HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Eine Einführung in die Regressionstests

Marc Herschel

Ausarbeitung für Projekt- und Qualitätsmanagement im Master-Studiengang „Angewandte Informatik“

20. Dezember 2020

Autor: Marc Herschel
1633667
marc@herschel.io

Prüfende: Denis Beßen und Dr. Gerald Delvos
Abteilung Informatik, Fakultät IV
Hochschule Hannover

Selbständigkeitserklärung

Mit der Abgabe der Ausarbeitung erkläre ich, dass ich die eingereichte Ausarbeitung zum Projekt- und Qualitätsmanagement selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 20. Dezember 2020 Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Softwareentwicklung ist ein langwieriger und komplexer Prozess, in dem die Qualität des zu entwickelnden Produktes eine entscheidende Rolle spielt. Dadurch, dass in dem Prozess der Softwareentwicklung verschiedene Stakeholder involviert sind und es sich um einen iterativen Prozess handelt, müssen Maßnahmen zur Qualitätssicherung getroffen werden, um ein den Anforderungen des Endkunden entsprechendes Produkt abzuliefern. Eine dieser qualitätssichernden Maßnahmen sind Softwaretests, die genutzt werden um die Validierung der Software bezogen auf ihren Einsatzzweck durchzuführen. In dieser Ausarbeitung zu der bereits erfolgten Präsentation steht nicht der Oberbegriff der Softwaretests im Vordergrund, stattdessen wird ein Einblick in die Untermenge der Regressionstests gegeben. Regressionstests unterscheiden sich in ihrer grundlegenden Funktionsweise von den weitverbreiteten Modul- und Integrationstests[1], sind allerdings dennoch ein Grundpfeiler der Softwarequalitätssicherung.

Zuerst wird in dieser Ausarbeitung auf die Definition und Grundlagen der Regressionstests eingegangen. Anschließend wird die Notwendigkeit derer in der heutigen Softwareentwicklung beleuchtet und auf die abstrakte Funktionsweise der Regressionstests eingegangen. Testautomatisierung und Testberichte werden ebenfalls behandelt und es werden verschiedene Techniken zur Durchführung von Regressionstests vorgestellt. Da der Autor dieser Ausarbeitung seine Bachelorarbeit über die Konzeption und Realisierung eines Regressionstestmechanismus für den Autobewerter *Graja*[2] der Hochschule Hannover geschrieben hat, beinhaltet diese Ausarbeitung ebenfalls eine Fallstudie bezüglich der Konzeption und Entwicklung dieses Regressionstestmechanismus, um einen beispielhaften Einblick in die praktische Anwendung des theoretischen Teils zu geben. Abgeschlossen wird diese Ausarbeitung durch eine anschließende Bewertung der Regressionstests – sowohl im Allgemeinen, als auch auf die Fallstudie bezogen. Schlussendlich wird auch noch einmal der Bezug zwischen Regressionstests und dem Projekt- und Qualitätsmanagement der Softwareentwicklung hergestellt und erläutert, warum Regressionstests ein wichtiger Teil dessen sind.

2 Grundlagen

Dieses Kapitel stellt den theoretischen Teil dieser Ausarbeitung dar und fokussiert sich darauf, die grundlegenden Eigenschaften und elementaren Funktionsweisen der Regressionstests zu vermitteln.

2.1 Definition: Regressionstest

Um zu verstehen wie und warum sich Regressionstests von Modul- und Integrationstests abgrenzen und ein elementares Grundverständnis zu erlangen werden im folgenden die Definitionen der *IEEE*[3] und der *FDA*[4] miteinbezogen.

So beschreibt die *IEEE* den Begriff des Regressionstests im *IEEE Standard Glossary of Software Engineering Terminology*[5] als:

[1] [Cha10] [7.1, 7.2]
[2] `http://graja.hs-hannover.de/`
[3] `https://www.ieee.org/`
[4] `https://www.fda.gov/`
[5] [IEE90] [S. 61]

> *„Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.“*

Während die *FDA* den Regressionstest im Leitfaden *General Principles of Software Validation - Guidance for Industry and FDA Staff*[6] wiefolgt beschreibt:

> *„Regression testing is the rerunning of test cases that a program has previously executed correctly and comparing the current result to the previous result in order to detect unintended effects of a software change.“*

Durch die beiden oben genannten Definitionen lässt sich ableiten, dass Regressionstests einen Vorgang des ständigen Neutestens von entweder dem ganzen oder Teilen eines Softwaresystems nach Änderungen darstellen. Regressionstests werden somit durchgeführt, um unerwartete Nebeneffekte, die möglicherweise durch Änderungen am Softwaresystem verursacht wurden, zu erkennen und zu vermeiden. Somit wird nach Modifikationen außerdem überprüft, ob das geänderte Softwaresystem bei einer Eingabe die gleiche Ausgabe wie bei der letzten stabilen Version vor der Änderung reproduziert und somit noch immer den Anforderungen der ursprünglichen Spezifikation (falls nicht abgeändert) entspricht.

Bei einem Regressionstest ist der Testfall also durch das Verhalten des Softwaresystems unter einer bestimmten Eingabe gestellt und das Ziel eines Testfalls ist es, eine gleichbleibende Ausgabe zu produzieren, um unerwünschte Änderungen durch Abweichungen zu detektieren. Differenzen zwischen den Ausgaben einer Eingabe der stabilen und modifizierten Version des Softwaresystems lösen daher ein Fehlschlagen eines Regressionstest-Testfalls und somit ein notwendiges Eingreifen der betreuenden Stakeholder aus, um die Qualität des Softwaresystems weiterhin zu gewährleisten.

2.2 Regressionen

Eine Regression stellt in der Softwareentwicklung einen Fehlertypen dar, der sich dadurch auszeichnet, dass Komponenten eines Softwaresystems nach Änderungen nicht mehr wie ursprünglich geplant funktionieren.[7] Solche Regressionen können z. B. durch das bloße Modifizieren einer Komponente auftreten, wie es z. B. im täglichen Entwicklungsalltag durch das Beheben von Fehlern oder dem Erweitern von Funktionalität geschehen kann.[8] Auch das falsche Anwenden von Refaktorisierungsmaßnahmen kann dazu führen, das sich eine Komponente innerhalb eines Softwaresystems nach einer wartenden Maßnahme nicht mehr identisch wie in der letzten stabilen Basisversion verhält.

Schon das einfache Aktualisieren einer externen Abhängigkeit wie z. B. einer *Library* oder einem *Framework* kann unteranderem zu einer Regression führen. Beispielsweise kann sich eine verwendete Funktion in der aktuelleren Version einer externen Abhängigkeit anders Verhalten als in der davor referenzierten Version, da z. B. beim Aufruf nun eine besondere Flagge übergeben werden muss, um das Verhalten der älteren Version zu reproduzieren. Regressionen können sich auch durch mehrere Module ziehen,

[6] [FDA02] [S. 25]
[7] [Cha10] [8.1]
[8] [Lig09] [S. 192]

falls diese in transitiver Abhängigkeit zu dem modifizierten Modul stehen. So können z. B. alle Modultests von **Modul A** erfolgreich durchlaufen, es aber später trotzdem zu Regressionen in **Modul C** kommen, da dieses von **Modul B** abhängt, das wiederum von **Modul A** abhängt und sich durch die Modifikationen in **Modul A** auch ohne direkte Änderungen an **Modul C** ein abweichendes Verhalten zur unmodifizierten Version auffindet.

Regressionsfehler werden zumeist unbewusst ausgelöst[9] und sind ohne automatisierte Softwaretests schwierig zu detektieren. Um solche Regressionen zu erkennen, wird das Softwaresystem Regressionstests unterzogen.

2.3 Gegebene Notwendigkeit

Die Verwendung von Regressionstests in einem Softwareentwicklungsprojekt erlaubt somit, eine Modifikation des projektbezogenen Softwaresystems durchzuführen und gleichzeitig gewissenhaft die korrekte Funktionalität des Systems nach Änderungen zu bewahren. Regressionstests sind deshalb eine Notwendigkeit für große, komplexe und von vielen Stakeholdern durchgeführten Softwareprojekte. Sie eignen sich außerdem auch für die Wartung von *Legacy Code*[10], für den möglicherweise keine Modul- und Integrationstests existieren und eine Erstellung dieser zu viel Aufwand bedeuten würde. Auch in Situationen, in denen möglicherweise keine Stakeholder mit Expertenwissen aus den Anfangsphasen eines Projekts involviert sind, können Regressionstests einem Team eine Sicherheit durch ein Auffangnetz bei der Wartung von *Legacy Code* geben.

Im Buch *Software Testing - Principles and Practices* wird dazu auch geschrieben[11]:

> *„Thus, regression testing can be defined as the software maintenance task performed on a modified program to instill confidence that changes are correct and have not adversely affected the unchanged portions of the program.“*

Demnach ist es ohne Regressionstests kaum möglich, effizient und selbstbewusst an einem komplexen Softwareprojekt zu arbeiten, da ansonsten das kaskadieren von Softwarefehlern nach Änderungen am Quellcode kaum verhindert werden kann. Die Wiederholung von Tests nach Modifikationen wird heutzutage in vielen Standards gefordert. Daher ist das Durchführen von Regressionstests eine notwendige Tätigkeit in der Softwareentwicklung, um Folgefehler und Seiteneffekte nach Modifikationen auszuschließen. Regressionstests stellen somit ein wichtiges Mittel zur Fehlerreduzierung dar.[12]

2.4 Grundlegende Eigenschaften

Regressionstests gehören zu den dynamischen Testtechniken, da hier das Verhalten einer aktuellen Software-Version gegen das Verhalten der Vorläuferversion abgeglichen wird.[13] Dieser Umstand bedeutet zwingend, dass die Software ausgeführt werden muss und ein Regressionstest immer zur Laufzeit stattfindet. Unter dieser Definition handelt

[9] [HK07] [S. 73]
[10] [Rö]
[11] [Cha10] [S. 256]
[12] [Lig09] [S. 194]
[13] [Lig09] [5]

es sich bei Regressionstests um *Black Box Testing*, da die interne Struktur der getesteten Software nicht berücksichtigt wird und lediglich Ausgaben einer gleichen Eingabe der modifizierten und unmodifizierten Softwareversionen miteinander verglichen werden.

Regressionstests sind des Weiteren keine fachlichen Tests. Die Zielsetzung liegt hier klar auf fachlicher Konsistenz (d. h. gleichbleibendes Verhalten nach Modifikation, sprich der Konsistenzüberprüfung). Sollte also ein fachlicher Fehler bereits in der Software vorhanden sein und eine Modifikation diesen beheben, wird ein Regressionstest negativ ausfallen, da dieser nur Veränderungen im Verhalten der Software erkennt, diese aber nicht eigenständig interpretieren kann.[14]

Aufgrund des hohen Wiederholungscharakters der Regressionstests sollte von einer manuellen Durchführung durch Tester abgesehen werden. Im besten Falle werden Regressionstest nach jeder Änderung am Softwaresystem durchgeführt, dies kann schnell zu einer hohen Testauslastung führen und ist beim manuellen Testen unwirtschaftlich und fehlerträchtig.[15][16] Eine Testautomatisierung mithilfe von Testwerkzeugen ist für die wirtschaftliche Durchführung von Regressionstests also unverzichtbar.

2.5 Abstrakte Funktionsweise

In diesem Unterkapitel wird der abstrakte Ablauf eines Regressionstests beschrieben, d. h. es wird nicht auf konkrete bereits bestehende Regressionstestwerkzeuge eingegangen, sondern auf den allgemeingültigen, auf alle Regressionstests anwendbaren Teil eingegangen.

Regressionstests sollten automatisch nach jedem *Build* des Softwaresystems ausgeführt werden. Je eher mögliche Regressionen entdeckt werden, desto kostengünstiger ist es, diese zu beheben. Somit können neueingeführte Regressionen schnellstens beseitigt werden, bevor diese auf andere Module des Softwaresystems überspringen.[17]

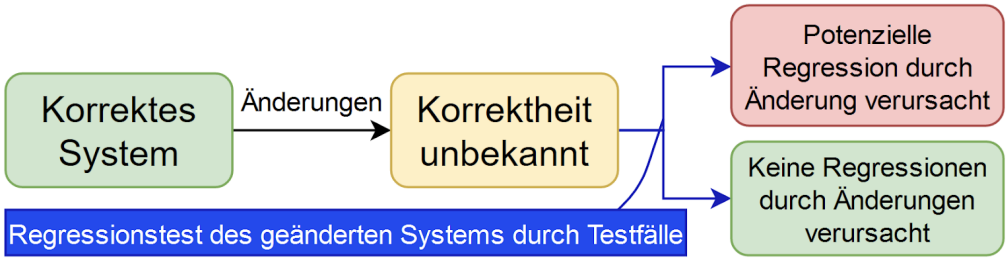


Abbildung 1: Abstrakte Funktionsweise eines Regressionstests

Es ergibt sich also ein wie in der Abbildung 1 dargestellter Ablauf. Änderungen an einem als funktional korrekt definierten Softwaresystem lösen eine Unsicherheit bei den Stakeholdern des Softwareprojekts aus. Es ist nun nicht mehr bekannt, ob das gesamte Softwaresystem auch nach den durchgeführten Änderungen die korrekte Funktionalität beibehält. Daher wird das Softwaresystem nach der abgeschlossenen Modifikation einem Regressionstest mithilfe von Testfällen unterzogen.[18]

[14] [Rö]
[15] [Lig09] [5.1.4.2]
[16] [Joh] [4a]
[17] [HK07] [S. 278]
[18] [Cha10] [8.2]

Es gibt nach einem Regressionstest nun zwei mögliche Ausgangssituationen:

- **Potenzielle Regressionen durch Änderung verursacht:** Das Verhalten des Softwaresystems der modifizierten Version stimmt bei mindestens einem Testfall nicht mit dem Verhalten der unmodifizierten Version überein. Hier liegt eine mögliche Regression vor und ein manueller Eingriff ist notwendig, um ein weiteres Vorgehen zu entscheiden.[19] Dieses weitere Vorgehen besteht dann zumeist aus einer Ursachenuntersuchung und anschließender Fehlerbehebung. Die Regressionstests müssen nach dieser Fehlerbehebung nochmal ausgeführt werden um sicherzustellen, dass keine weiteren Regressionen beim beheben der bereits vorher detektierten Regression eingeführt werden[20]. Ist die Abweichung des Softwaresystems durch eine Änderung in der Spezifikation gewollt, so muss der Testfall angepasst werden um das neue Verhalten zu reflektieren.
- **Keine Regressionen durch Änderung verursacht:** Es ist davon auszugehen, dass das getestete Softwaresystem fehlerfrei funktioniert und die durchgeführten Änderungen sicher übernommen werden können. Die geänderte Version stellt nun wieder ein korrektes System dar und gilt als Basis für folgende Regressionstests nach weiteren Änderungen.

Somit ist der abstrakte Ablauf eines Regressionstests behandelt, es bleibt allerdings zu klären, wie der eigentliche Testschritt, der die Regressionstests ermöglicht (in Abbildung 1 Blau dargestellt) durchgeführt wird.

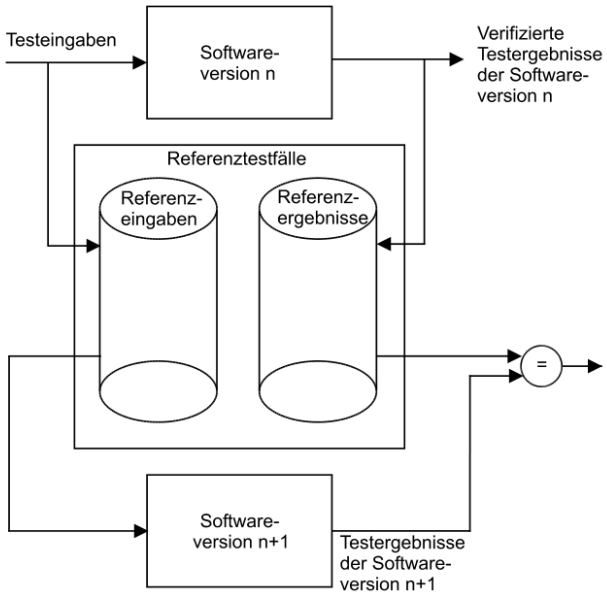


Abbildung 2: Prinzip des Regressionstests durch Verhaltensabgleich[21]

Regressionstests operieren wie in Abbildung 2 dargestellt mithilfe eines Verhaltensabgleichs zwischen dem Verhalten der modifizierten Softwareversion mit der Versionsnummer **N+1** und der validierten Basisversion mit der Versionsnummer **N**. Ein

[19] [Lig09] [S. 193]
[20] [Cha10] [8.4]
[21] Bildquelle: [Lig09] [S. 194]

Testfall besteht wie in Kapitel 2.1 beschrieben aus einer Eingabe der sogenannten Referenzeingabe und der damit assoziierten Ausgabe, dem sogenannten Referenzergebnis. Um einen neuen Testfall zu erstellen, wird Version **N** mit einer Testeingabe ausgeführt und das Verhalten der Software aufgezeichnet und anschließend als Referenzergebnis gespeichert. Im Verlauf dieser Ausarbeitung werden diese Referenzergebnisse der Version **N** als Soll-Verhalten definiert. Um nun einen Regressionstest auszuführen, wird Version **N+1** mit den gleichen Referenzeingaben der Testfälle ausgeführt und das Verhalten wieder aufgezeichnet. Im Verlauf dieser Ausarbeitung werden die Referenzergebnisse der Version **N+1** als Ist-Verhalten definiert. Anschließend findet ein Verhaltensabgleich zwischen Soll- und Ist-Verhalten statt und Differenzen dieses Abgleichs gelten als potenzielle durch Modifikation des Softwaresystems ausgelöste Regressionen, die einen manuellen Eingriff durch einen Stakeholder des Softwareprojekts erfordern.[22] Sollte eine Differenz des Soll- und Ist-Verhaltens expliziert erwünscht sein, da sich eine Anforderung an das Softwaresystem geändert hat, müsste das Ist-Verhalten als neues Soll-Verhalten im fehlschlagenden Referenztestfall definiert werden.

2.6 Automatisierte Testsysteme

Durch den in Kapitel 2.5 gewonnenen Einblick ist nun auch klar, warum eine Testautomatisierung bezüglich Regressionstests unabdingbar ist. Die Prozesse der Verhaltensaufzeichnung und des Verhaltensabgleichs sind repetitiv und manuell aufwendig durchzuführen. Wie schon in Kapitel 2.4 angesprochen, ist ein manuelles Testes aufgrund des hohen Wiederholungscharakters unwirtschaftlich für ein Softwareentwicklungsprojekt. Es wäre eine ganze Armada an Testern notwendig, um bei einem komplexeren Projekt mit einer großen Anzahl an Testfällen nach jeder Modifikation einen Regressionstest durchzuführen. Durch die repetitive Natur des manuellen Eingebens von Testeingaben und dem manuellen Vergleich des Ist- und Soll-Verhaltens können sich außerdem Fehler der menschlichen Natur in einen nicht automatisierten Regressionstest einschleichen.

Daher ist die Testautomatisierung mithilfe eines Werkzeugs für die wirtschaftliche Durchführung von Regressionstests notwendig. Es existieren bereits verschiedene plattformunabhängige Werkzeuge für Regressionstests, um z. B. grafische Benutzeroberflächen zu testen.[23] Sollte sich kein passendes Werkzeug für das Softwareprojekt finden sollte auch von einer Neuentwicklung eines passenden Werkzeugs nicht zurückgeschreckt werden. Eine zeitliche Investition in eine Regressionstestinfrastruktur lohnt sich langfristig, um bei einer neuen Version der Software keine Regressionsfehler mitauszuliefern.[24]

Ein solches automatisiertes Testwerkzeug sollte idealerweise die in Abbildung 1 und 2 dargestellten Operationen ausführen und somit die Verhaltensaufzeichnung der Referenzergebnisse übernehmen und außerdem den Abgleich des Soll-Verhaltens der stabilen Basisversion mit dem Ist-Verhalten der modifizierten Version übernehmen und somit bei Differenzen beim Verhaltensabgleich die im Projekt involvierten Stakeholder über die potenziellen entdeckten Regressionen informieren. Idealerweise werden die automatisierten Regressionstests nach jedem *Build* des Softwaresystems ausgeführt,

[22] [Lig09] [S. 194]
[23] [Lig09] [S. 194]
[24] [Joh] [4a]

um eine schnelle Übernahme der Modifikation, falls kein Fehlschlagen der Tests eintritt, zu garantieren und ansonsten potenzielle Regressionen schnellstens zu beseitigen. Die Ergebnisse der automatisierten Regressionstests sollten außerdem regelmäßig von Stakeholdern im Projekt gesichtet und besprochen werden, um bei Fehlalarmen die Testfälle anzupassen.[25] Hierbei kann auch auf bereits bestehende Konzepte hinsichtlich *DevOps* und *CI/CD* zurückgegriffen werden, um somit eine z. B. durch Werkzeuge wie *Jenkins*[26] oder *Gitlab Webhooks*[27] eine Regressionstest-Pipeline aufzubauen.

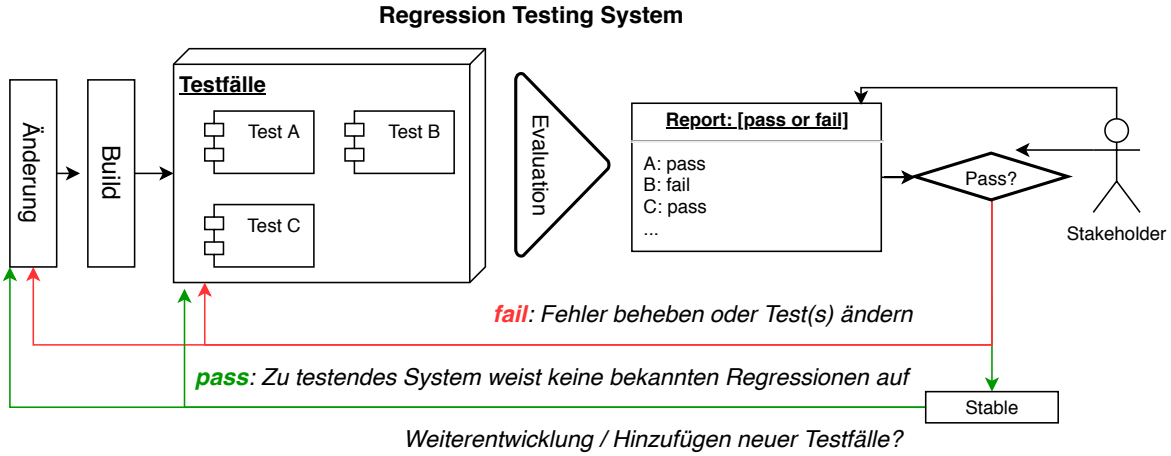


Abbildung 3: Möglicher Aufbau einer Regressionstest-Pipeline (eigene Darstellung)

In Abbildung 3 ist ein möglicher Aufbau einer solchen Regressionstest-Pipeline exemplarisch dargestellt. Nach einer Änderung am Softwaresystem wird automatisch der *Build*-Prozess ausgeführt und somit auch das Werkzeug für Regressionstests aufgerufen. Die verschiedenen Testfälle für das Softwaresystem werden auf die modifizierte Version angewandt und das Ergebnis wird in einen Testbericht geschrieben. Bei einem Bestehen der Regressionstests kann die modifizierte Version als neue stabile Basisversion übernommen werden und die Weiterentwicklung bzw. über das Hinzufügen neuer Testfälle geplant werden. Falls der Regressionstest fehlschlägt, ist allerdings manuelles Eingreifen durch Stakeholder des Projekts gefordert. Nun müssen entweder bei einem Fehlalarm (z. B. geänderte fachliche Anforderung d. h. geändertes Verhalten) die Testfälle geändert werden oder es muss eine Fehlerbehebung mit anschließender Wiederholung des Testvorgangs durchgeführt werden.

2.7 Testberichte

Nachdem in Kapitel 2.6 die Testautomatisierung bezüglich Regressionstests vorgestellt wurde und in Abbildung 3 ein Testbericht referenziert ist, muss nun klargestellt werden, welche Details ein Testbericht für einen Regressionstestmechanismus beinhalten sollte. Ein gut strukturierter Testbericht ist notwendig, um detektierte Regressionen nach einem Fehlschlagen des Regressionstests als Fehler oder Fehlalarm zu klassifizieren und sollte für diesen Anwendungsfall möglichst viele Details beinhalten, um mit dem Wissen über die getätigte Modifikation schnellstens eine Ortung der entdeckten Regression durchzuführen. Idealerweise sollten Testberichte durch das an der Software arbeitende

[25] [HK07] [S. 278-279]
[26] `https://www.jenkins.io/`
[27] `https://docs.gitlab.com/ee/user/project/integrations/webhooks.html`

Team gesichtet werden, ein guter Zeitpunkt stellt hier z. B. der Beginn des Arbeitstags der Teammitglieder dar. Regressionsfehler sollten außerdem vor dem Erweitern des Softwaresystems durch neue Funktionalitäten beseitigt werden und haben somit eine hohe Priorität. Gleiches sollte bei Fehlalarmen geschehen, um *False Positives* in Testfällen schnellstens durch Anpassung des Soll-Verhaltens zu beseitigen. Durch das Sichten des Testberichts im Team können weitere Fehler in der zukünftigen Entwicklung verhindert werden, da sich hier ein Lerneffekt bei den Teammitgliedern einstellt und erfahrenere Teammitglieder ihr Wissen über das Softwaresystem an unerfahrenere Teammitglieder weitergeben.[28]

Ein Testbericht sollte daher die folgenden Daten beinhalten[29]:

- Liste der im Regressionstest ausgeführten Testfälle.
- Anzahl aller ausgeführten, fehlschlagenden und erfolgreichen Testfälle und Fehlerrate.
- prozentuale Testabdeckung der Testfälle im Softwaresystem.
- Art des Problems bei Regressionen (z. B. Unerwartetes Ergebnis / Differenz der Ergebnisse, Absturz, Timeout, ...).
- Lokalisierungsinformationen (z. B. *Stacktrace* oder Informationen bezüglich der Eingabe/Ausgabe des Testfalls).

Im Idealfall ist es somit durch einen detailierten Testbericht und dem Wissen über die durchgeführte Modifikation schnell möglich, eine potenzielle Regression zu kategorisieren und abzuarbeiten.

2.8 Regressionstest-Techniken

Um Regressionstests durchzuführen, existieren verschiedene Testtechniken, die sind in Abbildung 4 dargestellt sind. Da es bei einem komplexen Softwaresystem mit vielen Testfällen meistens nicht wirtschaftlich ist, bei jeder Modifikation des Systems alle Testfälle auszuführen, werden im Folgenden diverse Testtechniken erörtert.

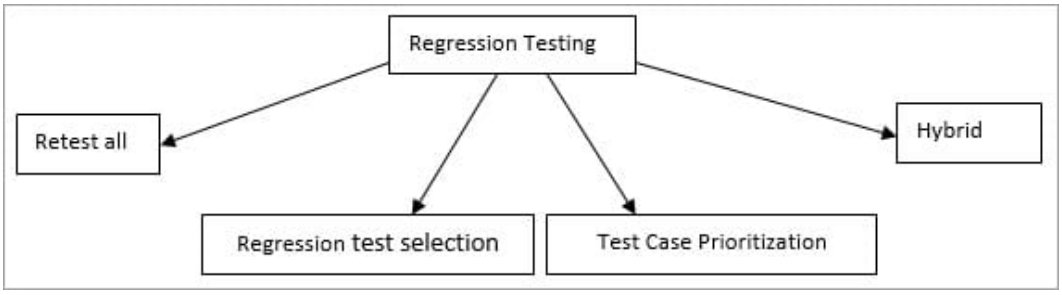


Abbildung 4: Überblick über unterschiedliche Techniken, um Regressionstests durchzuführen[30]

Es existieren als Testtechniken **Retest All**, **Regression Test Selection**, **Test Case Prioritization** und ein **Hybridansatz** bestehend aus einer Mischung von **Regression Test Selection** und **Test Case Prioritization**.[31]

[28] [HK07] [S. 278-279]
[29] [HK07] [8.5.2, 8.5]
[30] Bildquelle: [STH]
[31] [DS08]

Retest All stellt die teuerste Testtechnik dar, da hier alle bereits existierenden Testfälle zum Einsatz kommen. [32] Bei einem großen Softwaresystem kann das Ausführen aller Testfälle bei kleinen Änderungen schnell problematisch werden, da in der Zeit, in der die Regressionstests durchlaufen, eine Übernahme der Modifikationen nicht möglich ist. Der hohe Aufwand wird allerdings durch eine genaue Testabdeckung durch die Testfälle belohnt, unter der Verwendung aller Testfälle wird schließlich das gesamte Softwaresystem validiert. In der Praxis ist **Retest All** daher meist unpraktisch, da hier die Wirtschaftlichkeit im Vordergrund steht und lieber Abstriche in der Testabdeckung gegen den Gewinn von Zeit eingetauscht werden.

Bei der Testtechnik **Regression Test Selection** wird lediglich eine Untermenge aller Testfälle ausgeführt.[33] Dadurch ist ein Zeitgewinn möglich, da nicht mehr wie bei **Retest All** alle Testfälle ausgeführt werden.[34] Die Bestimmung einer solchen Untermenge kann z. B. durch die Analyse der getätigten Modifikation innerhalb des Quellcodes geschehen. Es werden demnach nur Testfälle ausgeführt, die Quellcode innerhalb des modifizierten Abschnitts zur Ausführung bringen.[35] Die Auswahl einer Selektionsstrategie für eine Untermenge an Testfällen stellt ein komplexes Problem dar[36] und wird in dieser Ausarbeitung aufgrund der Tiefe des Themas nicht weiter behandelt.

Test Case Prioritization ist eine Testtechnik, bei der jedem Testfall eine Priorität zugewiesen wird und die Testfälle dann geordnet von der höchsten bis zur niedrigsten Priorität ausgeführt werden. Ziel dieser Testtechnik ist es, potenzielle Regressionen schneller zu erkennen, indem häufiger fehlschlagende Testfälle zuerst ausgeführt werden.[37][38] Je nach Modifikation kann so eine Priorisierung z. B. aufgrund der geänderten Module oder dem betroffenen Quellcode geschehen.[39] Möglich ist es auch die Fehlerraten der Testfälle aufzuzeichnen und dann eine Ausführung geordnet der Fehlerrate durchzuführen.

Der **Hybridansatz** vermischt sowohl **Test Case Selection** und **Test Case Prioritization** und führt nach Priorität geordnete Testfälle einer bereits reduzierten Untermenge aller existierenden Testfälle aus.[40]

Schlussendlich stellt die Auswahl der Testfälle ein komplexes Problem bezüglich der Regressionstests dar, da hier immer ein Kompromiss zwischen der Zeit zum Durchlaufen aller Tests und dem betriebenen Aufwand zur Auswahl der Testfälle besteht. Ein **Retest All** ist zumeist in der praktischen Anwendung eine Utopie, wenn auch die sicherste Testtechnik. Bei guter Anwendung der anderen Testtechniken ist der gewonnene Zeitaufwand allerdings für den Entwicklungsprozess unabdingbar. Ein Risiko besteht dabei eine schlechte Auswahl der Testfälle für eine Untermenge zu treffen, da hier möglicherweise potenzielle Regressionen übersehen werden, da keine Ausführung passender Testfälle geschieht. Regressionstests fordern also ein stetiges vorsichtiges Abwägen zwischen Testabdeckung, Ausführungszeit und Aufwand der Testfallauswahl durch die im Projekt involvierten Stakeholder.

[32] [DS08] [2.1]
[33] [Cha10] [8.8]
[34] [DS08] [2.2]
[35] [Cha10] [S.262-263]
[36] [Cha10] [S. 262]
[37] [DS08] [2.3]
[38] [Cha10] [8.8.2]
[39] [Cha10] [12.6]
[40] [DS08] [2.4]

3 Fallstudie: Regressionstests für Graja

Da der Autor dieser Ausarbeitung im Rahmen seiner Bachelorarbeit[41] einen Regressionstestmechanismus für den an der Hochschule Hannover[42] in der Lehre eingesetzten Autobewerter für Lösungen studentischer Programmieraufgaben *Graja*[43] konzipiert und realisiert hat, wird in dieser kurzen Fallstudie ein Einblick in diesen Regressionstestmechanismus gewährt. Es ist so also möglich, die in Kapitel 2 erläuterten Konzepte in praktischer Anwendung zu sehen.

3.1 Notwendigkeit von Regressionstests in der Graja-Entwicklung

Der Autobewerter *Graja* stellt ein komplexes Softwaresystem dar, das vor der Arbeit aus ca. 60.000 Zeilen Java-Quellcode (Metrik: *SLOC - Source Lines of Code* d. h. keine Kommentare/Whitespace mitgezählt) besteht, welcher auf 31 *Gradle*[44]-Untermodule aufgeteilt ist. Modultests mittels *JUnit 4*[45] existieren für lediglich 3 dieser 31 Untermodule, wobei auch bei diesen keine vollständige Testabdeckung gegeben ist. Bezüglich der Untermodule existieren keine Integrationstests. Da *Graja* für die Lehre an der Hochschule Hannover missionskritisch ist, muss ein hoher Qualitätsstandard erfüllt werden, um Ausfälle im Vorlesungsbetrieb zu eliminieren. Die Ergänzung weiterer Modul- und Integrationstests stellt keine Alternative dar, da dieser Ansatz zu zeitaufwendig ist. Stattdessen muss geklärt werden, wie eine fehlerfreie Weiterentwicklung von *Graja* durch Regressionstests garantiert werden kann.

3.2 Definition und Auswahl der Testfälle

Im Kontext von *Graja* ist das korrekte Bewertungsverhalten von studentischen Einreichungen für die Aufgaben einer Vorlesung zu betrachten. Das Bewertungsergebnis einer beliebigen *Graja*-Aufgabe sollte nach Änderungen innerhalb eines *Graja*-Untermoduls bei gleichbleibenden Aufgaben-Verhalten reproduziert werden. Eine Ausnahme besteht hier natürlich, falls ein Testfall expliziert durch vorher unbekanntes Verhalten *„gebrochen“* wird. Solches vorher unbekanntes Verhalten könnte durch eine Änderung der Aufgabe oder einer geänderten Anforderung an *Graja* auftreten. In diesem Falle ist ein neuer Testfall, der das Verhalten der gewollten Änderungen abbildet, notwendig.

Für die Testfälle des Regressionstestmechanismus kann auf bereits bestehende Aufgaben mit den bereits bestehenden und dazugehörigen Musterlösungen zurückgegriffen werden. Der bisherige Verwendungszweck dieser Musterlösungen galt dem manuellen Testen einer *Graja*-Aufgabe durch eine grafische Oberfläche (*Graja GUI*). Wie bereits in Kapitel 2.6 erwähnt, ist ein solcher manueller Ansatz für Regressionstest unwirtschaftlich und kann daher nicht in Betracht gezogen werden. Musterlösungen simulieren studentische Einreichungen und werden in Verbindung mit der jeweiligen Aufgabe durch *Graja* bewertet. Diese können sowohl positiv (perfekte Bewertung: Ergebnis = 100%) als auch negativ (imperfekte Bewertung: Ergebnis < 100%) ausfallen, als auch sicherheitskritische Operationen (Ausführen von schädlichen *Shell*-Befehlen, DDoS-Attacken, ...), die durch *Graja* abgefangen werden, durchführen.

[41] [Her20]
[42] `https://www.hs-hannover.de/`
[43] `http://graja.hs-hannover.de/`
[44] `https://gradle.org/`
[45] `https://junit.org`

Für 57 solcher Aufgaben existieren bereits 437 Musterlösungen. Diese Kombination eignet sich gut für die Realisierung von Testfällen für einen Regressionstestmechanismus, da eine große Vielfalt der Aufgaben und Musterlösungen den gesamten Einsatzbereich von *Graja* abdeckt und diese direkt ohne manuelle Vorverarbeitung übernommen werden können.

3.3 Realisierung des Regressionstestmechanismus

Der entwickelte Regressionstestmechanismus arbeitet nach dem in Kapitel 2.5 beschriebenen Prinzip der Verhaltensaufzeichnung und des Verhaltensabgleichs. Referenztestfälle werden gemäß Abbildung 2 durch Musterlösungen und Aufgaben als Referenzeingabe und der resultierenden internen *Graja*-Datenstruktur als Referenzergebnis realisiert. *Graja* generiert aus dieser internen Datenstruktur ein HTML-Dokument, welches dann das finale Bewertungsergebnis darstellt. Für die Regressionstests ist dieses HTML-Dokument und der dazugehörige Transformationsschritt nicht von Relevanz, da dieser nach dem Bewertungsvorgang ausgeführt wird. Daher muss *Graja* im Kontext des Regressionstestmechanismus modifiziert werden, um an die interne Datenstruktur zu gelangen. Die Aufzeichnung besteht dann aus dieser internen Datenstruktur und dazugehörigen Metadaten.

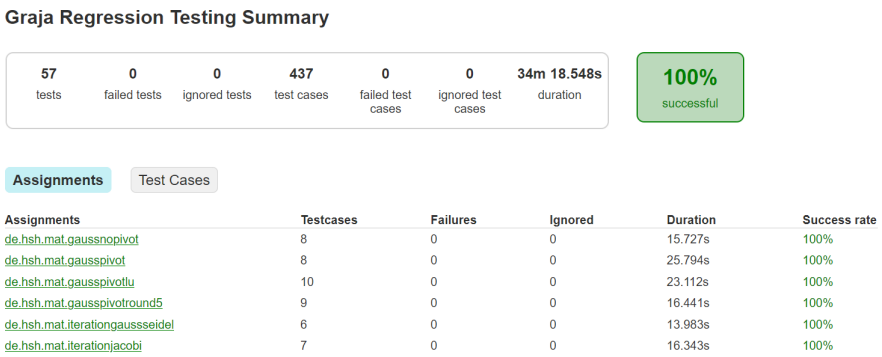
Der Verhaltensabgleich operiert dann mit dem Soll-Verhalten der validierten Basisversion und dem Ist-Verhalten der modifizierten *Graja*-Version. Aufgezeichnet werden unteranderem das Bewertungsschema, die generierten Kommentarbäume und Bewertungsabbrüche. Der Verhaltensabgleich ist durch einen inhaltlichen und strukturellen Vergleich der Kommentarbäume und des Bewertungsschemas realisiert. Bezüglich Bewertungsabbrüchen besteht ein binär operierender Vergleichsmechanismus.

Probleme während der Realisierung des Regressionstestmechanismus waren durch Nichtdeterminismus wie z. B. Aufgaben, die *Threading* beinhalten, gegeben. Für diese wurden mehrere Ausführungsstränge als gültige Referenzergebnisse gewertet, bei einem anschließenden Regressionstest reicht die Reproduktion eines aufgezeichneten Referenzergebnisses zum Bestehen aus. Um Nichtdeterminismus durch Zufallsgeneratoren zu vermeiden, wurden diese als *Mock*-Objekte erstellt und mit einem immer gleichbleibenden *Seed* initialisiert. Bezüglich der Verhaltensaufzeichnung musste auch eine Rauschunterdrückung entwickelt werden, um z. B. nicht gleichbleibende Zeitstempel und Dateipfade in den Aufzeichnungen zu neutralisieren. Um mit ähnlichen, jedoch nicht gleichbleibenden Zeichenketten umzugehen, wurde im Verhaltensabgleich ein *„fuzzy compare“* implementiert, um Fehlalarme durch eine Schwellwertoperation zu unterdrücken. Es existiert eine feingranulare Konfigurationsmöglichkeit auf Aufgaben- und Musterlösungsebene, um Fehlalarme durch spezielle Aufgabenstellungen mit nichtdeterministischen Verhalten zu vermeiden.

3.4 Testautomatisierung und Testberichte

Für die Testautomatisierung ist eine Schnittstelle über das *Build*-Werkzeug *Gradle* gegeben. Somit kann über einen Mechanismus wie z. B. *Gitlab Webhooks* eine Testautomatisierung nach einem *Commit* in das *Graja*-Repositorium stattfinden. Es werden an *JUnit* angelehnte Testberichte im HTML-Format generiert, die sich an den in Kapitel 2.7 empfohlenen Richtlinien für Testberichte halten.

In Abbildung 5 ist ein beispielhafter Ausschnitt des generierten Testberichts zu sehen. Es wird die Anzahl aller Testfälle nach *Graja*-Aufgaben gruppiert dargestellt und somit eine gesamte Fehlerquote der Regressionstests kalkuliert. Ausschnitte eines detaillierteren, auf die Aufgaben/Musterlösung-Kombination zugeschnittenen Testberichts sind in Abbildung 6 zu sehen.



Graja Regression Testing Summary

57 tests	0 failed tests	0 ignored tests	437 test cases	0 failed test cases	0 ignored test cases	34m 18.548s duration	100% successful

Assignments | Test Cases

Assignments	Testcases	Failures	Ignored	Duration	Success rate
de.hsh.mat.gaussnopivot	8	0	0	15.727s	100%
de.hsh.mat.gausspivot	8	0	0	25.794s	100%
de.hsh.mat.gausspivotlu	10	0	0	23.112s	100%
de.hsh.mat.gausspivotround5	9	0	0	16.441s	100%
de.hsh.mat.iterationgaussseidel	6	0	0	13.983s	100%
de.hsh.mat.iterationjacobi	7	0	0	16.343s	100%

Abbildung 5: Ausschnitt aus dem Testbericht des *Graja*-Regressionstestmechanismus

Der links-obere Teil in Abbildung 6 stellt einen fehlgeschlagenen Verhaltensabgleich im Bewertungsschema dar. Es sind der Pfad zum verletzenden Element im Bewertungsschema, eine Klassifikation nach potenzieller Regression und die verglichenen Rohdaten ersichtlich. Auf der rechts-oberen Seite hingegen ist ein fehlschlagender Vergleich zweier Zeichenketten zu sehen, im unteren Teil hängt auch ein *Stacktrace* an, der Aufschluss über die Herkunft dieser Zeichenkette im Quellcode gibt. Im unteren Teil ist ein Teilausschnitt eines Baumvergleiches zweier Kommentarbäume gegeben. Die Farbe Grün markiert identische Knoten, während Rot für Knoten, die im Soll-Verhalten fehlen, steht und Blau im Soll-Verhalten exzessive Knoten markiert. Orange Knoten deuten auf eine Existenz in beiden Referenzergebnissen hin, bei denen jedoch eine inhaltliche Diskrepanz vorliegt.

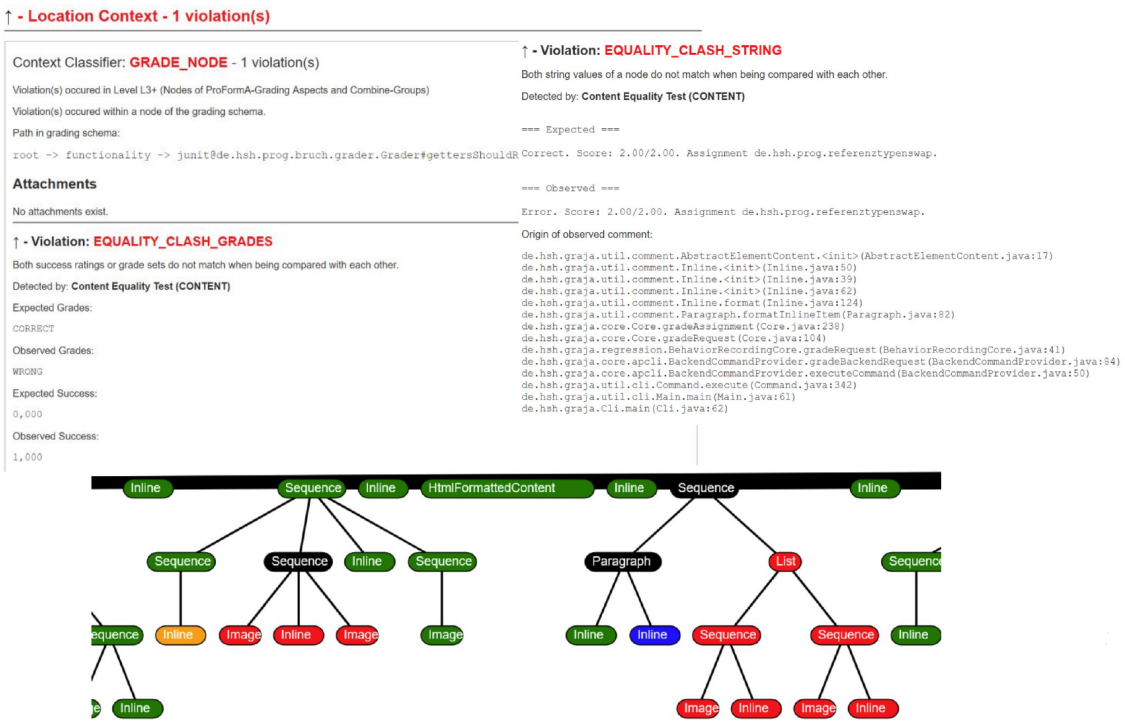


Abbildung 6: Ausschnitte aus der Detailansicht für potenzielle Regressionen

4 Abschließende Bewertung

Nachdem nun durch Kapitel 2 ein Einblick in Regressionstests gegeben wurde und mithilfe der Fallstudie aus Kapitel 3 eine praktische Demonstration des theoretischen Wissens geschehen ist, gilt es nun ein Fazit bezüglich Regressionstests zu ziehen.

4.1 Bewertung: Regressionstests im Allgemeinen

Regressionstests stellen in der heutigen Softwareentwicklung eine Notwendigkeit dar, um Modifikation an komplexen Softwaresystemen gewissenhaft durchzuführen. Es wird somit Effektiv das Entstehen von Pattsituationen bezüglich der korrekten Funktionalität eines Softwaresystems im Bezug zur Modifikation verhindert und somit der Qualitätsstandard der zu entwickelnden Software angehoben. Ohne Werkzeuge und Testautomatisierung sind Regressionstests allerdings nicht wirtschaftlich umsetzbar. Das Durchführen von Regressionstests erfordert außerdem gut definierte Referenztestfälle und ein stetiges Sichten des Testberichts. Bei fehlschlagenden Regressionstests sollte vom betreuenden Team eine sofortige Ursachenuntersuchung stattfinden, um die potenzielle Regression zu beheben oder die Testfälle den geänderten Anforderungen des Systems anzupassen. In der Praxis stellt das Ausführen von Regressionstests einen ständigen Kompromiss zwischen Testzeit und Auswahl der verwendeten Testfälle dar, da ein *Retest All* bei größeren Softwaresystemen selten wirtschaftlich durchführbar ist.

Trotz des positiven Einflusses in der Softwareentwicklung stellen Regressionstests ein komplexes Unterfangen dar. Große Softwaresysteme leiden meist unter einer langen Testausführung und auch die Erstellung von Referenzeingaben für Testfälle kann ein zeitaufwendiges Unterfangen darstellen. Bei fehlschlagenden Tests muss selbst bei einem automatischen Regressionstestmechanismus ein Stakeholder einspringen, um eine manuelle Sichtung des Testberichts durchzuführen. Die Kosten, welche für Regressionstests allokiert werden müssen, verringern außerdem das Budget für die Verbesserung der getesteten Software.[46] Trotz allem sind Regressionstests letztlich als positiv zu bewerten und sollten in der modernen Softwareentwicklung nicht ignoriert werden.

4.2 Bewertung: Regressionstests in der Graja-Entwicklung

Um den Regressionstestmechanismus für *Graja* zu bewerten, ist es noch zu früh. Das System ist erst seit September 2020 im Einsatz. Momentan baut eine weitere Bachelorarbeit auf diesem Regressionstestmechanismus auf und verwendet diesen für die Erweiterung von *Graja*. Erste Ergebnisse bezüglich der praktischen Anwendung im Entwicklungsalltag sollten also nach Abschluss dieser Arbeit vorliegen.

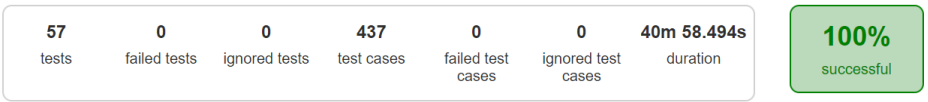


Abbildung 7: Teststrategie: *Retest All* - Ein langsames Unterfangen

Momentan verwendet dieser Regressionstestmechanismus die Testtechnik *Retest All*. Daher ist das Ausführen aller Regressionstests ein zeitlich aufwendiges Unterfangen, wie in Abbildung 7 zu sehen ist. In der Zukunft könnte dieses Problem entweder durch

[46] [Cha10] [8.7.1]

das Ändern der Testtechnik oder umstellen des Testmechanismus auf *Multi Threading* mit Priorisierung schnellerer Tests begegnet werden. Es ist außerdem ersichtlich, dass Nichtdeterminismus bezüglich Regressionstests in der Praxis ein Problem darstellt. Im Falle des *Graja*-Regressionstestmechanismus konnte dieses Problem gelöst werden, bei komplexeren Echtzeitsystemen könnte sich der Umgang mit Nichtdeterminismus in den Referenztestfällen als schwieriger gestalten.

4.3 Regressionstests im Kontext vom Projekt- und Qualitätsmanagement

Bei korrekter Umsetzung ist eine sofortige positive Auswirkung bezüglich der Qualität eines Softwaresystems bemerkbar. Regressionstests erlauben die Validierung von geänderten Abschnitten des Softwaresystems und validieren ebenso nicht verbundene Teile der Software, die möglicherweise durch Änderungen betroffen sind. Ebenso ist die Gewährleistung der gleichbleibenden Funktionalität des Softwaresystems nach einer Änderung überprüfbar. Die Verwendung von Regressionstests reduziert somit das Risiko schwerwiegender Fehler erheblich und sorgt somit für qualitativ hochwertige Software.[47]

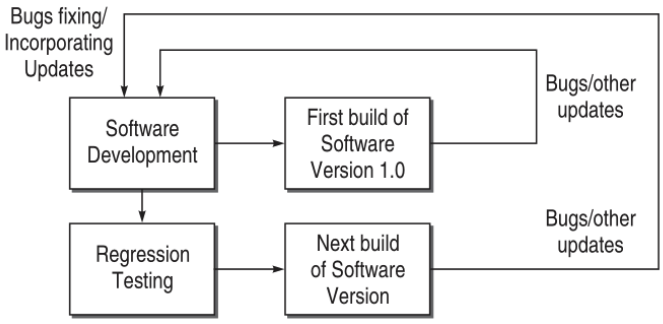


Figure 8.1 Regression testing produces Quality Software

Abbildung 8: Regressionstests führen zu qualitativer Software[48]

In Abbildung 8 ist ein durch Regressionstest gestützter Entwicklungsprozess visualisiert. Nachdem eine Basisversion (hier 1.0) einer Software entwickelt wurde, werden Regressionstests nach allen weiteren Modifikationen am Softwaresystem angewandt. Sollten diese Regressionstests fehlschlagen, findet erst eine Fehlerbehebung/Anpassung der Testfälle, falls durch geänderte Anforderungen von Nöten statt, bevor die modifizierte Basisversion als aktuelle stabile Softwareversion veröffentlicht wird. Es ist durch diesen Entwicklungsprozess also nicht mehr möglich, eine Softwareversion mit bekannten Regressionsfehlern zu veröffentlichen, da erst alle Regressionsfehler behoben werden müssen.

5 Ausblick

Durch diese Ausarbeitung wurde ein grundlegender Einblick in die Welt der Regressionstests gewährt. Es wurden sowohl theoretische Konzepte als auch deren praktische

[47] [Cha10] [8.2]
[48] Bildquelle: [Cha10] [S. 257]

Anwendung behandelt. Regressionstests sind in der heutigen Welt der Softwareentwicklung unabdingbar und somit ein Stützpfeiler der Software-Qualitätssicherung. Die korrekte Anwendung von Regressionstests stellt noch immer ein komplexes Problem dar, da hier viele Kompromisse eingegangen werden müssen. Besonders im Kontext der Laufzeit von Regressionstests und der Komplexität bezüglich der Auswahl von Testfällen für die Ausführung. Nichtsdestotrotz stellen Regressionstests eine gute Möglichkeit dar, um qualitativ hochwertige Software auszuliefern. So können z. B. auch neue Teammitglieder sorglos an einem Softwaresystem arbeiten, ohne mit der Angst leben zu müssen, durch ihr noch nicht vorhandenes Wissen über das Softwaresystem eine Regression als Nebeneffekt zu verursachen. Auch im Hinblick auf missionskritische Systeme in z. B. der Luftfahrt oder Medizintechnik bieten Regressionstest eine gute Möglichkeit, schwerwiegende Fehler zu verhindern und somit Menschenleben zu retten.

Literaturverzeichnis

[Cha10] Chauhan, Naresh: *Software Testing - Principles and Practices*. Oxford University Press, 2010. – ISBN 9780198061847

[DS08] Duggal, Gaurav ; Suri, Bharti: Understanding Regression Testing Techniques. (2008), 01

[FDA02] FDA: General Principles of Software Validation - Guidance for Industry and FDA Staff. (2002). https://www.fda.gov/regulatory-information/search-fda-guidance-documents/general-principles-software-validation

[Her20] Herschel, Marc: Musterlösungen in Graja als Mechanismus für Regressionstests. (2020). http://dx.doi.org/10.25968/OPUS-1721. – DOI 10.25968/OPUS–1721

[HK07] Huizinga, Dorota ; Kolawa, Adam: *Automated Defect Prevention*. John Wiley & Sons, Inc., 2007. http://dx.doi.org/10.1002/9780470165171. http://dx.doi.org/10.1002/9780470165171

[IEE90] IEEE Standard Glossary of Software Engineering Terminology. In: *IEEE Std 610.12-1990* (1990), S. 1–84. http://dx.doi.org/10.1109/IEEESTD.1990.101064. – DOI 10.1109/IEEESTD.1990.101064

[Joh] Johner, Christian: *Regressionstest: Ihre Software-Versicherung*. https://www.johner-institut.de/blog/iec-62304-medizinische-software/regressionstest, Abruf: 15.12.2020. – Version vom 31.01.2019

[Lig09] Liggesmeyer, Peter: *Software-Qualität - Testen, Analysieren und Verifizieren von Software*. Berlin Heidelberg New York : Springer-Verlag, 2009. – ISBN 978–3–827–42203–3

[Rö] Rössler, Jeremias: *5 Dinge, die Sie über Regressionstests wissen sollten*. https://www.informatik-aktuell.de/entwicklung/methoden/5-dinge-die-sie-ueber-regressionstests-wissen-sollten.html, Abruf: 15.12.2020. – Version vom 07.03.2017

[STH] *What is Regression Testing? Definition, Tools, Method, and Example*. https://www.softwaretestinghelp.com/regression-testing-tools-and-methods, Abruf: 15.12.2020. – Version vom 13.11.2020