Qt Quality Assurance

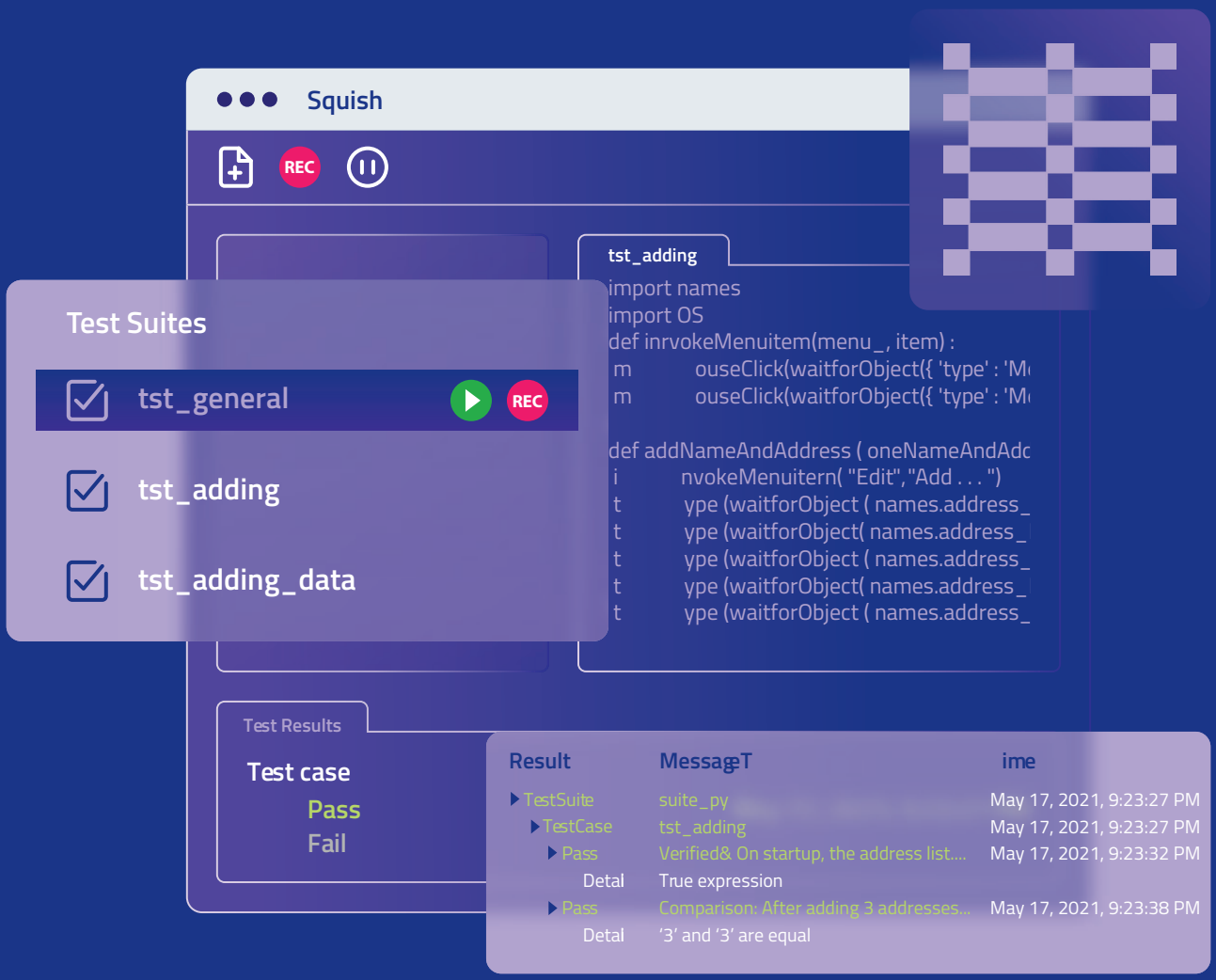
WHITEPAPER

高効果、低メンテナンス：テスト自動化の戦略

テストメンテナンスとは？

自動化テストメンテナンスに関するベストプラクティスとは？

Squishを使用した自動GUIテストで低メンテナンスなテストを実現するための戦略と実践について学びましょう



高効果、低メンテナンス：テスト自動化の戦略

GUIテストの自動化の目的は、手動テストに費やす時間を削減し、より複雑で重要なタスクに集中できるようにすることで効率を向上させることです。
しかし、どのように実現するのでしょうか?

自動テストメンテナンスのベストプラクティスを議論する前に、まずテストメンテナンスの定義を見てみましょう。

テストメンテナンス

テストスクリプトの定期的なメンテナンスは、手動でも自動でも、コードの変更に対応して同期させるために不可欠です。この積極的なアプローチは、時間とコストを節約し、高品質なテストを保証します。

例えば、UI要素の識別属性（ID、クラス、XPathなど）が変更された場合、テストが失敗しないようにテストを適切に更新する必要があります。

しかし、戦略的なアプローチがないと、テストメンテナンスは困難で時間がかかるものになります。チームは新しいテストの作成や既存のテストの強化よりも、壊れたテストを修正することに多くの時間を費やすことになるかもしれません。

ここで、低メンテナンステストの価値が際立ちます。

低メンテナンスとは?

低メンテナンステストは、テスト対象のアプリケーションが変更されても安定し、最小限の更新しか必要としないように設計されています。これらのテストは、長期間にわたって壊れにくい構造になっており、頻繁で広範な修正を必要とせずにアプリケーションの変更に適応できるようになっています。

その目標は、メンテナンスの手間が少なくて済む、堅牢なテストスクリプトとフレームワークを作成することです。

Squishを使用した低メンテナンステストを達成するための5つの実践的な方法

Squishを使用した自動GUIテストにおいて低メンテナンスなテストを実現するための戦略と実践方法

以下は、Squishが自動GUIテストで低メンテナンスなテストを実現するための戦略と実践方法です：

方法 #1 Squishはデフォルトで低メンテナンスなテストケースを記録する

GUI要素を識別するための堅牢で安定したメカニズム
Squishのオブジェクトマップとオブジェクト名
Squishがテストケースを記録する際、ユーザーが操作するすべてのオブジェクトに対して、オブジェクトマップにシンボリック名とリアル名のペアを追加します。
テストケースを記録したり、Spyでオブジェクトを選択したりする際、Squishは各オブジェクトに対して自動的に名前を作成し、後でテストスクリプトなどで識別できるようにします。

この名前は、ほとんどのツールキットにおいて複数のプロパティ名（リアル名）で構成されています。これにより、アプリケーションの変更に柔軟で信頼性の高いオブジェクト識別が可能となります。

シンボリック名 vs リアル名
オブジェクトマップは、各オブジェクト名をそれぞれシンボリック名と呼ばれる名前に関連付けます。これにより、テストスクリプトはシンボリック名を使用してオブジェクト名を参照し、テストケースが実行される際にシンボリック名が自動的に参照されたオブジェクト名にマップされます。
Squishのオブジェクトマップは、テスト対象のアプリケーションがオブジェクトの階層や名前を変更した場合でも、テストスクリプトのメンテナンスを容易にするよう設計されています。テストスクリプトのコードベースの複数の箇所でオブジェクト名を繰り返し記述する代わりに、オブジェクトマップの概念は、各オブジェクト名を一度だけ中央で定義されたリポジトリで管理することです。

アプリケーションの変更があった場合、そのオブジェクト名を言及するオブジェクトマップのエントリーを変更するだけで済みます。シンボリック名はAUT（テスト対象アプリケーション）に依存しないため、修正の必要はありません。

そのため、すべてのテストスクリプトは変更されず、テストケースは従来通り再生されます。テストスクリプト内の各シンボリック名は、自動的に新しい更新されたオブジェクト名にマッピングされるためです。

重要なポイント：

- ツールよりも自分のAUT（テスト対象アプリケーション）を深く理解し、AUTの変更が起こりうる箇所を予測するために、AUTに関する知識を活用する
- 抽象化レベルをさらに高める
  - オブジェクトを識別するために異なるプロパティを使用する
  - オブジェクトを識別するためのプロパティの数を減らす

方法 #2 リアル名を編集して抽象化レベルを高める

Squishでテストを記録する際、ユーザーインターフェースの各要素はその特定の要素に固有の一連のプロパティで識別されます。要素を識別するために使用されるプロパティには、可視性、テキスト内容、親ウィンドウなどがあります。

テストを記録した後、オブジェクトマップを調べて、SquishがGUI要素を一意に識別するために選択したプロパティを確認することは役立ちます。

オブジェクトマップを調べる際に以下を心に留めておいてください：

- Squishがあまりにも多くのプロパティを選択してオブジェクトを過度に一意にしすぎることがあります。例えば、ボタンをテキスト内容に基づいて識別するとします。次のバージョンのテスト対象アプリケーションでこのテキストが変更されると、テストが失敗する可能性があります。
  しかし、テスト対象アプリケーションに精通した経験豊富な開発者やテスターとして、将来のバージョンでテキストが変更される可能性を予測し、さまざまなオブジェクトプロパティを使用してボタンを一意に識別することができます。
- 一方、Squishがあまりにも少ないプロパティを選択してリアル名が曖昧になる可能性もあります。例えば、特定のメニュー内の位置でボタンを識別する場合、テスト対象アプリケーションの新バージョンで同じビューに新しいボタンが追加されると、突然、2つのボタンが同じリアル名に一致する可能性があります。開発者やテスターとして、元のボタンのリアル名に追加の識別プロパティを加えて、将来の更新でも一意性が保たれるようにすることができます。

二つ目の方法では、オブジェクトマップを開いて異なるリアル名を調べ、不要なプロパティを削除したり追加すべきプロパティを見つけたりすることが役立つことが示されています。

特にテキストプロパティの場合、ワイルドカードを使用してリアル名のテキストプロパティを複数の異なるテキスト値にマッチさせることができます。

重要なポイント：
AUTの変更があると、テストが壊れることがあります。
これらの修正の手間を軽減するためには、次のような方法があります：

- オブジェクトマップの使用
- 関数の抽出
- テストロジックとテストデータの分離

方法 #3 コードの反復を削除する

三つ目の方法は、テストスクリプトにリアル名を直接埋め込まないことの重要性を強調しています。

ほとんどの場合、記録時にSquishはテストスクリプトで通常、シンボリック名を使用するため、リアル名はスクリプトから削除されます。

記録機能を使用せずに手動でスクリプトを作成する場合、オブジェクトマップのリアル名に対応するシンボリック名を活用することが重要です。

テストスクリプトでは、以下のようになるはずです：

```
tapObject(waitForObject(names.UIButton))
```

逆に、次のようになっている場合は間違っています：

```
tapObject(waitForObject({"type": "UIButton", "visible":
1, "window": {"type": "UIWindow", "visible": 1}}))
```

スクリプトでシンボリック名のみを使用することで、コードの反復が減少します。

重要なポイント

テストスクリプトでオブジェクトマッピングを使用する主な利点の1つは、オブジェクトのプロパティを変更または更新する必要がある場合、スクリプトコード内のオブジェクト参照のすべてのインスタンスを更新する代わりに、オブジェクトマップで修正を行うだけで済むことです。これにより、メンテナンスが大幅に簡素化され、テストスクリプトのエラーの可能性が減少します。

方法 #4 メソッドの抽出とテストケース間でのコードの共有

この実践的な例では、SquishのIDEが標準的なリファクタリングツールを提供することで、関数の作成とリファクタリングプロセスを効率化する方法を見ていきます。これにより、コードが常にメンテナンスされやすくなります。

さらに、Squishでは共有関数の作成が可能で、変更を単一のファイルに集約し、異なるテストスクリプトでこれらの関数を呼び出しても影響を受けないようにします。

これらの共有関数は、特定のテストスイート内の任意のテストケースで利用することができます。

1. スクリーンショットの検証

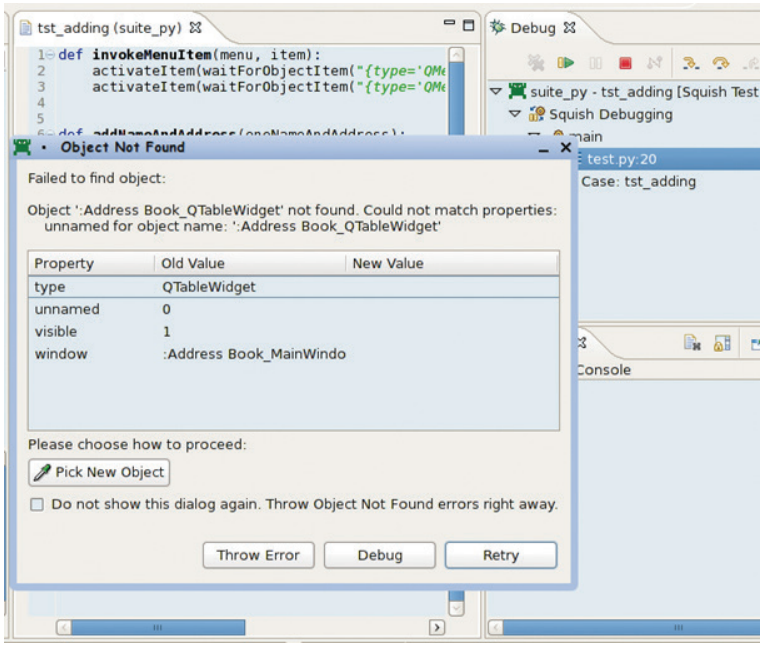
Squish IDEでは、ユーザーは新しいスクリーンショットを「期待される画面」として使用することができます。アプリケーションの変更によりスクリーンショットが失敗した場合、失敗した画面を「期待される画面」として使用することで、テストスクリプトを新しいGUIに適応させることができます。

また、Squishは使用ケースに応じて異なるスクリーンショット比較モードを選択することができます。例えば、閾値プロパティ、相関、ヒストグラムなどです。

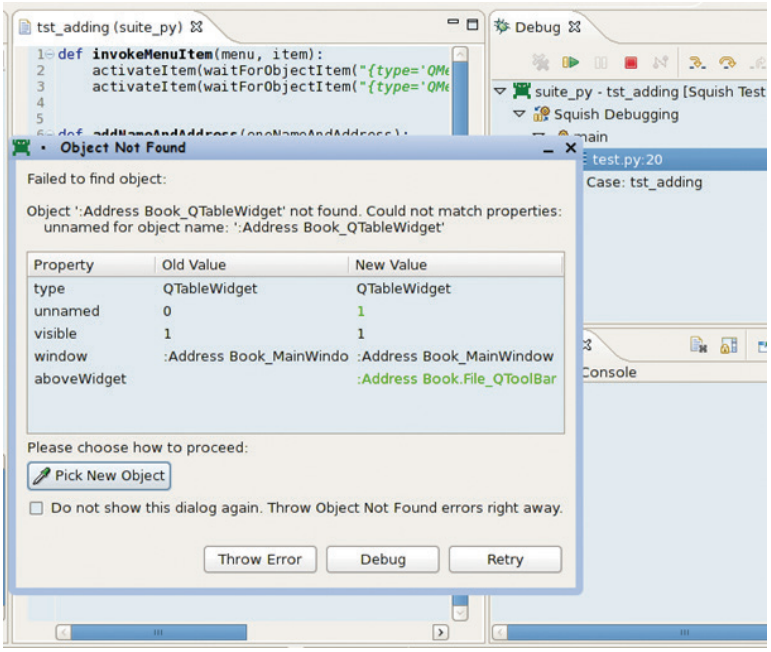
2. 再試行メカニズムの「Object Not Found」ダイアログ

「Object Not Found」（オブジェクトが見つかりません）の場合、Squish IDEでは別のオブジェクトを選択することができます。新しいオブジェクト名を使用して、テストスクリプトの実行を再試行することができます。

テストの実行中にwaitForObjectコマンドが指定された（またはデフォルトの）タイムアウトに達すると、「Object Not Found」（オブジェクトが見つかりません）ダイアログが自動的に開きます。



ダイアログには、ルックアップエラーが生成されたエラーメッセージと、ルックアップが実行されたオブジェクト名が表示されます。次の方法でエラーを解決できます。「Pick New Object」（新しいオブジェクトを選択）ボタンをクリックして、AUTからオブジェクトを選択し、そのプロパティをオブジェクトの検索に使用されたプロパティと比較します。これにより、プロパティがどのように変更されたか、そしてその変更がルックアップエラーの原因となったかを調査できます。



「Throw Error」(エラーをスロー)ボタンをクリックして、テストの実行を継続し、適切なエラーを生成します。この時点で実行を継続させると、テストスクリプトがルックアップエラーをキャッチして自己回復しない限り、テストが終了する可能性があります。

「Debug」(デバッグ)ボタンをクリックしてダイアログを閉じ、オブジェクトマップエディターで対応する名前を開きます。これにより、Squish IDEがテストデバッグパースペクティブに切り替わり、通常のデバッグツールを使用して問題をさらに調査できます。

「Retry」(再試行)ボタンをクリックしてオブジェクトのルックアップを再実行します。新しいオブジェクトを選択しなかった場合、デフォルトの名前を使用してオブジェクトの準備がデフォルトのタイムアウトよりも長くかかる場合にルックアップエラーが発生するかどうかを確認するのに役立ちます。新しいオブジェクトを選択した場合、新しいオブジェクト名がオブジェクトマップのシンボリック名の下に保存され、その新しい名前でルックアップが実行されます。

3. Squishはテストスクリプトに絶対座標を記録しない

これにより、記録された操作は画面上のウィジェットの絶対位置に依存せず、相対的にクリックが発生するべき位置を示します。記録された座標は絶対ではなく、ウィジェットの左上隅を基準としています。

4. スマートウェイトメカニズムの利用

waitFor()、waitForObject()、waitForItem() の同期化
Squishには非同期動作を処理するためのスマートウェイト機構があります。ハードコードされたスリープ文の代わりに、Squishは多くのwaitFor* APIを提供して、操作可能なオブジェクトの利用可能性を同期化します。

waitForObject()、waitForObjectExists()、waitForObjectItem()関数は、デフォルトでオブジェクトが表示され、使用可能になるのを待つために20000ミリ秒(20秒)待ちます。タイムアウトすると、キャッチ可能な例外が発生します。

方法 #5 データ駆動テスト

ソフトウェアテストでは、テストデータをテストロジックから分離することが有益です。この分離により、スケーラビリティと保守性が向上し、テストデータの変更がテストスクリプトの修正を必要とせず、プログラムエラーのリスクを低減することができます。スプレッドシートやデータベースなどの外部データソースを使用してこの分離を実現することができます。

このアプローチの利点は、テストデータが変更された場合、外部ソースを更新することでテストスクリプトを変更する必要がなくなることです。

Squishでは、データセットをシステムにインポートし、それをテストに挿入される値のソースとして利用することが可能です。Squishは.tsv、.csv 、.xls 、.xlsxなどの形式でデータをインポートする機能をサポートしています。

インポートプロセスは、Squish IDEを使用するか、ファイルマネージャやconsoleコマンドを介して手動で実行することができます。両方の方法について詳細に説明しますが、まずはSquish IDEを使用する方法について述べます。

自動テストメンテナンスの ベストプラクティス

これらの実践は、テストが最新であることを確保し、手動作業の必要性を減少させ、高品質なテストを維持するのに役立ちます。

自動化テストをスムーズに実行するためには、いくつかのメンテナンスのベストプラクティスに従うことが重要です。これらのプラクティスに従うことで、テストが最新の状態を保ち、手動作業の必要性を減らし、高品質なテストを維持することができます。

以下に、心に留めておくべき重要なプラクティスをいくつか挙げます：

1. テストスクリプトのバージョン管理の実装

バージョン管理システムは、テストスクリプトの変更を管理するための安全ネットのような存在です。これにより、チームは変更を追跡し、効果的に共同作業を行い、必要に応じて以前のバージョンに戻すことができます。

基本的に、バージョン管理は、テストスクリプトへの変更が記録され、確認や取り消しができるようにすることで、すべてを管理しやすく調整できることを保証します。

変更の履歴を追跡し、テストスクリプトの異なるバージョンを保存することで、問題が発生した場合でも安定したバージョンに簡単に戻ることができます。これにより、すべてのテストスクリプトが整理された方法で管理され、安心して更新することができます。

2. テスト自動化のためのコードレスツールの利用

コードレスのテスト自動化ツールは、テストスクリプトの作成とメンテナンスを容易にします。これらのツールには、視覚的なインターフェースや記録再生機能があり、プログラミングの深い知識の必要性を低減します。

技術的な要素が少ないテストほど、チームメンバーがテストの作成とメンテナンスに協力しやすくなります。これにより全体的な生産性とテストカバレッジが向上し、テスターやビジネスアナリスト、専門家が特化した自動化エンジニアに依存せずに自動化テストを作成・管理できるようになります。

さらに、異なるQAエンジニアは異なるコーディング文化や命名規則を持っています。視覚的なツールは混乱を避け、テストの品質を維持するのに役立ちます。

3. 継続的インテグレーション／継続的デリバリー（CI/CD）の統合

コードが変更されるたびに自動的にテストが実行されることを確認することは重要です。これにより、開発者は迅速なフィードバックを受け取り、問題が再発する可能性を減らすことができます。開発プロセスでのテストの自動化により、テストが常にコードの変更に合わせていることが保証されます。

CI/CDツールは、コードが更新されるたびにテストを実行する便利な助手のような存在です。CI/CDの統合を導入することは、開発プロセスの早い段階で問題を検出し、それが大きな問題にエスカレートする前に欠陥の解決を簡素化するのに役立ちます。

4. テストデータ管理戦略の策定

信頼性の高い繰り返し可能な自動化テストを確保するためには、テスト情報を効果的に管理する戦略が必要です。

これには、繰り返し使用できる高品質なテスト情報セットの作成が含まれます。新しいテスト情報の生成、実際の情報のマスキング、またはダミーデータの使用など、さまざまな方法があります。これにより、テスト要件を満たしつつ個人情報を公開せずにテストを実施します。さらに、異なるテスト環境間での一貫性の維持も重要です。

適切なテストデータ管理により、不整合を排除し、データに起因するテストの失敗の可能性を減少させ、予測可能なテスト条件を提供します。

5. 自動監視とアラート設定の導入

ツールを使用してテスト実行を監視することで、問題を迅速に特定し、即座に修正することができます。自動アラートは、問題が大きなものにエスカレートする前に対処するのに役立ちます。

自動監視ツールを設定してテスト結果やパフォーマンス数値を追跡することで、通常のパラメータからの逸脱があれば通知を受けることができます。これは、一歩先を見据えるための方法の一つです。

6. チームワーク

テストをスムーズに実行するためにはチームワークが不可欠です。チームメンバー同士のコミュニケーションと知識共有を促進することで、問題の議論や解決策の共有、ベストプラクティスの文書化が可能になります。

開発チームとテストチームの間で定期的に予定された会議や振り返りを通じて、テストのメンテナンスに焦点を当て、改善点を特定し、持続的な改善文化を育成することが重要です。

テストの耐久性向上

Squishは、メンテナンスに必要な労力を削減しつつ、堅牢で持続性のあるテストを作成します。戦略的な実践を取り入れることで、テストの耐久性をさらに向上させることができます。このワイトペーパーからの主な要点は以下の2つです：

1. 抽象化レベルの向上するために：

- オブジェクトのプロパティを最小限に抑える - GUI要素を識別するために必要な最小限のオブジェクトプロパティを使用することが重要です。これにより、テストがテスト対象のアプリケーション（AUT）で変更があった際に壊れる可能性が低くなります。各オブジェクトを一意に識別するための必要なプロパティに焦点を当てることで、より頑健なテストを作成します。
- ワイルドカードを使用する - オブジェクトのプロパティや値にワイルドカードを実装することで、テストの適応性が向上します。ワイルドカードを使用すると、プロパティ値のわずかな変動に対応でき、AUTの変更があってもテストが失敗しにくくなります。この柔軟性は、頑健なテストスイートを維持するために重要です。
- ドメイン知識を活用する - あなたはツールよりもAUTをよく理解しています。AUTに関する深い理解を活用して、リアル名に保存されているオブジェクトプロパティを洗練させます。関連性のないプロパティを特定して削除するか、要素を一意に識別するためのより安定したプロパティに変更します。これにより、テストが正確であり、アプリケーションのインターフェースの変更にも耐えられるようになります。
- 待機関数を利用する - ハードコードされた待機間隔をwaitForObject()のようなインテリジェントな待機関数に置き換えることで、不必要な遅延を回避し、タイミングの問題によるテスト失敗の可能性を減少させます。この関数は、オブジェクトが利用可能になるまで待機し、その後に進行するため、AUTが準備完了したときにのみテストが進むようにします。

2. テストが壊れたときの修正作業を簡素化するために:

- シンボリック名を使用してリアル名を参照する - テストスクリプトでシンボリック名を使用してリアル名を参照することは、保守性のために不可欠です。シンボリック名は実際のオブジェクトプロパティのエイリアスとして機能し、1か所でリアル名を更新できます。その更新はテストスクリプトのすべての参照に自動的に反映され、アプリケーションに変更があった際のテスト修正の労力を大幅に削減します。
- 反復スクリプトコードを別の関数に抽出する - テストスクリプトをモジュール化することで、冗長性を最小限に抑え、共通の操作を一元化します。変更が必要な場合は、関数を1か所で更新するだけで、その関数を利用するすべてのテストで一貫性が保たれ、メンテナンスが容易になり、エラーも少なくなります。
- データ駆動アプローチを利用する - テストデータをテストロジックから分離し、テストデータを外部ファイルに保存してテストパラメータを駆動させます。このアプローチにより、テストスクリプト自体の変更が最小限に抑えられ、スクリプトの更新時にエラーを導入する可能性が減少します。また、テストロジックを変更することなく、テストデータのスケーリングや更新が容易になります。

これらの実践により、自動化テストの耐久性と保守性が大幅に向上します。結局のところ、GUIテストの自動化の目的は、人間のテスターがより複雑で重要なタスクに集中できるようにすることではありませんか？

Qt Quality Assurance