



Fakultät für Informatik

Studiengang Informatik

Automatisierte UI Tests für Windows Qt-Anwendungen

Software Qualitätssicherung

von

Dominik Sieberer

Stand der Version: 12.06.2022

Abstract

TODO

Inhaltsverzeichnis

1	Automatisierte UI Tests	1
1.1	Einleitung	1
1.2	Anwendung	2
1.3	Anforderungen	4
2	Squish	5
2.1	Squish	5
2.2	Durchführung der Tests	7
2.3	Tests	9
3	Qt Test	11
3.1	Einführung Qt Test	11
3.2	Durchführung der Tests	13
3.3	Tests	15
4	Vergleich	20
4.1	Squish	20
4.2	Qt Test	20
4.3	Vergleich	20
5	Fazit	21
	Literaturverzeichnis	22

Abbildungsverzeichnis

1.1 Anwendung	3
2.1 Squish IDE	6
2.2 Squish IDE	7
2.3 Squish IDE	8

1 Automatisierte UI Tests

1.1 Einleitung

Automatisiertes UI-Testing bietet zahlreiche Vorteile in der Softwareentwicklung. Einige davon sind:

- Automatisierte Tests haben vergleichbare Ergebnisse und sind weniger anfällig für menschliche Fehler
- Höhere Testabdeckungsrate der Software
- Erhöhte Testabdeckung fördert debugging
- Erstellter Testcode kann wiederverwendet werden, wodurch das Testen leicht skalierbar wird
- Automatisierte Tests sind im Vergleich zu manuellen Tests viel schneller
- Sie sind Kosten- und Zeiteffizienter als manuelle Tests

Der Global Quality Report zeigt, dass mehr als 60 % der Unternehmen aufgrund der höheren Testabdeckung durch Testautomatisierung in der Lage sind, Fehler schneller zu erkennen. Darüber hinaus stellten 57 % der Befragten fest, dass die Wiederverwendung von Testfällen durch den Einsatz von Automatisierung zunahm. Automatisiertes Testing gilt als de facto Standard in der Software Entwicklung [Cap].

Es äußerst wichtig, das richtige Gleichgewicht zwischen manuellen und automatisierten Tests zu finden. Jedes Projekt ist einzigartig, und es gilt, verschiedene Aspekte wie wirtschaftliche Machbarkeit, zeitliche Beschränkungen und die Art der durchzuführenden Tests zu berücksichtigen. Hier muss jedes Teams fundierte Entscheidungen treffen. Dieses Dokument dient als Entscheidungsgrundlage um ein Framework zu wählen, mit dem automatisierte UI Tests für Windows Qt-Anwendungen durchgeführt werden. Die Zielgruppe dieses Dokuments sind Softwareentwickler die mit dem C++ Qt Framework vertraut sind.

1.2 Anwendung

Die Anwendung, die zum Vergleich der Frameworks verwendet wird, ist eine stark vereinfachte Version der Anlagensteuerungssoftware der Ambright GmbH.

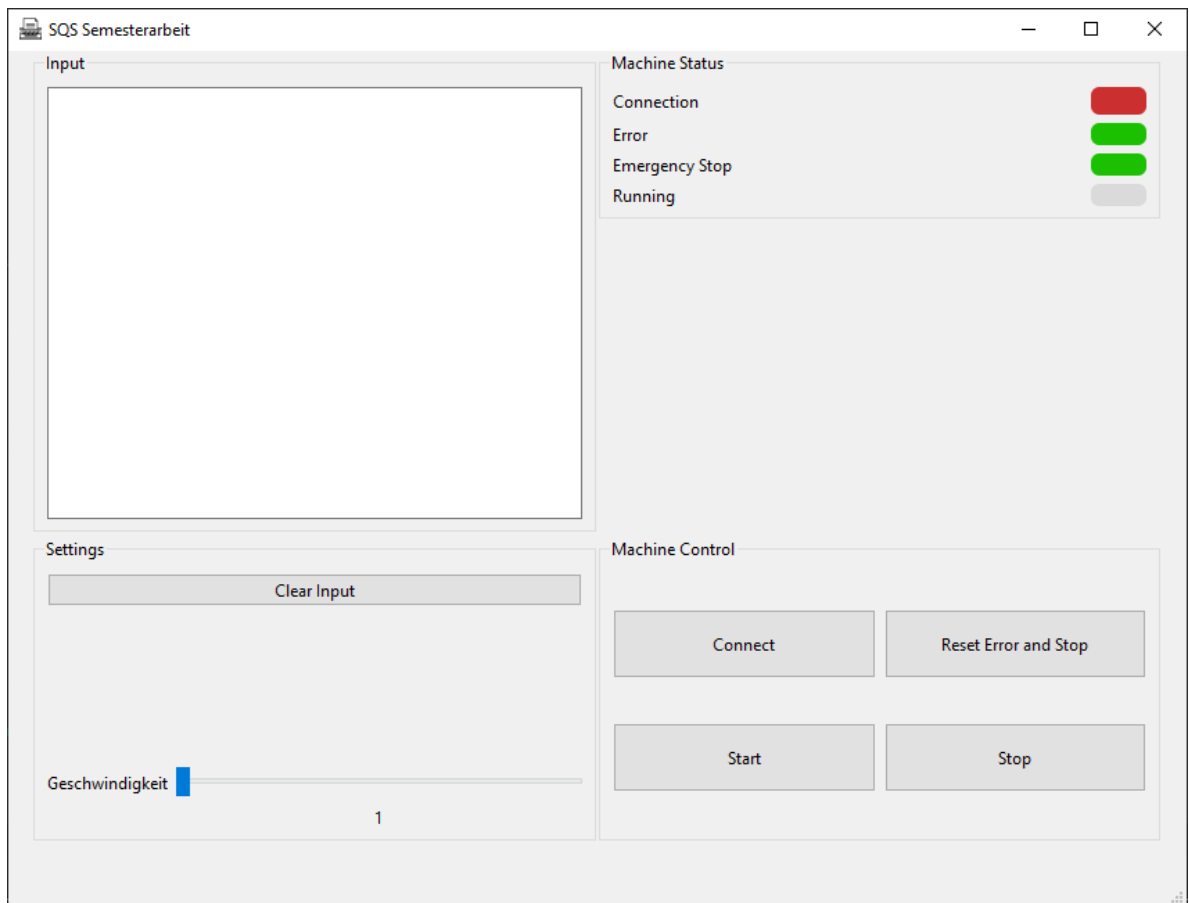


Abbildung 1.1 Anwendung an der die Tests Evaluiert werden

Die Anwendung (Abbildung 1.1) lässt sich in 4 Bereiche Unterteilen. Der erste Bereich, links oben, ist ein Text Feld. Dieses Text Feld kann mit User Input gefüllt werden. Der Input wird durch drücken des Start Buttons im vierten Bereich, rechts unten, evaluiert. Wenn der Input 'disconnect' oder 'error' beinhaltet werden entsprechende Methoden ausgeführt und der Status der Maschine verändert.

Im zweiten Bereich, rechts oben, sind 4 Statusanzeigen zu sehen. Die Statusanzeigen signalisieren durch Farben den Status der Maschine. Wenn eine aktive Verbindung besteht ist die Anzeige der Connection grün, andernfalls rot. Falls beim ausführen des Inputs aus dem ersten Bereich ein Fehler auftritt wird die Error Anzeige rot. Wenn kein Fehler auftritt ist sie grün eingefärbt. Die Emergency Stop Anzeige zeigt an wenn der Nothalt Schalter gedrückt wurde mit roter Farbe. Falls der Nothalt Schalter nicht gedrückt wurde ist die Anzeige grün. Die letzte Anzeige ist grün wenn gerade ein Script aus dem Inputfeld ausgeführt wird, und bleibt grün sofern keine Fehler auftreten oder Stop gedrückt wird.

Im dritten Bereich, links unten, befinden sich Settings zur Maschine. Mit dem Button Clear Input werden alle Zeichen aus dem Input Feld aus dem ersten Bereich entfernt. Durch den Slider kann die Geschwindigkeit innerhalb des Wertebereichs 1 bis 10 eingestellt werden. Der eingestellte Geschwindigkeitswert wird durch ein Label unter dem Slider angezeigt.

Im letzten Bereich, rechts unten, befinden sich 4 Buttons. Der erste Button stellt eine Verbindung her. Der zweite Button setzt den Error Status zurück. Die letzten beiden Buttons sind zum Starten und Stoppen der Maschine.

Tabelle 1.1 Anforderungstabelle

ID	Anforderung	Metrik
01	Kosten	Kosten sollen so gering wie möglich sein
02	Lizenz	Bestenfalls GPL oder vergleichbar
03	Open Source	Source-Code öffentlich verfügbar
04	Integration in vorhandene Pipeline	Wie viel Zeit benötigt es die Tests in die Pipeline zu integrieren
05	Lesbarkeit	Kann der Testcode von Entwicklern gelesen werden
06	Aufwand	Benötigte Zeit zum erstellen eines Tests
07	Änderbarkeit	Benötigte Zeit zum ändern eines Tests
08	Support / Community	Aktivität und Größe der Community (Stackoverflow, eigenes Forum, ...)
09	Updates	Update-Zyklus des Frameworks
10	Reife / Robustheit	Stabilität des Frameworks
11	In Qt Editor ausführbar	Sind die Tests im Qt Editor ausführbar
12	Abdeckung der Qt Features	Mögliche Abdeckung der mit Qt erstellbaren Widgets
13	Laufzeit	Wie viel Zeit beansprucht die Ausführung eines Tests
14	Dokumentation	Umfang und Qualität der Dokumentation der Tests
15	Auswertbarkeit der Ergebnisse	Lassen sich die Ergebnisse automatisiert auswerten

1.3 Anforderungen

Die Folgende Tabelle gibt einen Überblick über die Anforderungen, welche an die UI Testing Frameworks gestellt werden. Anhand dieser Anforderungen werden die gewählten Frameworks verglichen und eine Entscheidung getroffen. Jede Anforderung ist mit einer ID versehen. Diese IDs werden im Dokument zur Referenzierung verwendet.

TODO: Erklärung zu jeder Tabellenzeile (Tabelle 1.1)

2 Squish

2.1 Squish

Squish ist ein plattformübergreifendes UI- und regression test Tool, das eine Vielzahl von Anwendungen mit UIs testen kann. Unter anderem können Qt-, Java-, Web- und Mobile-Anwendungen getestet werden. Es wird von Froglogic entwickelt und gepflegt. Froglogic hat seinen Hauptsitz in Hamburg und bietet das Tool Squish seit 2003 an[Abo22]. Squish erlaubt es sowohl die UI Tests in der bereitgestellten IDE (Abbildung 2.1) aufzunehmen und durch die Aufnahme Code zu generieren, als auch den Testcode selbst zu schreiben. Der Testcode kann in JavaScript, Perl, Python, Ruby oder Tcl geschrieben, bzw. generiert werden.

2 Squish

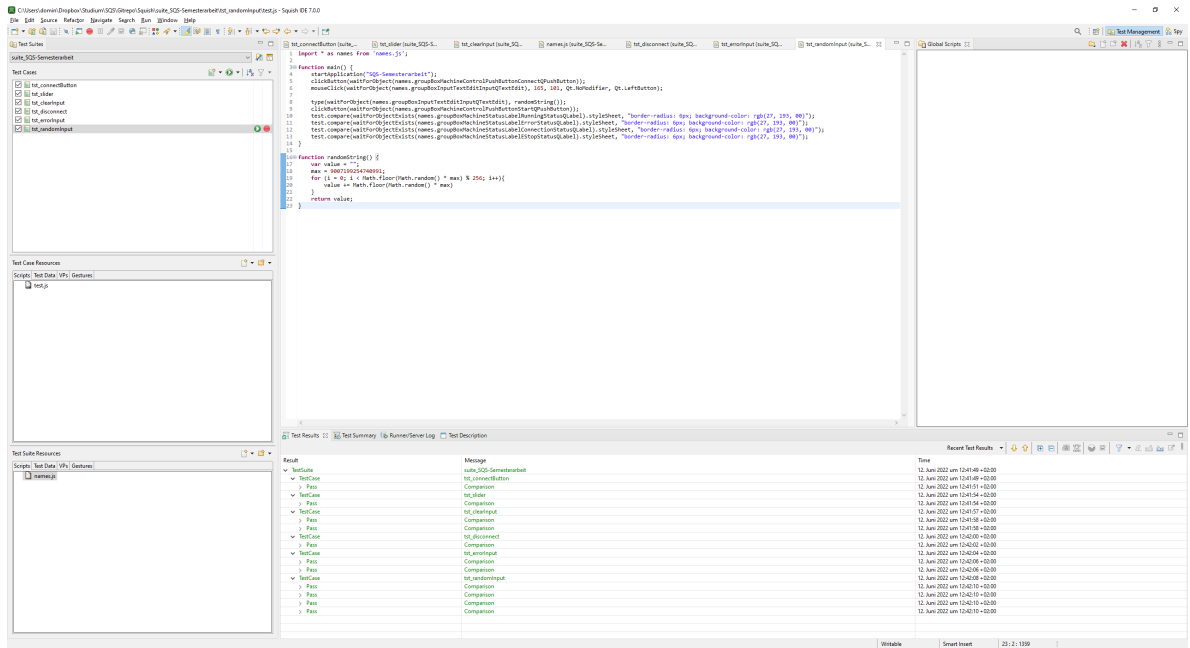


Abbildung 2.1 Squish IDE in der die Tests erstellt werden.

Als erster Schritt muss eine Test Suite angelegt werden. In der Test Suite wird das GUI toolkit, die Script Sprache der Tests sowie der Pfad zu der zu testenden Anwendung festgelegt. Das GUI toolkit legt fest für welche Art von Anwendung die Tests sind. In diesem Fall wurde Qt festgelegt. Zu einer Test Suite können beliebig viele Test Cases hinzugefügt werden. Die ausgewählte Test Suite sowie die Test Cases sind in der Squish IDE in der linken oberen Ecke angesiedelt (Ausgeschnitten in Abbildung 2.3). In der Mitte befindet sich der Script Editor, mit dem die erstellten Test Cases bearbeitet werden können. Unter dem Script Editor ist eine Anzeige bei der Zwischen den Testergebnissen, einer Zusammenfassung der Testergebnisse, den durch die Tests generierten Logs und der Beschreibung der Test Cases gewechselt werden kann. Um einen Test Case aufzunehmen muss der rote 'record' Button (Abbildung 2.3)) gedrückt werden. Mit dem Start der Aufnahme wird sowohl die Anwendung als auch eine Control Bar (Abbildung 2.2)) zum steuern der Aufnahme geöffnet. Mit der Control Bar kann die Aufnahme beendet werden sowie Schritte zur Verifizierung als auch einfügen Nutzereingaben für die Anwendung in den Test eingefügt werden. Zur Verifizierung gibt es Verschiedene möglichkeiten. In dieser Arbeit wurde zur Verifizierung die Attribute einzelner UI Elemente abgeglichen. Andere Möglichkeiten wären beispielsweise abgleiche mit Screenshots oder Erkennung durch OCR.

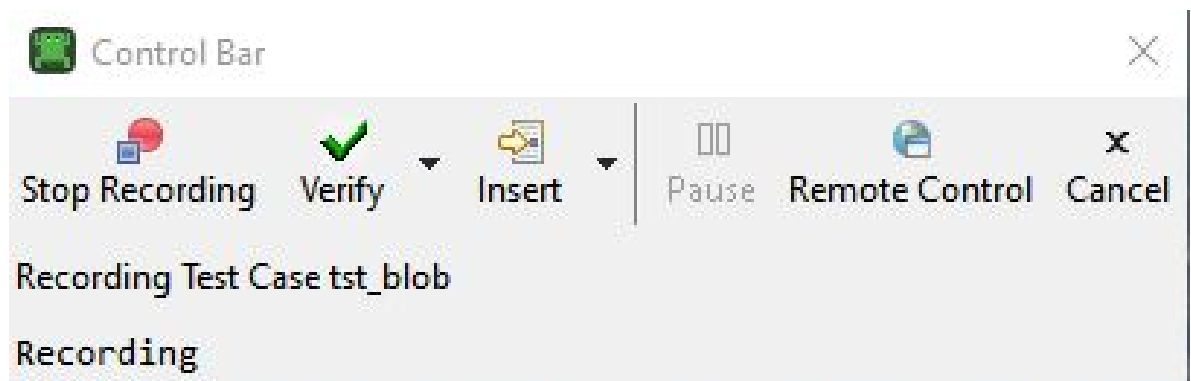


Abbildung 2.2 Control Bar mit der die Aufnahme eines Tests gesteuert wird

2.2 Durchführung der Tests

Um einen einzelnen Test oder die gesamte Test Suite zu starten gibt es verschiedene Möglichkeiten. UI Zum Starten Automatischer Aufruf zum Starten TODO

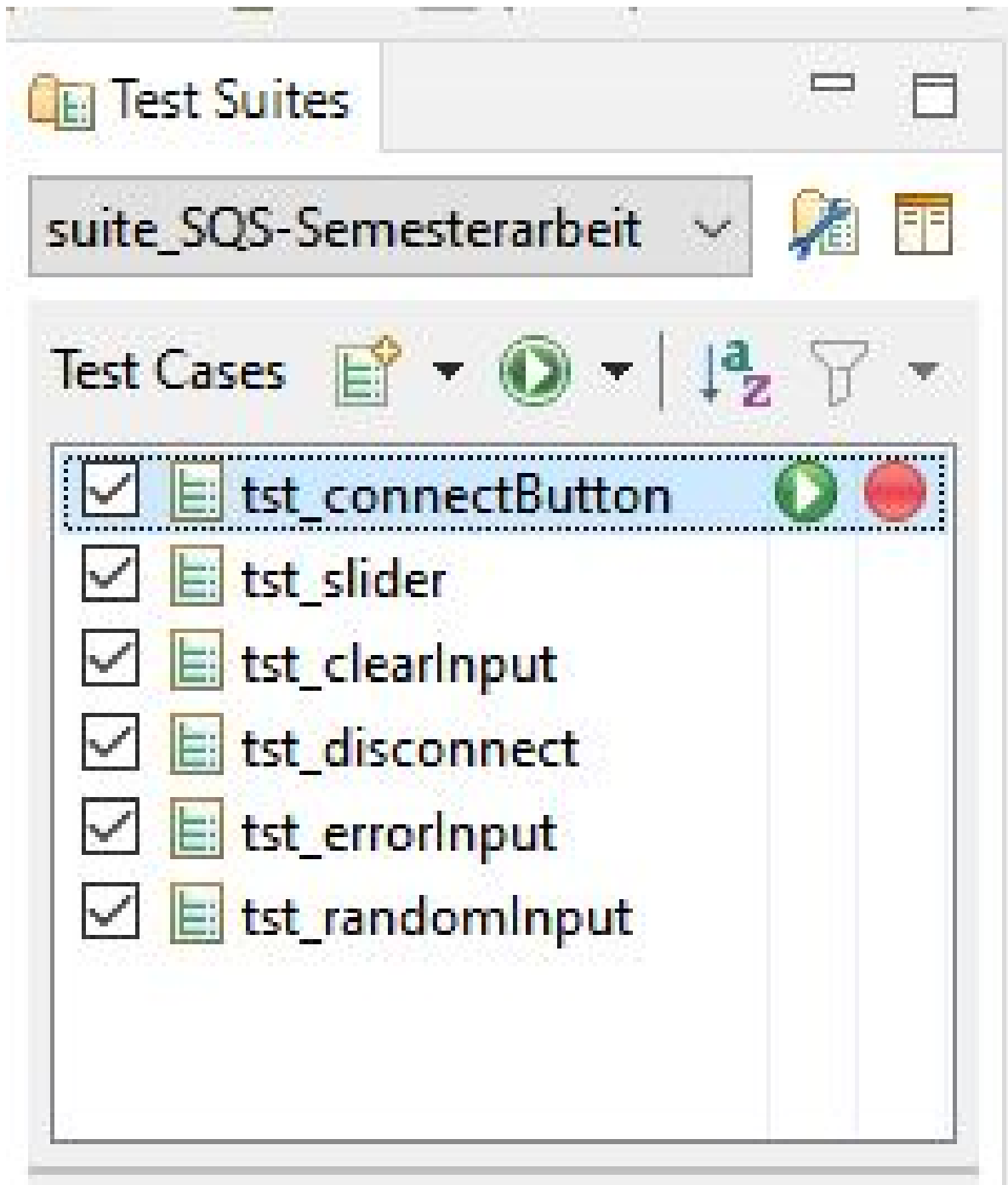


Abbildung 2.3 Squish IDE, Ausschnitt der Test Suite und Test Cases

2.3 Tests

Mit dem Test `tst_connectButton` wird überprüft ob durch drücken des Connect Buttons die Anzeige Connection grün wird. Dazu wird zuerst mit der Funktion `startApplication()` die Anwendung gestartet und danach mit der Funktion `clickButton()` der Connect Button angeklickt. Um zu überprüfen ob die Connected Anzeige grün ist wird die Funktion `test.compare()` aufgerufen.

Listing 2.1 `tst_connectButton`

```

1  function main() {
2      startApplication("SQS-Semesterarbeit");
3      clickButton(waitForObject(
          names.groupBoxMachineControlPushButtonConnectQPushButton)
        );
4      test.compare(waitForObjectExists(
          names.groupBoxMachineStatusLabelConnectionStatusLabel)
        .styleSheet, "border-radius: 6px; background-color: rgb
          (27, 193, 00)");
5  }
```

Der zweite Test testet nach starten der Anwendung ob durch bewegen des Sliders das Label, dass die eingestellte Geschwindigkeit anzeigt wird korrekt aktualisiert wird. Der Slider wird durch die Funktion `scrollTo()` auf die Position 6 bewegt. Darauf folgend wird mit `test.compare()` verglichen ob das Label den richtigen Wert anzeigt.

Listing 2.2 `tst_slider`

```

1  function main() {
2      startApplication("SQS-Semesterarbeit");
3      scrollTo(waitForObject(
          names.groupBoxSettingsHorizontalSliderSpeedQSlider), 6);
4      test.compare(waitForObjectExists(
          names.groupBoxSettingsLabelSpeedValueQLabel).text, "6");
5  }
```

Durch den Test Case `tst_errorInput` wird nach Anwendungsstart in das input Feld geklickt und dort der String 'error' über die Funktion `type()` als Keyboard Input eingegeben. Darauf folgend wird der Start Button gedrückt um die Maschine mit dem im Input stehenden Daten zu starten. Da dort 'error' steht wird erwartet, dass die Maschine in den Zustand Error wechselt. Um dies zu überprüfen wird das aussehen der Error Anzeige durch `test.compare()` abgeglichen.

Listing 2.3 `tst_errorInput`

```

1  function main() {
2      startApplication("SQS-Semesterarbeit");
3      clickButton(waitForObject(
          names.groupBoxMachineControlPushButtonConnectQPushButton));
```

2 Squish

```
4  mouseClicked( waitForObject(
    names.groupBoxInputTextEditInputQTextEdit), 234, 156,
    Qt.NoModifier, Qt.LeftButton);
5  type( waitForObject( names.groupBoxInputTextEditInputQTextEdit),
    "error");
6  clickButton( waitForObject(
    names.groupBoxMachineControlPushButtonStartQPushButton));
7  test.compare( waitForObjectExists(
    names.groupBoxMachineStatusLabelErrorStatusLabel)
    .styleSheet, "border-radius: 6px; background-color: rgb
    (203, 47, 47)");
8  clickButton( waitForObject(
    names.groupBoxMachineControlPushButtonResetErrorAndEStopQPushButton
    ));
9  test.compare( waitForObjectExists(
    names.groupBoxMachineStatusLabelErrorStatusLabel)
    .styleSheet, "border-radius: 6px; background-color: rgb(27,
    193, 00)");
10 }
```

3 Qt Test

3.1 Einführung Qt Test

Qt ist ein cross-platform Framework zur Entwicklung von Desktop-, Embedded- und Mobile-Anwendungen. Qt ist keine eigenständige Programmiersprache, sondern ein in C++ geschriebenes Framework. Ein Präprozessor, der MOC (Meta-Object Compiler), wird verwendet, um die C++-Sprache um Funktionen wie Signals und Slots zu erweitern. Vor dem Kompilieren analysiert der MOC die in Qt-erweitertem C++ geschriebenen Quelldateien und erzeugt daraus standardkonforme C++-Quelldateien. Somit können das Framework selbst und die Anwendungen/Bibliotheken, die es verwenden, von jedem standardkonformen C++-Compiler kompiliert werden. Qt ist unter kommerziellen sowie open-source Lizenzen verfügbar. Das Framework befindet sich seit 1995 in Entwicklung und erhält seitdem regelmäßig Updates [Doc22a]. Auf Stackoverflow gibt es derzeit 82.522 Fragen mit dem Tag Qt [Qt 22]. Zudem weist das Qt Forum eine solide Aktivität auf. Daher ist von einer Mittelgroßen, aktiven Community auszugehen. Für dieses Projekt wird die Qt Version 5.15 verwendet.

Qt Test ist ein Teil des Qt Frameworks. Es bietet Klassen für Unit-Tests von Qt-Anwendungen und -Bibliotheken. Alle öffentlichen Methoden befinden sich im QTest-Namensraum. Darüber hinaus bietet die Klasse QSignalSpy eine einfache Introspektion der Signale und Slots von Qt, und der QAbstractItemModelTester ermöglicht das nicht-destruktive Testen von Elementmodellen. Dadurch können auch UIs getestet werden. Es lassen sich Maus und Tastatur Eingaben durch ein von der Anwendung unabhängiges Eventsystem simulieren [Doc22b].

Beispiel für einen Test:

Listing 3.1 Hello World Beispiel

```
1 void TestGui::testGui()  
2 {  
3     QLineEdit lineEdit;  
  
5     QTest::keyClicks(lineEdit, "hello world");  
  
7     QCOMPARE(lineEdit.text(), QString("hello world"));  
8 }
```

Die Methode TestGui::testGui() erstellt ein Eingabefeld. Durch die Methode QTest::keyClicks() wird das als Parameter übergebene QWidget fokussiert und der als Parameter übergebene String als Tastatur Eingabe dem QWidget übergeben. Mit dem Makro QCOMPARE() werden zwei Werte verglichen. Wenn die beiden übergebenen Werte den selben Inhalt haben wird das Programm fortgeführt. Sollte dies nicht der Fall ist wird der eine Fehlermeldung im Log

ausgegeben und die Methode beendet. Für Mauseingaben gibt es mehrere Methoden. Unter anderem die Methoden `QTest::mouseMove()` und `QTest::mouseClick()`. `QTest::mouseMove()` bewegt den Mauszeiger auf die übergebene Position, ausgehend von dem Übergebenen `QWdiget`. `QTest::mouseClick()` führt einen Mausklick auf die übergebene Position für die übergebene Zeit aus.

3.2 Durchführung der Tests

Die Tests werden durch folgende Zeilen in der main Methode aufgerufen:

Listing 3.2 main.cpp

```

1 #ifdef QT_DEBUG
2     TestSuite::executeTests();
3 #endif

```

Ob das QT_DEBUG flag gesetzt ist hängt von den Parametern mit denen das Programm aufgerufen wird ab.

Die Methode TestSuite::executeTests() führt alle registrierten Tests aus und meldet, ob sie erfolgreich waren oder nicht.

Listing 3.3 testsuite.cpp

```

2 QList<QObject *> *TestSuite::m_suites = nullptr;

4 TestSuite::TestSuite()
5 {
6     static bool init = true;
7     if(init){
8         m_suites = new QList<QObject *>;
9         init = false;
10    }
11    m_suites->append(this);
12 }

14 void TestSuite::executeTests()
15 {
16     if(m_suites == nullptr){
17         qDebug() << "No tests available";
18         return;
19     }

21     int failedTests = 0;
22     QList<QObject* >::iterator it = m_suites->begin();
23     for(; it != m_suites->end(); it++){
24         failedTests += QTest::qExec(*it);
25     }
26     qDebug() << "-----";
27     qDebug() << "Finished Testing";
28     qDebug() << "Test Files: " << QString::number(m_suites->length())
29         );
29     qDebug() << "Failed Tests: " << QString::number(failedTests);
30     qDebug() << "-----";

```

```
32     if (failedTests != 0) {  
33         exit(1);  
34     }  
35 }
```

Alle Testklassen registrieren sich durch eine statische instanz selbst in der `QList<QObject*> m_suites`. Mit einem iterator wird durch die Liste `m_suites` iteriert und jede Testklasse mit `QTest::qExec()` aufgerufen. `QTest::qExec()` führt alle Methoden in der Klasse auf die als private slots deklariert sind. Sollten darunter die Methoden `initTestCase()` oder `init()` deklariert sein, so werden diese vor jedem Test einzelnen Test, beziehungsweise vor allen Tests ausgeführt. Analog geschieht dies wenn die Methoden `cleanupTestCase()` oder `cleanup()` deklariert sind nach den Tests. `QTest::qExec()` gibt als Rückgabewert '0' falls keiner der Tests fehlgeschlagen ist, andernfalls die Anzahl an fehlgeschlagenen Methoden. Die Ergebnisse der Tests werden auf der Konsole ausgegeben und sollte ein oder mehrere Tests fehlgeschlagen sein wird das Programm beendet.

3.3 Tests

Die Klasse `QtGuiTest` enthält 6 Tests, die den gleichen Ablauf haben wie die für Squish definierten Tests. Die Methode `initTestCase()`, die vor dem Durchlauf aller Tests aufgerufen wird, `cleanupTestCase()` welche nach dem Durchlauf aller Tests aufgerufen wird sowie zwei kleinen Methoden welche öfter in den Tests verwendet werden.

Listing 3.4 `qtguitest.cpp`

```

1  class QtGuiTest : public TestSuite
2  {
3      Q_OBJECT

5      friend class MainWindow;

7      private Q_SLOTS:
8      void initTestCase();
9      void cleanupTestCase();

11     void testConnectButton();
12     void testSlider();
13     void testClearInput();
14     void testDisconnect();
15     void testErrorInput();
16     void testRandomInput();

18     private:
19     /**
20      * @brief establishConnection: verifies that the machine is
21         connected
22      * using the GUI
23      * @return true if there is a established connection, otherwise
24         false
25      */
26     bool establishConnection();

27     /**
28      * @brief connectedAndRunning: Checks if the Machine is
29         connected and
30      * running
31      * For Integrity it is also verified that there are no active
32         errors
33      * or that the emergency stop is activated
34      * @return true if the machine is connected and running,
35         otherwise
36      * false
37      */
38     bool connectedAndRunning();

```

```

36 private:
37     QWidget *m_window;
38 };

```

Zudem ist die Klasse MainWindow als friend class markiert um auf die UI Elemente des MainWindows in den Tests zugreifen zu können. Ein Pointer auf eine Instanz von MainWindow wird in der Membervariable m_window gespeichert.

In QtGuiTest::initTestCase() wird die Membervariable m_window initialisiert und das Window angezeigt. Sollte das anzeigen des Windows fehlschlagen wird mit dem Makro QVERIFY2() ein Fehler ausgegeben.

Listing 3.5 qtguitest.cpp

```

1 void QtGuiTest::initTestCase()
2 {
3     MainWindow *mainWindow = new MainWindow();
4     m_window = mainWindow->window();

6     mainWindow->show();

8     while (!m_window->isVisible()) {
9         QTest::qWait(200);
10    }

12    QTest::qWait(500);

14    QVERIFY2(m_window, "Window could not be crated");
15 }

```

Die Methode QtGuiTest::testConnectButton() wird der Mauszeiger auf den Connect Button im vierten Bereich des Fensters bewegt und daraufhin angeklickt. Nach dem Klick wird überprüft ob die Anzeige im zweiten Bereich der Anwendung anzeigt das eine aktive Verbindung besteht.

Listing 3.6 qtguitest.cpp

```

1 void QtGuiTest::testConnectButton()
2 {
3     QPushButton *button = m_window->findChild<QPushButton *>("
4         pushButton_connect");
5     QVERIFY2(button, "Connect Button not found");

6     QTest::mouseMove(button, QPoint(20, 10));

8     QTest::mouseClick(button, Qt::LeftButton, Qt::NoModifier,
9         QPoint(20, 10));

```

```

10  QTest::qWait(250);

12  QLabel *connectionStatusLabel = m_window->findChild<QLabel *>
    ("label_connectionStatus");
13  QVERIFY2(connectionStatusLabel, "Connect Status Label not
    found");

15  QVERIFY2(connectionStatusLabel->styleSheet().contains("
    background-color: rgb(27, 193, 00)"), "Color of
    label_connectionStatus is not Green");
16  }

```

Durch `QtGuiTest::testSlider()` wird der Mauszeiger auf die Nullposition des Geschwindigkeitssliders im dritten Bereich der Anwendung bewegt. Durch den Aufruf von `QTest::mousePress()` wird die linke Maustaste gedrückt gehalten. Durch das Bewegen der Maus um 50 Pixel nach rechts sollte der Geschwindigkeitsslider auf dem Wert 6 stehen, was mit dem Makro `QCOMPARE()` überprüft wird. Ebenso wird geprüft ob das Label das den Wert des Sliders dem Nutzer anzeigt ebenso den Wert 6 anzeigt.

Listing 3.7 qtguitest.cpp

```

1  void QtGuiTest::testSlider()
2  {
3      QSlider *slider = m_window->findChild<QSlider *>("
        horizontalSlider_speed");
4      QVERIFY2(slider, "Slider not found");

6      QTest::mouseMove(slider, QPoint(0, 0));

8      QTest::mousePress(slider, Qt::LeftButton, Qt::NoModifier,
        QPoint(0, 0));
9      QTest::mouseMove(slider, QPoint(50, 0));

11     QTest::mouseRelease(slider, Qt::LeftButton, Qt::NoModifier,
        QPoint(0, 0));

13     QCOMPARE(slider->value(), 6);

15     QLabel *speedLabel = m_window->findChild<QLabel *>("
        label_speedValue");
16     QVERIFY2(speedLabel, "Speed Label not found");

18     QCOMPARE(speedLabel->text(), "6");
19 }

```

Mit der Methode `QtGuiTest::testErrorInput()` wird der `QString` 'error' durch die Methode `QTest::keyClicks()` in das Input Feld geschrieben. Durch drücken des Start Buttons triggert der Input 'error' den Machine State 'Error', worauf die Error Anzeige rot wird. Mit drücken

des Reset Error and Stop Button wird dieser Status wieder zurückgesetzt und die Anzeige aktualisiert. Dieses Verhalten wird durch den Testcase `QtGuiTest::testErrorInput()` vollständig abgedeckt.

Listing 3.8 `qtguitest.cpp`

```

1 void QtGuiTest::testErrorInput()
2 {
3     QVERIFY2(establishConnection(), "Connection could not be
        established");

5     QTextEdit *textField = m_window->findChild<QTextEdit *>("
        textEdit_input");
6     QVERIFY2(textField, "Input Field not found");

8     QTest::mouseMove(textField, QPoint(50, 50));
9     QTest::mouseClick(textField, Qt::LeftButton, Qt::NoModifier,
        QPoint(50, 50));

11    QString inputString = "error";
12    QTest::keyClicks(textField, inputString);
13    QTest::qWait(50);
14    QCOMPARE(textField->toPlainText(), inputString);

16    QPushButton *startButton = m_window->findChild<QPushButton *>
        ("pushButton_start");
17    QVERIFY2(startButton, "Start Button not found");
18    QTest::mouseMove(startButton, QPoint(50, 10));
19    QTest::mouseClick(startButton, Qt::LeftButton, Qt::NoModifier,
        QPoint(50, 10));

21    QTest::qWait(50);
22    QLabel *errorStatusLabel = m_window->findChild<QLabel *> ("
        label_errorStatus");
23    QVERIFY2(errorStatusLabel, "Connect Status Label not found");
24    QVERIFY2(errorStatusLabel->styleSheet().contains("background-
        color: rgb(203, 47, 47)"), "Color of label_errorStatus is
        not Red");

26    QPushButton *resetButton = m_window->findChild<QPushButton*>("
        pushButton_resetErrorAndEStop");
27    QVERIFY2(resetButton, "Reset Button not found");
28    QTest::mouseMove(resetButton, QPoint(50, 10));
29    QTest::mouseClick(resetButton, Qt::LeftButton, Qt::NoModifier,
        QPoint(50, 10));

31    QTest::qWait(50);
32    QVERIFY2(errorStatusLabel->styleSheet().contains("background-

```

```
        color: rgb(27, 193, 00)"), "Color of label_errorStatus is  
        not Green");  
  
34     textField->clear();  
35 }
```

Auf die Methoden `QtGuiTest::testClearInput()`, `QtGuiTest::testDisconnect()` und `QtGuiTest::testRandomInput()` wird an diesem Punkt des Dokuments nicht näher eingegangen, da der Testaufbau, Syntax sowie der erreichbare Umfang des Qt Test Frameworks durch die gezeigten Methoden ersichtlich sein sollte.

4 Vergleich

4.1 Squish

4.2 Qt Test

4.3 Vergleich

5 Fazit

Literaturverzeichnis

- [Abo22] About Squish. <https://www.froglogic.com/squish/>, 2022. [Zugegriffen am 13.06.2022].
- [Cap] S. Capgemini. Microfocus, "World Quality Report 2021". URL: <https://www.capgemini.com/research/world-quality-report-wqr-2021-22>.
- [Doc22a] Q. Documentation. About Qt. https://wiki.qt.io/About_Qt#History, 2022. [Zugegriffen am 10.06.2022].
- [Doc22b] Q. Documentation. Qt Test. <https://doc.qt.io/qt-6/qttest-index.html>, 2022. [Zugegriffen am 10.06.2022].
- [Qt 22] Questions tagged [qt]. <https://stackoverflow.com/questions/tagged/qt>, 2022. [Zugegriffen am 10.06.2022].