציפתי.

False negative

הטסט נכשל לא כי התוצאה לא נכונה אלה כי יש בעיה בהגדרת הטסט עצמו (בקוד).

**False positive is worse than false negative**

--------------------------------------------------------------------------------

API VS. UI

מאחורי כל פעולה ב "יו-אי" מתבצעת פעולה ב "אא-פי-אי"

ב "אא-פי-אי" הפעולות יותר מהירות והוא יותר יציב מ ה"יו-אי"

נשאף לעשות כמה שיותר פעולות דרך ה"אא-פי-אי" ובסוף ניגשים לנקודה אשר רוצים דרך ה"יו-איי"

כל בדיקה שמתבצעת מאחורי לקלעים היא בדיקת "אא-פי-אי".

In automation testing, API (Application Programming Interface) testing and UI (User Interface) testing are two distinct approaches with different focuses, tools, and purposes. Here's a detailed comparison of the two:

**API Testing**

**Focus:**

* Tests the backend or server-side components of an application.
* Ensures that the APIs (functions/methods) are working as expected.
* Focuses on the functionality, reliability, performance, and security of the API.

**Tools:**

* Postman
* SoapUI
* REST Assured
* JMeter
* Swagger

**Advantages:**

* **Speed**: API tests are generally faster because they don't involve rendering and interacting with the UI.
* **Stability**: Less prone to changes compared to UI tests, as UI changes frequently but backend logic remains more stable.
* **Coverage**: Can cover more test scenarios, including edge cases, which might be difficult to achieve through the UI.
* **Early Detection**: Helps in identifying issues early in the development cycle.
* **Automation**: Easier to integrate into CI/CD pipelines due to their speed and reliability.

**Disadvantages:**

* **Limited Scope**: Doesn't test the user interface and user experience.
* **Complexity**: Requires understanding of the API endpoints, parameters, and responses.
* **Debugging**: Sometimes harder to debug issues as they are closer to the business logic and data layer.

**UI Testing**

**Focus:**

* Tests the frontend or client-side components of an application.
* Ensures that the user interface elements are working correctly and providing the expected user experience.
* Focuses on the look and feel, layout, and usability of the application.

**Tools:**

* Selenium
* Cypress
* TestComplete
* QTP/UFT
* Protractor

**Advantages:**

* **User Experience**: Tests the application from the end-user's perspective.
* **Visual Verification**: Ensures that the UI elements are displayed correctly and functioning as intended.
* **Interactivity**: Validates the interaction between various UI components.

**Disadvantages:**

* **Speed**: Generally slower due to the need to render the UI and perform actions like clicking buttons and filling forms.
* **Fragility**: More prone to breakages due to frequent changes in the UI layout and elements.
* **Maintenance**: Requires frequent updates and maintenance as the UI evolves.
* **Resource-Intensive**: Consumes more computational resources, making it less suitable for extensive test automation.

**When to Use Which:**

**API Testing:**

* Suitable for testing the core logic of the application.
* Ideal for validating business rules and workflows.
* Useful for performance testing and checking how the system handles load.
* Critical for ensuring data integrity and consistency.

**UI Testing:**

* Essential for validating user workflows and scenarios.
* Important for ensuring the visual correctness of the application.
* Necessary for verifying that the UI behaves as expected under different user actions.
* Useful for end-to-end testing, where the focus is on simulating real user interactions.

--------------------------------------------------------------------------------

Sleep & Retries

Sleep: time between each iterations

Retries number: number of tries

**ברגע שאותו אלמנט מחכים לו מגיע יוצאים מהלולאה של הבדיקות החוזרות, וגם אם לא הגיע הלולאה תצא ברגע מסוים.**

In automated testing, inserting a sleep or delay between iterations can be important for several reasons. However, it's crucial to understand that using sleep should be a last resort, and other synchronization mechanisms like explicit waits, implicit waits, or fluent waits are generally preferred. Here are some reasons why a delay or sleep might be necessary:

**1. Application Performance and Stability**

* **Load Handling**: Some applications may not handle rapid-fire requests or actions well. A delay between iterations can prevent overwhelming the application, especially in a test environment with limited resources.
* **Resource Management**: Allowing time for resources to be released between tests can prevent issues like memory leaks or database locks that might occur if tests run too quickly in succession.

**2. Asynchronous Processes**

* **Data Propagation**: In applications where data needs to propagate through various systems or databases, a delay ensures that the data is fully updated before the next test begins.
* **Background Tasks**: Some actions might trigger background tasks (e.g., email sending, file processing) that need time to complete before the next test step can proceed.

**3. UI Rendering and Stability**

* **Element Availability**: In UI testing, it ensures that elements are fully rendered and available for interaction. This is especially important for dynamic web pages where elements may take time to load.
* **Animations and Transitions**: Allows for animations or transitions to complete, ensuring interactions are performed on a stable state of the UI.

**4. Network and Server Latency**

* **Network Delays**: Accounts for network latency, ensuring requests are fully processed and responses received before proceeding to the next iteration.
* **Server Response Time**: Gives the server sufficient time to respond to the previous request, especially under load.

**5. Test Readability and Debugging**

* **Log Review**: Makes logs easier to review and debug by spacing out actions. This can be particularly helpful when investigating intermittent issues.
* **Human Monitoring**: When running tests in an environment where humans are monitoring the output, delays can make it easier to follow what the test is doing.

**6. Third-Party Services and APIs**

* **Rate Limits**: Respecting rate limits of third-party services and APIs by introducing delays between calls to avoid being blocked or throttled.

--------------------------------------------------------------------------------

Always green, clean & seen

להשאיר מאחורינו סביבה נקיה כמו שקיבלנו.