



Fakultät für Informatik

Studiengang Informatik

Automatisierte UI Tests für Windows Qt-Anwendungen

Software Qualitätssicherung

von

Dominik Sieberer

Stand der Version: 17.06.2022

Abstract

Das Automatisierte Testen von UIs während der Entwicklung ist Industriestandard und bietet viele Vorteile wie beispielsweise Wiederverwendbarkeit Skalierbarkeit und Kosteneffizienz. Um dieses bisher ungenutzte Potenzial vollständig auszuschöpfen soll ein Framework gefunden werden welches ermöglicht Automatisierte UI Tests für Windows Qt-Anwendungen durchzuführen. Die Anforderungen, die an das Framework gestellt werden, sind in Tabelle 1.1 aufgeführt. Die drei möglichen Kandidaten, die für die Tests infrage kommen sind Qt Test, Squish und OpenHMI Tester. Qt Test ist Teil von Qt. Die Test-Cases werden in C++ geschrieben und bei jedem Aufruf der Software im Debug Modus ausgeführt. Qt Test hat die gestellten Anforderungen am besten erfüllt. Squish ist ein von Froglogic angebotenes Framework. Die Tests können in verschiedenen Sprachen, wie beispielsweise JavaScript oder Python geschrieben werden, oder über die Squish IDE aufgenommen und generiert werden. In OpenHMI Tester werden die Test-Cases durch das UI aufgenommen. Da OpenHMI Tester offiziell nicht auf der aktuellen Version von Qt läuft und auch auf der von Entwickler angegebenen Version nicht in einen lauffähigen Stand gebracht werden konnte, wurde es nicht weiter evaluiert.

Insgesamt erfüllten sowohl Squish als auch Qt Test die gestellten Anforderungen gut. Da Qt Test kostengünstiger, leichter aufzusetzen ist und die Test-Cases eine kürzere Laufzeit haben, fällt die Entscheidung auf Qt Test.

Inhaltsverzeichnis

1	Automatisierte UI Tests	1
1.1	Vorteile von automatisierten UI Tests	1
1.2	Anwendung	1
1.3	Anforderungen	3
1.4	Wahl der Frameworks	4
2	Squish	5
2.1	Einführung zu Squish	5
2.2	Durchführung der Tests	6
2.3	Tests	7
3	Qt Test	9
3.1	Einführung Qt Test	9
3.2	Durchführung der Tests	11
3.3	Tests	13
4	Vergleich	18
4.1	Anforderungserfüllung von Squish	18
4.2	Anforderungserfüllung von Qt Test	18
4.3	Vergleich	19
5	Fazit	20
	Literaturverzeichnis	21

Abbildungsverzeichnis

1.1 Anwendung	2
2.1 Squish IDE	6
2.2 Squish Control Bar	6
2.3 Squish Test Case UI	7

1 Automatisierte UI Tests

Das erste Kapitel beginnt mit einer Erklärung, warum automatisierte UI Tests durchgeführt werden sollten und welche Vorteile sie bieten. Danach wird die Anwendung, anhand der die beiden ausgewählten Frameworks verglichen werden, vorgestellt. Kapitel 1.3 stellt die gestellten Anforderungen in Tabellenform dar und im letzten Kapitel wird beschrieben, wie es zur Wahl der beiden Frameworks kam.

1.1 Vorteile von automatisierten UI Tests

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 Softwareentwicklung [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.

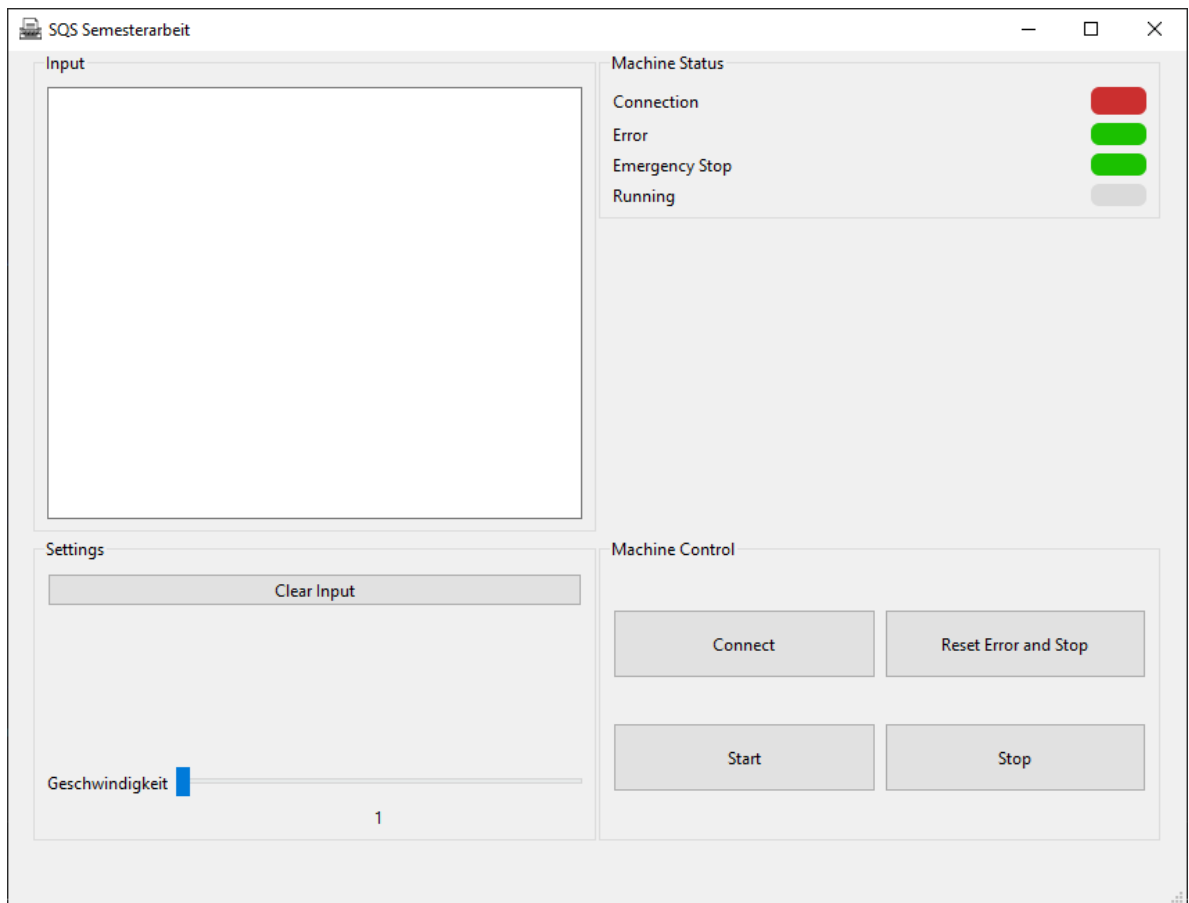


Abbildung 1.1 Anwendung an der die Tests evaluiert werden

Die Anwendung (Abbildung 1.1) lässt sich in 4 Bereiche unterteilen. Der erste Bereich, links oben, ist ein Textfeld. Dieses Textfeld kann mit User Input gefüllt werden. Der Input wird durch Drücken des Startbuttons 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. Sollte 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	Ist der Source-Code öffentlich verfügbar
04	Integration in vorhandene Pipeline	Wie viel Zeit benötigt es die Tests in die Pipeline zu integrieren
05	In Qt Editor ausführbar	Sind die Tests im Qt Editor ausführbar
06	Aufwand	Benötigte Zeit zum Erstellen eines Tests
07	Lesbarkeit	Kann der Testcode von Entwicklern gelesen werden
08	Änderbarkeit	Benötigte Zeit zum Ändern eines Tests
09	Support / Community	Aktivität und Größe der Community (Stack Overflow, eigenes Forum, ...)
10	Reife / Robustheit	Stabilität des Frameworks
11	Updates	Update-Zyklus des Frameworks
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 (1.1) 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.

Anforderung [01] legt fest, dass die Kosten, die für das UI Testing Framework anfallen sollen, so gering wie möglich sein. Bestenfalls sollten keine Kosten anfallen. Dies ist direkt mit der Anforderung [02] verknüpft. Eine GNU General Public Lizenz oder vergleichbares wäre daher vorteilhaft. [03] Wenn der Source Code des UI Testing Frameworks frei verfügbar ist, bestünde die Möglichkeit, Änderungen oder Ergänzungen am Framework vorzunehmen. Die Priorität von [03] ist als eher gering einzustufen. [04] Wenn das Framework nicht in die verwendete Pipeline integriert werden kann, kann es nicht automatisiert verwendet werden und würde damit direkt ausgeschlossen werden. [05] Der beste Fall hierfür wäre die Möglichkeit die Tests im Qt Editor ausführen zu können. [06] Ebenso wichtig ist, wie viel Aufwand das Schreiben oder Aufnehmen eines Tests ist, sowie die benötigte Zeit, um mit dem Framework familiär zu werden. [07] Dabei ist es ebenso vorteilhaft, wenn man den Testcode lesen, nachvollziehen und verstehen kann. [08] Ebenso wird verglichen, wie viel Aufwand eine Änderung eines Tests ist, wenn sich beispielsweise die Position eines Buttons verändert.[09] Zu untersuchen ist auch, wie groß die Community der Frameworks auf Seiten wie Stack Overflow, Github oder einem eigenen Forum ist. Mit Anforderung [10] wird sichergestellt, dass Framework keine bekannten kritischen Bugs hat und bei ordnungsgemäßer Verwendung immer dasselbe Ergebnis liefert. Dazu prüft [11], ob die Aussicht auf

zukünftige Updates besteht. Ein weiteres Kriterium ist der Umfang des Frameworks. Mit [12] wird gemessen, wie weit sich alle durch Qt bereit gestellten Widgets testen lassen und ob es die Möglichkeit gibt custom Widgets zu testen. Für den konkreten Anwendungsfall sind nur Nutzerinteraktionen mit Tastatur und Maus zu erwarten. [13] Die Laufzeit der Tests ist ebenso ein bedeutendes Kriterium. Mit [14] wird der Umfang und die Qualität der Dokumentation der Frameworks verglichen. Zuletzt wird mit [15] getestet, ob sich die Ergebnisse der Tests automatisiert auswerten lassen.

1.4 Wahl der Frameworks

Durch Recherche im Internet wurden 3 Frameworks gefunden, mit denen es möglich ist Qt-Windows Anwendungen zu testen. Als erstes Ergebnis kam das Qt Unit-Test Framework Qt Test, mit dem auch UI Tests möglich sind. Danach wurden sowohl Squish als auch OpenHMI Tester gefunden. Beim Aufbau der Tests für OpenHMI Tester wurde festgestellt, dass OpenHMI Tester nicht mit der neuesten Version von Qt kompatibel ist. Ein Downgrade der Qt Version der Anwendung verlief problemlos. Allerdings konnte nach über 2,5 Tagen OpenHMI Tester nicht zum Laufen gebracht werden. Da das Aufsetzen der anderen beiden Frameworks größtenteils reibungslos verlief, wurde die Entscheidung getroffen OpenHMI Tester nicht mehr weiter zu betrachten.

2 Squish

In diesem Kapitel gibt es eine kurze Einführung zum UI Testing Tool Squish, wie man mit Squish Tests durchführt und die ein Teil der UI Tests für die Anwendung aussehen.

2.1 Einführung zu 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.

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.

2 Squish

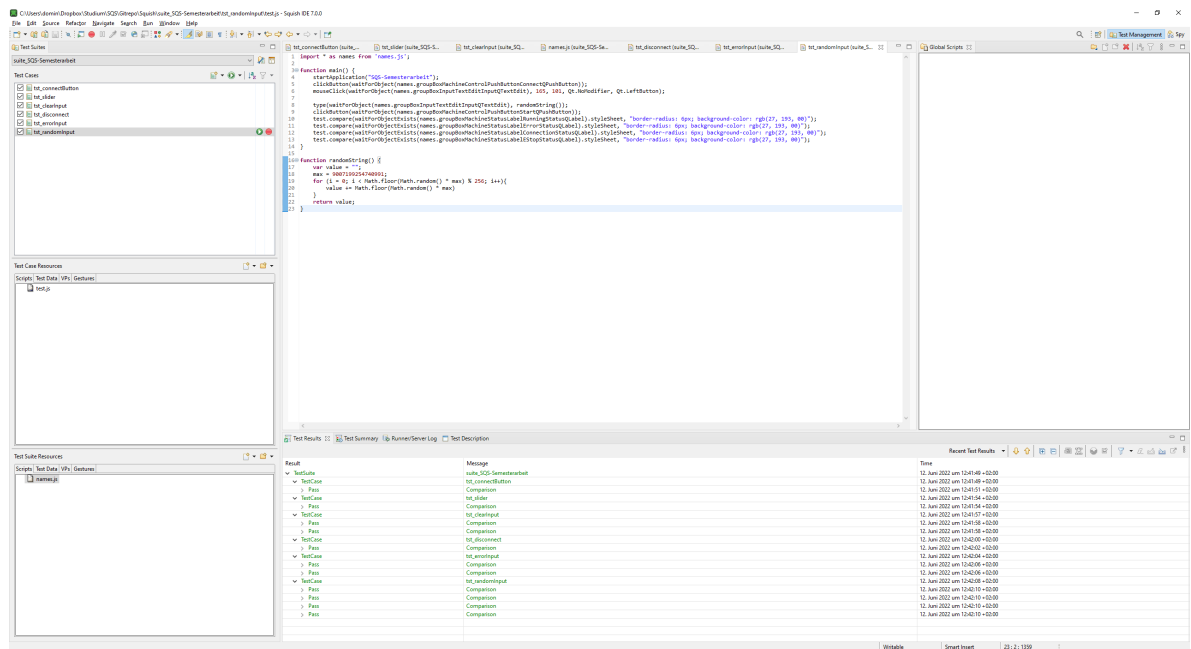


Abbildung 2.1 Squish IDE, in der die Tests erstellt werden.

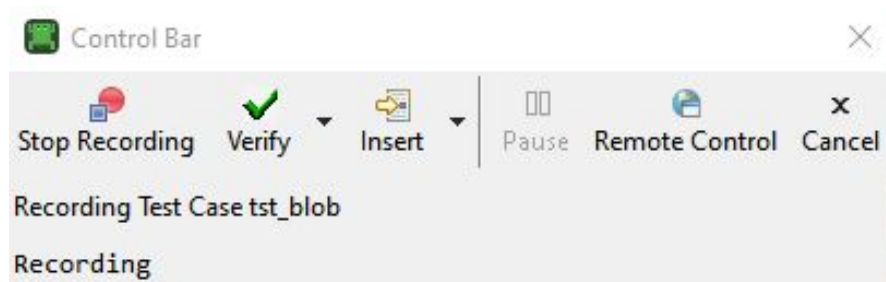


Abbildung 2.2 Control Bar, mit der die Aufnahme eines Tests gesteuert wird

2.2 Durchführung der Tests

Um einen einzelnen Test oder die gesamte Testsuite zu starten, gibt es verschiedene Möglichkeiten. Über die Squish IDE kann ein einzelner Test durch den Grünen Start Button neben dem Test gestartet werden (Abbildung 2.3). Die gesamte Test-Suite wird über Run -> Run Test Suite ausgeführt. Um die Tests automatisiert von der Command Line ausführen zu lassen, benötigt es eine laufende Instanz eines squishservers. Squishserver gib es sowohl für Windows als auch Unix Systeme. Mit dem Command *squishrunner -host <IP_AddressOfSquishServer> -port 4322 -testsuite <PathToTestSuite>* wird die in dem Pfad hinterlegte Test Suite ausgeführt. Der squishserver wird über das Command *squishrunner -host %SQUISH_SERVER_HOST% -port %SQUISH_SERVER_PORT% -testsuite %TESTSUITE% -reportgen xml2.1,%REPORTPATH%/report.xml* gestartet [Squ22a]. Die dadurch generierte report Datei kann automatisiert ausgewertet werden.

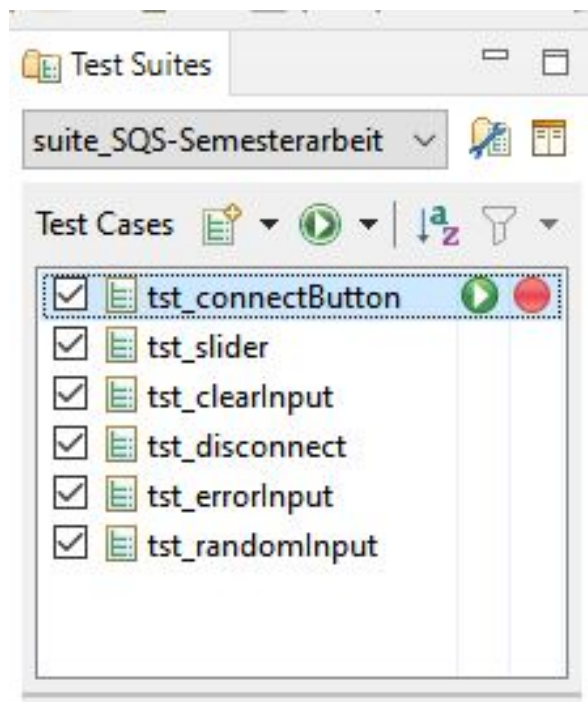


Abbildung 2.3 Squish IDE, Ausschnitt der Test Suite und Test Cases

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(
4         names.groupBoxSettingsHorizontalSliderSpeedQSlider), 6);
5     test.compare(waitForObjectExists(
6         names.groupBoxSettingsLabelSpeedValueQLabel).text, "6");
7 }

```

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(
4         names.groupBoxMachineControlPushButtonConnectQPushButton));
5     mouseClicked(waitForObject(
6         names.groupBoxInputTextEditInputQTextEdit), 234, 156,
7         Qt.NoModifier, Qt.LeftButton);
8     type(waitForObject(names.groupBoxInputTextEditInputQTextEdit),
9         "error");
10    clickButton(waitForObject(
11        names.groupBoxMachineControlPushButtonStartQPushButton));
12    test.compare(waitForObjectExists(
13        names.groupBoxMachineStatusLabelErrorStatusLabel)
14        .styleSheet, "border-radius: 6px; background-color: rgb(
15        203, 47, 47)");
16    clickButton(waitForObject(
17        names.groupBoxMachineControlPushButtonResetErrorAndEStopQPushButton));
18    test.compare(waitForObjectExists(
19        names.groupBoxMachineStatusLabelErrorStatusLabel)
20        .styleSheet, "border-radius: 6px; background-color: rgb(27,
21        193, 00)");
22 }

```

3 Qt Test

Kapitel drei gibt einen kurzen Überblick zu Qt und Qt Test. Danach wird zur Schau gestellt, wie die Tests in der Entwicklungsumgebung aufgerufen werden und wie ein Teil der UI Tests aussieht.

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 Stack Overflow 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 `QWdiget` fokussiert und der als Parameter übergebene String als Tastatureingabe dem `QWdiget` übergeben. Mit dem Makro `QCOMPARE()` werden zwei Werte verglichen. Sobald die beiden übergebenen Werte denselben 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();
5
6     mainWindow->show();
7
8     while (!m_window->isVisible()) {
9         QTest::qWait(200);
10    }
11
12    QTest::qWait(500);
13
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, dass 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
7     QTest::mouseMove(button, QPoint(20, 10));
8
9     QTest::mouseClick(button, Qt::LeftButton, Qt::NoModifier,
10         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. Mit 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 Startbuttons 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

Im Kapitel Vergleich wird zuerst geprüft welche der im ersten Kapitel gestellten Anforderungen von Squish und welche von Qt Test erfüllt werden. im Anschluss werden die beiden Frameworks miteinander verglichen.

4.1 Anforderungserfüllung von Squish

Der genaue Preis von Squish konnte nicht ermittelt werden. Laut der Webseite werden die Squish Subscriptions als floating licenses vertrieben, die alles von kleinen individuellen Gruppen bis hin zu großen Unternehmen abdeckt [Squ22b]. Auf Nachfrage bei Froglogic per Mail ist der Preis abhängig von Branche, Unternehmensgröße sowie der Anzahl von laufenden Testinstanzen. Der genaue Preis lässt sich deshalb nicht ermitteln und Anforderung [01] ist daher nicht quantifizierbar und nicht exakt beantwortbar. [02] Wie eben genannt wird Squish über eine gesonderte Lizenz vertrieben, die nicht mit GPL vergleichbar ist. [03] Squish ist keine Open-Source-Software. Um Squish in die vorhandene Pipeline zu integrieren, [04] wird etwa ein halber Tag bis Tag benötigt. [05] Squish ist nicht im Qt Editor ausführbar. Um einen einzelnen Test in Squish zu erstellen, [06] ist vergleichsweise wenig Zeit nötig. Dies gilt sowohl für das Aufnehmen der Tests über das UI als auch das direkte Schreiben der Tests. Die benötigte Zeit skaliert logischerweise mit der Komplexität des Tests. [07] Der Testcode ist für Entwickler leicht lesbar. [08] Durch den geringen Aufwand zum Erstellen eines Tests und der guten Lesbarkeit ist der Aufwand einen Test zu ändern [08] gering. Bei Änderungen im UI, die nicht die Identifier oder geprüfte Attribute der UI Elemente betreffen, beispielsweise die Position, sind keine Änderungen der Test Cases nötig. [09] Es gibt keinerlei Foren und kaum Fragen auf Stack Overflow zu Squish. Mit der Lizenz ist technischer Support mit inbegriffen. Die Frage nach der Lizenz war am nächsten Werktag beantwortet. Da Squish seit 14 Jahren auf dem Markt ist, wird von einer hohen Robustheit und Reife [10] ausgegangen. Es sind keinerlei technische Probleme beim Testen aufgetreten. [11] In der gekauften Lizenz von Squish sind Updates inklusive. Wie häufig Updates von Squish ausgerollt werden und wie umfangreich diese sind, konnte nicht quantifiziert werden. [12] Es konnten keine Qt Widgets gefunden werden, die nicht mit Squish testbar sind. [13] Die Ausführung der 6 Tests dauerte 18 Sekunden. Die Dokumentation [14] von Squish ist sehr umfangreich. Nach einer kurzen Einfindungsphase war die Dokumentation gut verständlich und gesuchte Informationen konnten in angemessener Zeit gefunden werden. [15] Die Testergebnisse sind automatisiert auswertbar.

4.2 Anforderungserfüllung von Qt Test

[01] Qt und damit Qt Test ist, solange keine kommerzielle Lizenz nicht benötigt wird, kostenlos. [02] Qt ist unter GPL und LGPLv3 verfügbar. [03] Die LTS Version von Qt ist nicht Open Source. [04] Qt Test ist mit minimalem Aufwand in die Pipeline integrierbar und im Qt Editor ausführbar [05]. [06] Der Aufwand um einen Test zu erstellen ist angemessen. [07] Da

es sich um C++ Code handelt, sind die Tests für Entwickler ohne Mehraufwand lesbar. [08] Da es sich um C++ Code handelt, ist der Aufwand um Änderungen einzubringen hauptsächlich von der Code-Qualität abhängig. Bei Änderungen im UI, die nicht die Identifier oder geprüfte Attribute der UI Elemente betreffen, beispielsweise die Position, sind keine Änderungen der Test Cases nötig. [09] Wie in Kapitel 3.1 erwähnt, handelt es sich um eine mittelgroße, aktive Community. Zusätzlicher Support von Qt könnte über eine kommerzielle Lizenz eingekauft werden. [10] Da Qt seit 2003 entwickelt wird und von vielen Unternehmen verwendet, weshalb Qt Test als ausgereift und robust und ausgereift eingestuft wird. Während des Tests gab es keine Probleme. [11] Es gibt regelmäßig Updates sowie neue Versionen von Qt. Qt Test deckt alle Qt Widgets [12] ab. Die 6 Testcases für die Anwendung benötigen eine Zeit von 2150ms. Die Laufzeit [13] ist damit sehr gering. [14] Die Dokumentation von Qt und Qt Test ist qualitativ gut und sehr umfangreich. [15] Die Tests sind automatisiert auswertbar.

4.3 Vergleich

Für Anforderung [01] und [02] schneidet Qt Test eindeutig besser ab als Squish. Anforderung [03] wird von keinen von beiden erfüllt. Da Qt Test leichter in die vorhandene Pipeline integrierbar ist, als Squish ist Qt Test für Anforderung [04] zu bevorzugen. Allerdings ist die benötigte Zeit um Squish in die Pipeline zu integrieren tragbar.[05] Qt Test ist im Qt Editor ausführbar, Squish nicht. Der zeitliche Aufwand um einen Test zu erstellen [06] ist bei Squish geringer als bei Qt Test. Anforderung [07] bis [12] werden von beiden Kandidaten gleichermaßen erfüllt. Die Laufzeit von Qt Test ist mit 2,2 Sekunden bedeutend geringer als die 18 Sekunden von Qt Test. Die Anforderungen [14] sowie [15] sind sowohl mit Squish als auch mit Qt Test gegeben.

Im direkten Vergleich erfüllt Qt Test die gestellten Anforderungen besser als Squish. Daher wird die Entscheidung getroffen, automatisierte UI Tests mit dem Qt Test Framework durchzuführen. Diese Entscheidung basiert vor allem an den niedrigeren Kosten von Qt Test sowie der geringeren Laufzeit gegenüber Squish. Allerdings ist zu erwähnen, dass Squish ebenso ein akzeptables Framework zum automatisierten Testen des UIs ist.

5 Fazit

Automatisiertes Testen von UIs während der Entwicklung ist Industriestandard. Für unsere Anwendung kamen die drei Frameworks OpenHMI Tester, Squish und Qt Test infrage. Nachdem OpenHMI Tester nicht in einen lauffähigen Stand gebracht werden konnte, wurde OpenHMI Tester nicht weiter verfolgt. Deshalb wurden nur für Squish und Qt Tests Test Cases geschrieben und geprüft, inwiefern die Frameworks die in Kapitel 1.3 gestellten Anforderungen erfüllen. Sowohl Squish als auch Qt Test haben die gestellten Anforderungen akzeptabel erfüllt. Qt Test kostet im Gegensatz zu Squish kein Geld, war schneller aufzusetzen ist und die Test Cases haben eine geringere Laufzeit. Aus diesen Gründen wurde die Entscheidung getroffen, Qt Test zu verwenden.

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].
- [Squ22a] Squish CLI. <https://www.froglogic.com/blog/how-to-run-squish-from-the-command-line-interface/>, 2022. [Zugegriffen am 14.06.2022].
- [Squ22b] Squish Pricing. <https://www.froglogic.com/squish/prices/>, 2022. [Zugegriffen am 16.06.2022].