Projektarbeit 1

Evaluation Swing-kompatibler Capture-and-Replay-Tools

DANIEL KRAUS
Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät für Informatik und Wirtschaftsinformatik

Matrikelnummer: 54536
Referent: Prof. Dr.-Ing. Holger Vogelsang

Zusammenfassung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Stichwörter: Java, GUI, Swing, Capture and Replay

1 Einleitung

Die Entwicklung von Software ist sehr dynamisch, d. h. die zugrunde liegende Codebasis wird kontinuierlich verändert. Diese Änderungen dürfen die Korrektheit der Anwendung in keinster Weise negativ beeinflussen. Konsequentes Testen stellt daher eine unerlässliche Tätigkeit dar, die maßgeblich über den Erfolg oder Misserfolg von Software entscheidet. Effektive Tests sowie eine hohe Testabdeckung verringern das Risiko von Fehlern und unerwünschten Seiteneffekten, die bei Anwendern rasch auf

Frustration stoßen oder gar zum Boykott einer Anwendung führen können. Bei modernen Entwicklungsmethoden wie etwa Test-Driven Development (TDD), werden die Tests noch vor der Implementierung der zu testenden Komponenten angefertigt. Martin[1] zufolge sind Tests sogar wichtiger als die eigentliche Anwendung, da sie vielmehr eine exakte Spezifikation dieser darstellen. [Mar13]

Doch was sind gute Tests? Neben implementierungstechnischen Details stellt sich auch die Frage, welche Bausteine in welchem Umfang adressiert werden sollen. Eine mögliche Antwort hierauf liefert die von Cohn[2] eingeführte *Test-Pyramide* (siehe Abbildung 1). Demnach sind automatisierte und isolierte Low-Level-Tests zu bevorzugen: „This way tests can run fast, be independent (from each other and shared components) and thereby stable“ [and13, S. 42]. Hierfür stehen im Java-Umfeld zahlreiche Frameworks zur Verfügung, wie beispielsweise JUnit[3] (Automatisierung) und Mockito[4] (Isolation). Nichtsdestotrotz müssen auch die oberen Schichten verifiziert werden, besonders kritisch ist dabei das Testen des *Graphical User Interface (GUI)*. Die GUI stellt oftmals die einzigste Schnittstelle zwischen dem Anwender und der Software dar, wodurch ihre korrekte Funktionsweise die subjektive Qualität stark beeinflusst. Das Testen der GUI gilt jedoch als sehr mühsam, Fowler bezeichnet dies im Allgemeinen als „brittle, expensive to write, and time consuming to run“ [Fow12]. Zum einen führen bereits kleine Eingabemasken zu zahlreichen Kombinationsmöglichkeiten, was das Erstellen adäquater Tests erschwert, zum anderen entscheidet in der Regel die Sequenz dieser Eingaben darüber, welche Funktionen in der GUI überhaupt getestet werden können.

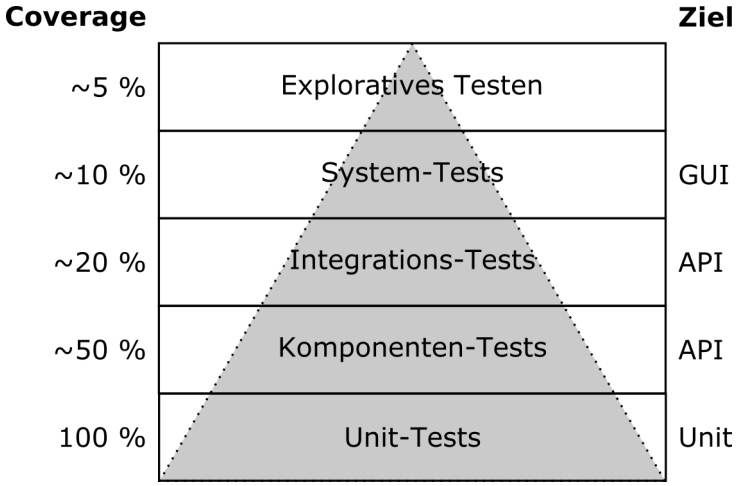

Abbildung 1: Test-Pyramide nach Cohn [and13, S. 60]

[1] Robert C. Martin, auch bekannt als „Uncle Bob“, ist ein US-amerikanischer Softwareentwickler und Initiator des agilen Manifests – dem Fundament der heutigen agilen Vorgehensmodelle. [Hig01]

[2] Mike Cohn ist ebenfalls US-amerikanischer Softwareentwickler und gilt, u. a. mit Jeff Sutherland und Ken Schwaber, als einer der Erfinder des agilen Vorgehensmodells Scrum. [Den12]

[3] `http://junit.org`.

[4] `http://mockito.org`.

Insbesondere Regressionstests führen hierbei zu Problemen. Beim Regressionstesten werden Testfälle fortlaufend wiederholt, sodass Änderungen keine neuen Fehler – die Regressionen – in bereits getesteten Komponenten hervorrufen bzw. um diese frühzeitig zu entdecken. Ändert sich die GUI, kann sich jedoch der Ausführungspfad (ergo die Sequenz der Eingaben) eines solchen Tests ebenso ändern, wenn z. B. die Position einer GUI-Komponente modifiziert wurde. Tests müssen daher immer wieder erstellt bzw. angepasst, durchgeführt und ausgewertet werden.

Ein weit verbreiteter Ansatz um dies zu automatisieren ist *Capture and Replay*. Die Tests werden hierfür einmalig manuell durchgeführt und dabei aufgezeichnet (Capture), anschließend können diese beliebig oft und vollautomatisch wiederholt werden (Replay). Für gewöhnlich „lauscht“ das Tool zu diesem Zweck auf verschiedene GUI-*Events*, wie etwa Mausklicks oder Tastatureingaben, und erstellt auf dieser Basis ein *Testskript*. Dies hat den Vorteil, dass Tests rasch erstellt werden können und keine Programmierkenntnisse voraussetzen, wodurch sich solche Werkzeuge besonders für Domänenexperten eignen. Dennoch entstehen auch hier Probleme bei der Durchführung der Regressionstest:

> [...] these test cases often cause difficulty during software maintenance and regression testing, because relatively minor changes to the GUI can cause a test case to break, or, cease to be executable against an updated version of the software. When such a situation occurs, a large manual effort is often required to repair some subset of the cases in a test suite, or worse yet; [sic] [MM09, S. 1]

In dieser Arbeit werden fünf kommerzielle sowie quelloffene Capture-and-Replay-Tools für die Oberflächentechnologie *Swing* evaluiert. Swing dient der Implementierung von Rich Clients[5] mit Java, zwar gilt die Technologie als veraltet, nichtsdestotrotz wird sie noch in zahlreichen Anwendungen eingesetzt und ist in puncto Popularität nahezu gleichauf mit dem Nachfolger JavaFX [Col15].

Zunächst wird in Abschnitt 2 der Capture-and-Replay-Ansatz im Detail erläutert, in Abschnitt 3 findet sich eine Einführung in Java Swing. Abschnitt 4 beinhaltet die eigentliche Evaluation und beginnt mit der Beschreibung des Testszenarios (4.1). Anschließend werden die getesteten Tools aufgeführt und kurz beschrieben (4.2), woraufhin die Präsentation der Evaluationsergebnisse erfolgt (4.3). Im letzten Teil der Arbeit, Abschnitt 5, werden diese Ergebnisse interpretiert und kritisch bewertet.

[5] Bezeichnet eine Anwendung, bei der Geschäftslogik sowie GUI clientseitig implementiert sind.

2 Capture and Replay

3 Java Swing

4 Evaluation

4.1 Testszenario

Wie bereits in Abschnitt 2 erwähnt, führen das GUI Element Identification Problem sowie das Temporal Synchronization Problem häufig zu Komplikationen bei der Anwendung von Capture and Replay und nicht zuletzt zu einer kontroversen Reputation: „Record-playback tools are almost always a bad idea for any kind of automation, since they resist changeability and obstruct useful abstractions“ [Fow12]. Die Fähigkeit eines Tools Testskripte auszuführen, die auf einem obsoleten Stand der Software basieren (d. h. bestmöglich beide Probleme lösen), ist somit ein entscheidendes Qualitätskriterium. Der Fokus dieser Arbeit liegt daher auf der Komponentenerkennung und der daraus resultierenden Robustheit eines Tools. Zu diesem Zweck wird das Testszenario in folgende vier Schwierigkeitsstufen unterteilt:

Stufe 0 Initial wird überprüft, welche Swing-Komponenten grundsätzlich aufgezeichnet und abgespielt werden können. Hierzu werden verschiedene ausgewählte Komponenten und Funktionen klassifiziert sowie getestet:

- Data: `JTable`, `JTree`, `JList`.
- Controls: `JButton`, `JCheckBox`, `JRadioButton`, `JComboBox`, `JSlider`, `JSpinner`.
- Text: `JFormattedTextField`, `JHistoryTextField`, `JPasswordField`, `JEditorPane`.
- Choosers: `JFileChooser`, `JColorChooser`, `JOptionPane`.
- Misc: Drittanbieter-Komponenten, Drag and Drop, asynchrone Prozesse.

Stufe 1 Die erste Stufe befasst sich mit den *externen* Eigenschaften von Komponenten, d. h. Eigenschaften, die das GUI-Element zwar beeinflussen, jedoch von außen gesteuert werden können (z. B. das Look and Feel oder das Layout).

Stufe 2 In der zweiten Stufe der Evaluation werden die *internen* Eigenschaften von Komponenten betrachtet. Ergo Merkmale, die grundsätzlich über die Komponente selbst verändert werden, wie etwa die Größe, die Farbe oder ein Label.

Stufe 3 Die dritte und letzte Stufe kombiniert die zuvor eingeführten Anpassungen. Beispiel: Innerhalb einer Änderung wird eine Komponente in einem zusätzlichen Layout-Container verschachtelt, umbenannt und dessen Größe modifiziert.

Diese unterschiedlichen Stufen erlauben bei Fehlschlägen nicht nur Rückschlüsse auf die Strategie bei der Wiedererkennung von Komponenten, sondern dienen auch der Simulation alltäglicher Entwicklungsprozesse und der damit verbundenen Probleme.

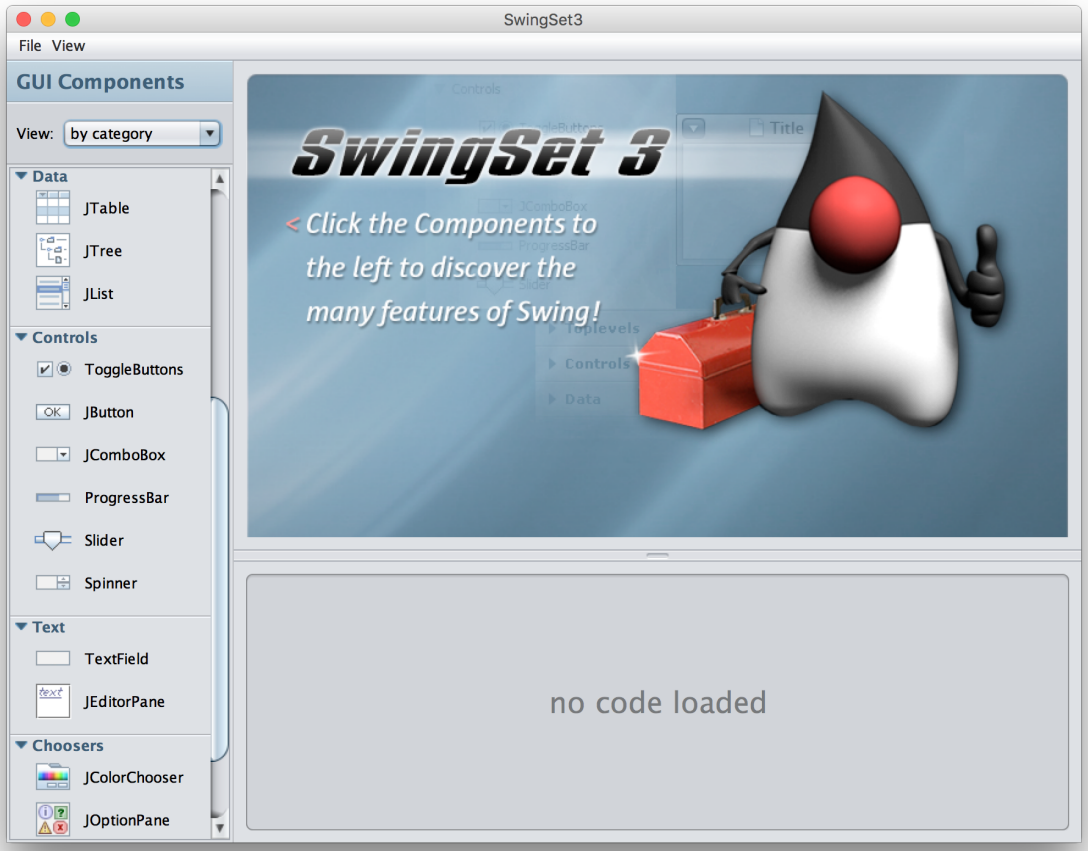

Abbildung 2: Screenshot der SwingSet3-Oberfläche

Als *System Under Test (SUT)* dient die von Oracle bereitgestellte *SwingSet3*-Demo[6]. Hierbei handelt es sich um eine offizielle Beispielanwendung, die einen Großteil der verfügbaren Swing-Funktionen und -Komponenten beinhaltet, ähnlich dem heutigen Ensemble[7] für JavaFX. Wie Abbildung 2 zeigt, finden sich dort bereits die Klassen mit den zugehörigen Komponenten aus Stufe 0. Für `JFileChooser` sowie Drag and Drop existieren separate Minianwendungen, als Drittanbieter-Komponente wird die Datumsauswahl `JDatePicker`[8] in SwingSet3 integriert. Zum Test asynchroner Prozesse eignet sich die vorhandene `JProgressBarDemo`, welche zusätzlich mit einer variierenden Ladezeit versehen wird.

Auf dieser Basis werden nun die Stufen 1 – 3 inkrementell hinzugefügt, um so die

[6] `https://java.net/projects/swingset3`.
[7] `http://oracle.com/technetwork/java/javase/overview/javafx-samples-2158687.html`.
[8] `http://jdatepicker.org`.

Komplexität der Tests schrittweise zu erhöhen. Um dies für jedes zu testende Tool möglichst einfach und wiederholbar durchführen zu können, wird das SwingSet3-Projekt zunächst nach Git[9] portiert und via GitHub[10] veröffentlicht. Dort kann auch die Umsetzung der einzelnen Schwierigkeitsstufen auf Quellcodeebene begutachtet werden.

Wenn Änderungen in einer lokalen Git-Repository eingecheckt werden sollen, so geschieht dies mithilfe von Commits. Jeder Commit erzeugt dabei eine neue *Revision*, die den Zustand der zugrunde liegenden Codebasis mittels SHA-1-Hashes versioniert. Das zugehörige Protokoll lässt sich mit dem `log`-Kommando ausgeben, wie Listing 1 exemplarisch zeigt. Die Parameter `--abbrev-commit` sowie `-3` sorgen dafür, dass lediglich ein eindeutiges Präfix des 20-Byte-Hashwerts angezeigt und die Ausgabe auf die letzten drei Revisions reduziert wird. Mittels `checkout` kann nun die lokale Repository auf einen beliebigen Zustand zurückversetzt werden. So führt etwa `git checkout 408a37f` dazu, dass im Beispiel die letzten beiden Änderungen (temporär) verworfen werden. Anschließend ist es möglich, den Endzustand Schritt für Schritt wiederherzustellen. Dadurch lassen sich die Änderungen am Quellcode bzw. die korrespondierenden Schwierigkeitsstufen mühelos und unter gleichen Bedingungen für jedes Tool realisieren.

```
$ git log --abbrev-commit -3
commit 406c93d
Author: Daniel Kraus <daniel.kraus@mailbox.org>
Date:   Mon Dec 14 23:40:20 2015 +0100

    Clear history button.

commit 2019c22
Author: Daniel Kraus <daniel.kraus@mailbox.org>
Date:   Thu Nov 12 12:38:33 2015 +0100

    History text field for login.

commit 408a37f
Author: Daniel Kraus <daniel.kraus@mailbox.org>
Date:   Tue Nov 10 15:13:26 2015 +0100

    Added login functionality.
```

Listing 1: Beispielausgabe von `git log`

[9] Im Folgenden werden Grundkenntnisse in verteilten Versionsverwaltungssystemen bzw. Git vorausgesetzt. Unter `https://git-scm.com` findet sich eine entsprechende Einführung sowie Dokumentation zur Vertiefung.

[10] `https://github.com/beatngu13/swingset3`.

Die Durchführung der Evaluation erfolgt auf einem Apple MacBook Pro (Retina 13″, Anfang 2015) mit 2,7 GHz Intel Core i5 und 8 GB Arbeitsspeicher sowie OS X El Capitan (10.11.2). Zur Ausführung und Anpassung von SwingSet3 bzw. Java-basierten Tools wird das Java Development Kit (JDK) in Version 6u65 verwendet.

4.2 Tools

4.2.1 Pounder

`http://sourceforge.net/projects/pounder`

4.2.2 Marathon

`http://marathontesting.com`

4.2.3 QF-Test

`https://qfs.de`

4.2.4 Tosca Testsuite

`http://tricentis.com`

4.2.5 Ranorex

`http://ranorex.de`

4.3 Ergebnisse

5 Fazit und Ausblick

Hier ein Beispiel für ein Code-Listing:

```
1  // Single-line comment.
2  private double notUsed = 47.11;
3
4  /*
5   * Multi-line comment.
6   */
7  public String foo(int bar) {
8          int baz = 42 + bar;
9
10         return "Look: " + baz;
11 }
```

Folgend noch eine Liste möglicher weiterer Literatur:

- [Ada+11]
- [Fow13]
- [NB13]
- [Ngu+14]
- [Rod+15]
- [Ste+00]

Literaturverzeichnis

[Ada+11] Andrea Adamoli et al. „Automated GUI Performance Testing“. In: *Software Quality Journal* 19.4 (Dez. 2011), S. 801–839.

[and13] andrena objects AG. *Agile Software Engineer. Student Edition*. Karlsruhe: Selbstverlag, 2013.

[Col15] Stephen Colbert. *Should Oracle Spring Clean JavaFX?* Nov. 2015. URL: `https://codenameone.com/blog/should-oracle-spring-clean-javafx.html` (besucht am 11. Dez. 2015).

[Den12] Steve Denning. *The Power of Scrum*. März 2012. URL: `http://forbes.com/sites/stevedenning/2012/03/01/the-power-of-scrum` (besucht am 4. Dez. 2015).

[Fow12] Martin Fowler. *TestPyramid*. Mai 2012. URL: `http://martinfowler.com/bliki/TestPyramid.html` (besucht am 16. Nov. 2015).

[Fow13] Martin Fowler. *PageObject*. Sep. 2013. URL: `http://martinfowler.com/bliki/PageObject.html` (besucht am 12. Nov. 2015).

[Hig01] Jim Highsmith. *History: The Agile Manifesto*. Feb. 2001. URL: `http://agilemanifesto.org/history.html` (besucht am 8. Dez. 2015).

[Mar13] Robert C. Martin. *Test First*. Sep. 2013. URL: `https://blog.8thlight.com/uncle-bob/2013/09/23/Test-first.html` (besucht am 8. Dez. 2015).

[MM09] Scott McMaster und Atif M. Memon. „An Extensible Heuristic-Based Framework for GUI Test Case Maintenance“. In: *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Edmonton, Alberta (Canada), 2009.

[NB13] Stanislava Nedyalkova und Jorge Bernardino. „Open Source Capture and Replay Tools Comparison“. In: *Proceedings of the Sixth International C* Conference on Computer Science and Software Engineering*. Porto (Portugal), 2013.

[Ngu+14] Bao N. Nguyen et al. „GUITAR: An Innovative Tool for Automated Testing of GUI-driven Software“. In: *Automated Software Engineering* 21.1 (März 2014), S. 65–105.

[Rod+15] Elder M. Rodrigues et al. „An Empirical Comparison of Model-based and Capture and Replay Approaches for Performance Testing“. In: *Empirical Software Engineering* 20.6 (Dez. 2015), S. 1831–1860.

[Ste+00] John Steven et al. „jRapture: A Capture/Replay Tool for Observation-Based Testing“. In: *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis*. Portland, Oregon (USA), 2000.