Testautomatisierung – Gute Qualität fällt nicht vom Himmel

Maximilian Azimi, Jens-Rainer Felske, Sebastian Lauber, Jan-Henrich Mattfeld, Pascal Schneider, Krischan Stapelfeldt, Timm Suhl, Nils Techau, Karin Vosseberg

SG Informatik und Wirtschaftsinformatik
Hochschule Bremerhaven
An der Karlstadt 8
27586 Bremerhaven
kvosseberg@hs-bremerhaven.de

Abstract: Gerade in agilen Projekten ist mit Praktiken wie testgetriebener Softwareentwicklung (Test Driven Development) eine weitestgehende Testautomatisierung verbunden. Viele Unternehmen setzen gerade in die Testautomatisierung große Hoffnungen die Qualität ihrer Produkte nachhaltig zu verbessern. Eine effektive Testautomatisierung ist jedoch mehr als nur das Implementieren von Testfällen. Wie können Studierende auf diese Anforderungen vorbereitet werden?

1 Einleitung

Qualitätssicherung hat in der Praxis einen sehr hohen Stellenwert. Im Studium wird das Thema Qualitätssicherung oft noch vernachlässigt. Die konstruktiven Anteile der Entwicklung von Systemen stehen im Mittelpunkt der Ausbildung. Mit der Diskussion über eine testgetriebene Softwareentwicklung in Verbindung mit agilen Vorgehensmodellen rücken die Aufgabenbereiche der Rollen „Tester“ und „Entwickler“ immer stärker zusammen. Mit den Aufgaben der Testautomatisierung benötigen Tester verstärkt auch Softwareentwicklungskompetenzen sowie Entwickler Wissen über systematisches Testen. Im Rahmen des Moduls „Grundlagen des Qualitätsmanagement“ wurde an der Hochschule Bremerhaven für die Studierenden aus den Bachelorstudiengängen Informatik und Wirtschaftsinformatik eine Lernumgebung geschaffen, in der sie Erfahrungen in der testgetriebenen Softwareentwicklung sammeln konnten.

Der Schwerpunkt in der zugehörigen Vorlesung wurde auf die Vermittlung der Grundlagen der Softwarequalitätssicherung auf Basis des international anerkannten Lehrplans „ISTQB Certified Tester Foundation Level“ [CT11] gelegt. Begleitend zur Vorlesung wurde ein Projekt aufgesetzt in dem Studierende in Teams von 4 Personen eine kleine Gebäudeverwaltung testgetrieben entwickelt haben. Der Fokus in diesem Projekt lag jedoch nicht auf der Gebäudeverwaltung als Produkt, sondern auf den testgetriebenen Entwicklungsprozess und die damit verbundenen Aufgaben zur Testautomatisierung unter der Verwendung systematischer Testansätze. Die Aufgabe der Studierenden war für die Testautomatisierung eine adäquate Entwicklungs- und Testumgebung aufzubauen (vgl. [SBB11]), Testfälle systematisch auf Basis der vermittelten Testentwurfsmethoden auf

unterschiedlichen Teststufen zu entwerfen und auf dieser Basis die zugehörigen Funktionalitäten der Gebäudeverwaltung zu implementieren.

Unterstützend haben Kollegen der HEC GmbH Bremen[16] den Studierenden einen Einblick in reale Softwareentwicklungsprojekte gegeben und die von ihnen verwendeten Testumgebungen für eine Testautomatisierung vorgestellt. Darüber hinaus konnten Studierende aus der Veranstaltung während der iqnite 2014 an einem Workshop zur Testautomatisierung mit Jubula[17] teilnehmen. Mit diesem Wissen haben sie eine Testumgebung für ihren testgetriebenen Softwareentwicklungsprozess aufbauen können. Eine ähnliche Werkzeugkette ist in [HK12] beschrieben.

Im Anschluss an die Veranstaltung erhielten die Studierenden die Möglichkeit das zusätzliche Zertifikat „ISTQB Certified Tester Foundation Level“ zu erwerben.

2 Werkzeugkette für eine Testautomatisierung

Neben der Vermittlung von methodischem Wissen zu Verfahren der Softwarequalitätssicherung war ein wesentliches Ziel der Lehrveranstaltung den Studierenden die umfangreichen Aufgaben in der Testautomatisierung aufzuzeigen. Die Werkzeugkette für die Testautomatisierung wurde von den Teams parallel zur Entwicklung der Gebäudeverwaltung aufgesetzt. Die Grundlagen dafür wurden von gemeinsam in geblockten Übungseinheiten erarbeitet. Zwei Teams haben umfassende Lösungen für die gestellte Aufgabe erarbeitet.

In einem ersten Übungsblock haben sich die Studierenden mit den Anforderungen auseinandergesetzt, die aus einem Vorgehen im Sinne von Acceptance Test Driven Development (vgl. [Am06]) entstehen. Sie haben Werkzeuge untersucht, die es ihnen ermöglichen Testfälle auf der Systemtestebene zu automatisieren, die dann über eine (zu dem Zeitpunkt noch nicht vorhandene) graphische Oberfläche ausgeführt werden. Für die Erstellung ihrer Testfälle auf Systemtestebene haben sich die Teams für Jubula entschieden und haben hier ihre ersten Systemtestfälle implementiert. In dem nächsten Übungsblock wurden entlang der Umsetzung der Gebäudeverwaltung bezüglich der definierten Systemtestfälle erste Komponententests automatisiert. Da die Gebäudeverwaltung in Java implementiert wurde, haben die Teams für die Umsetzung der Komponententests JUnit genutzt. In einem dritten Übungsblock wurde das Thema Continous Integration aufgegriffen und ein Build-Prozess erarbeitet. Für die Aufgaben des Continuous Integrations haben sich die Teams einen Jenkins-Integrationsserver aufgesetzt und Maven als Build-Management-Werkzeug genutzt oder entsprechende Skripte mit Ant. Im Laufe des Projekts sind dann weitere Werkzeuge in den Build-Prozess integriert worden:

- **GitHub** – zur Verwaltung des Source Codes
- **Maven** – als Build-Management Werkzeug
- **Checkstyle** und **FindBugs** – zur statischen Analyse der Sourcen

[16] http://www.hec.de/startseite.html
[17] http://www.bredex.de/guidancer_jubula_de.html

- **Jubula** – Implementierung und Ausführung von Testfällen auf der Integrations- und Systemteststufe
- **JUnit** – Implementierung / Ausführung von Testfällen auf der Modulteststufe
- **JaCoCo** – zur Ermittlung der Codeabdeckung
- **Mantis** – zur Fehlerverfolgung
- **TestLink** – als Testmanagementwerkzeug zur Verwaltung von Testfällen.

Mit kleineren Variationen haben beide Teams eine ähnliche Testpipline aufgebaut, die sie gemeinsam in den Übungsblöcken diskutiert haben und sich hier gegenseitig in der Umsetzung unterstützt haben. Abbildung 1 zeigt beispielhaft die automatisierte Testpipeline eines Teams [Az14].

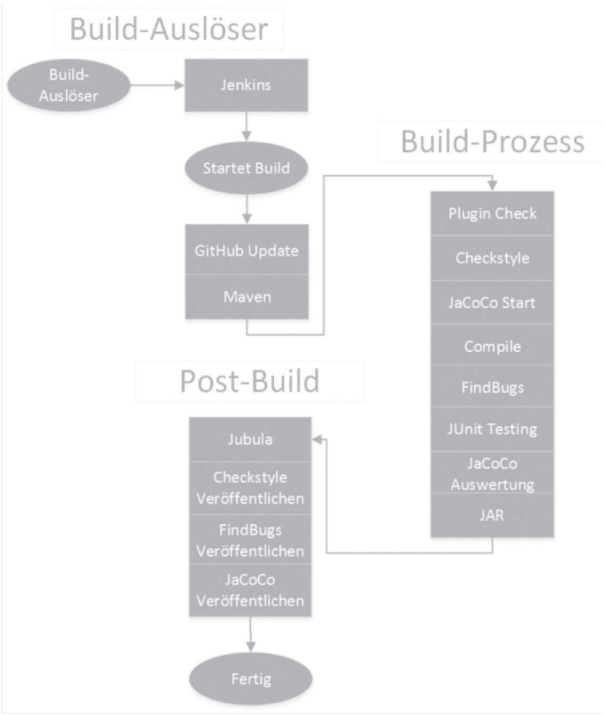

Abbildung 1: Automatisierte Testpipeline

Die Integration der Werkzeuge in den Continuous Integration Server war für die Teams eine großer Herausforderung, da sie sich in viele neue Werkzeuge und Schnittstellen einarbeiten mussten. Mit dem Aufsetzen der Testpipeline haben die Studierenden ein Ziel der Lehrveranstaltung erreicht. Darüber hinaus hatten die Studierenden jedoch auch die Aufgabe, den testgetriebenen Softwareentwicklungsprozess und die Qualität der automatisierten Testfälle im Auge zu behalten.

3 Systematische Testfallerstellung

Die automatisierte Ausführung von Testfällen führt alleine noch nicht zu einer effektiven Qualitätssicherung. Hier spielt die Qualität der erstellten Testfälle eine große Rolle. Um diese zu erhöhen, mussten die Teams den Prozess der testgetriebenen Softwareentwicklung wie ihn Scott Ambler mit einfachen Schritten beschreibt [Am06] erweitern. Abbildung 2 zeigt den erweiterten Prozess zur testgetriebenen Softwareentwicklung (vgl. [VS14]), der auch auf die Ebene von ATDD adaptierbar ist.

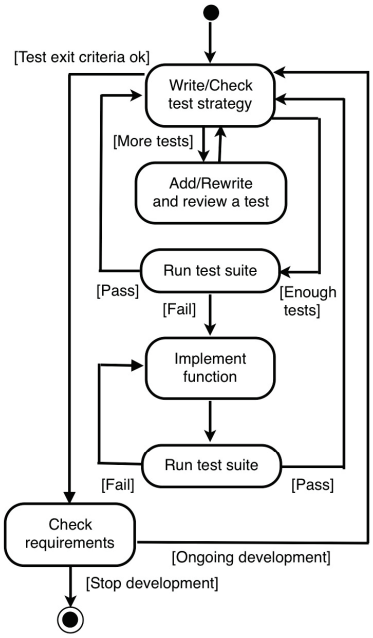

Abbildung 2: Erweiterter Prozess der testgetriebenen Softwareentwicklung

Für die systematische Testfallherleitung haben die Studierenden auf die in der Veranstaltung vermittelten Testentwurfsmethoden zurückgegriffen und abhängig von den zu entwickelnden Funktionalitäten, Methoden der Blackbox-Testentwurfsverfahren wie beispielsweise Äquivalenzklassenbildung oder Grenzwertanalysen, eingesetzt (vgl. [SL14]).

Darüber hinaus haben sie ihre Testfälle einem Review unterzogen und mit Whitebox-Testentwurfsverfahren analysiert. Sie haben damit sichergestellt, dass hinreichend viele Testfälle erstellt wurden, um Anforderungen wie beispielsweise eine 100%-Anweisungsabdeckung zu erreichen. Ihre Abdeckungsergebnisse haben sie in Form von JaCoCo Reports erstellt.

4 Erfahrungen der Studierenden

Auch wenn beide Teams am Ende zu ähnlichen, sehr guten Ergebnissen gekommen sind, wird in den Ausarbeitungen die unterschiedlichen Herangehensweisen und Erfahrungen sehr deutlich.

Das eine Team [Fe14] hat auf Basis der in der Aufgabe beschriebenen funktionalen Anforderungen und den zugehörigen Akzeptanzkriterien erste Mock-ups[18] der graphischen Oberfläche und ein Datenmodell erstellt und dies als Basis für die Implementierung erster Testfälle genutzt. Parallel dazu haben sie die Testpipeline aufgesetzt. Beim Aufsetzen des Continuous Integration Servers auf Basis von Jenkins haben sich Werkeuge wie beispielsweise Jubula als sehr sperrig erwiesen und viel Zeit in Anspruch genommen. Erste Testfälle hatten sie frühzeitig implementiert. „Im Laufe der Implementierung stellte sich jedoch heraus, dass einige Anforderungen und Annahmen am Datenmodell falsch waren, sodass Tests unsinnig oder logisch falsch angelegt waren. (...) Die Zeit, die das Aufsetzen der Build Services beanspruchte, fehlte immer mehr in der Entwicklungskapazität für die Anwendung, dabei wurde immer mehr vom TDD Ansatz abgewichen, da den fehlerhaften Tests zunächst keine Beachtung mehr geschenkt wurde.“ [Fe14] Das Team hatte seinen Fokus sehr stark einerseits das Aufsetzen der Testpipeline und andererseits auf die Entwicklung der Gebäudeverwaltung gelegt. Die Qualität der Testfälle ist zunächst etwas aus den Augen verloren worden. Damit beschreibt das Team eine typische Situation, wie wir sie immer wieder aus Diskussionen über testgestriebene Softwareentwicklungsprojekte zurückgespiegelt bekommen.

Diese ersten Erfahrungen haben in dem Team dazu geführt, ihre Vorgehensweise erneut zu reflektieren. Mit Fertigstellung ihrer Testpipeline haben sie ihre testgetriebene Softwareentwicklung konsequenter umgesetzt. „In der letzten Phase des Projektes lief der gesamte Testprozess besser. Es wurde diszipliniert getestet und programmiert. (...) Diese strukturierte Arbeitsweise führte zu deutlich weniger Fehlern im Programmcode und erleichterte uns auch die Durchführung des Projektes. Durch die Unterstützung der Toolchain, hatten wir ein mächtiges Werkzeug, das uns auch bildlich aufzeigte, dass unsere strukturierte Arbeitsweise qualitativ besser ist.“ [Fe14] Abbildung 3 zeigt sehr deutlich die verbesserte Abdeckung ihres Sourcecodes durch entsprechende Testfälle nach dem die Testpipeline etabliert war und sie konsequenter ihr testgetriebenes Vorgehen umgesetzt haben. Die Abbildung zeigt jedoch auch, dass noch nicht alle Bereiche ausreichend abgedeckt sind. Dies ist auf den begrenzten Zeitrahmen der Lehrveranstaltung zurückzuführen. Das Team hat die erzeugten Testfälle anforderungsbasiert, systematisch erfasst und in Testlink verwaltet. Informationen aus der statischen Codeanalyse von Checkstyle wurden verwendet, um beispielsweise fehlende Zweige in ihrem Programmcode zu erkennen.

[18] (Papierbasierte) Skizzen der Oberflächengestaltung

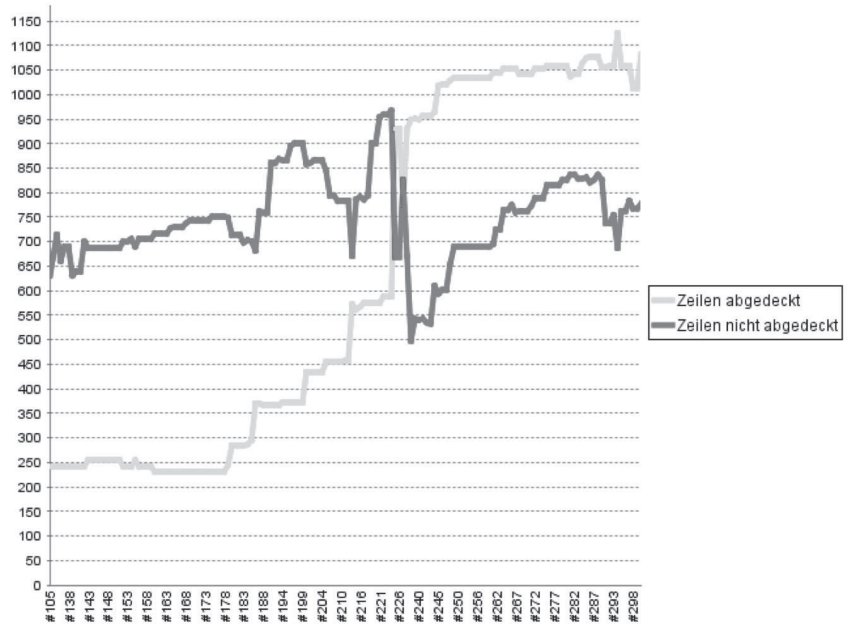

Abbildung 3: Trend Jacoco-Code-Abdeckung

Das zweite Team hat nicht die Umsetzung der Testpipeline in den Vordergrund gerückt, sondern sich sehr viel stärker auf die systematische Betrachtung der Testfälle fokusiert [Az14]. Angefangen haben sie mit einem Review der Anforderungen und den zugehörigen Akzeptanzkriterien bezüglich der Testbarkeit. Sie haben eine Zuordnung von Testentwurfsverfahren auf den verschiedenen Teststufen vorgenommen, um die entsprechenden Akzeptanzkriterien überprüfen zu können. Für die entwickelten Testfälle wurden explizit Testentwurfsverfahren wie Äquivalenzklassen und Grenzwertanalysen herangezogen. Dabei haben sie an konkreten Beispielen diskutiert, wo die gewählten Verfahren sinnvoll genutzt werden können und wo diese keinen weiteren Mehrwert bringen. Außerdem wurden Zustandsmodelle der Gebäudeverwaltung genutzt, um Testfälle herzuleiten. Auf der anderen Seite haben sie systematisch Whitebox-Verfahren eingesetzt, um eine entsprechende Code-Abdeckung durch ihre Testfälle zu erreichen. Gerade komplexe Codesegmente haben sie mit Hilfe von Kontrollfussgraphen analysiert, um die Qualität ihrer bis dahin entwickelten Testfälle zu prüfen. Darüber hinaus haben sie an Beispielen aufgezeigt, wo System- und Integrationstestfälle Fehler aufzeigen, die Unittestfälle nicht finden konnten. „Konkretes Beispiel: Die InputFilter eines Textfeldes werden darüber nicht geprüft. Es kann also sein, dass eine korrekte Eingabe trotz erfolgreicher Unit-Tests gar nicht möglich ist – Der InputFilter verhindert sie: Eine bisher unbemerkter Fehlerwirkung.“ [Az14]

Ihre zu Beginn definierten Testendekriterien – alle in der Aufgabe definierten Akzeptanzkriterien müssen mit mindestens einem Testfall abgedeckt sein und eine 100% Anweisungsabdeckung ist zu erreichen – hat dieses Team am Ende der Projekts mit allen ihren unterschiedlichen Maßnahmen erreicht. Ihr Fazit lautete: „Viel Spaß hat das Vorge-

hen mithilfe von Pair Programming gemacht. Auch die enge Zusammenarbeit zwischen Entwicklern und Testern hat sich bewährt. Die aufgebaute Toolchain wurde sehr gut (von allen Teammitgliedern) aufgenommen und hat einen kontinuierlichen Workflow ermöglicht. Denn unangenehm waren in vorherigen Projekten vor allem die manuellen Regressionstests. Durch die nun vorhandene Toolchain mit vorbereiteten Testfällen entfällt dieser Aufwand. Häufig konnte so eine bisher verdeckte Fehlerwirkung gefunden und automatisch dokumentiert werden. Entsprechend hoch ist die Akzeptanz des neuen Vorgehens. Abschließender Eindruck: Abzüglich des einmaligen Aufwands zur CI-Server-Konfiguration wurde Software schneller und in einer besseren Qualität erstellt!“ [Az14]. Darüber hinaus haben sie auch die Reichweite ihrer Testautomatisierung an konkreten Beispielen reflektiert. „Der automatische Test der Akzeptanzkriterien garantiert nicht automatisch eine hohe Userakzeptanz. Sehr deutlich wird das am Beispiel des Komponentenbaums in der Gebäudeverwaltung: Das Standardverhalten schließt den Komponentenbaum nach jeder Add/Remove-Aktion des Nutzers. Für den automatischen Test durch Jubula ist das kein Problem – der Computer klickt sich rasend schnell wieder an die richtige Stelle. Ein Benutzer muss aber umständlich händisch den Baum aufklappen. Die Spezifikation ist an dieser Stelle erfüllt. Usability wird aber nicht überprüft. Auch die Mockups haben das Problem nicht verdeutlicht. Ganz klar wird also die Notwendigkeit von häufigen (manuellen) Anwendertests direkt an den jeweiligen Prototypen.“ [Az14]

5 Fazit

Wenn der Trend agiler Softwareentwicklungsprojekte und damit verbunden Praktiken wie testgetriebene Softwareentwicklung verstärkt eingesetzt werden, müssen wir die Studierenden auf diese Anforderungen vorbereiten. Im Vergleich zu vorherigen Erfahrungen aus Lehrveranstaltungen mit dem Inhalt „ISTQB Certified Tester Foundation Level“ hat die Kombination aus theoretischem Wissen und einem eng darauf bezogenen Projekt als Experimentierraum einen großen Mehrwert für die Studierenden gebracht. Die Ausarbeitungen haben gezeigt, dass die Studierenden sehr reflektiert die theoretischen Konzepte der Softwarequalitätssicherung in ihren Projekten umsetzen konnten.

Von den Teams wurde das Etablieren eines Acceptance Test Driven Development Prozesses zunächst unterschätzt. Dies hat sich vor allem bei der Entwicklung der Testfälle gezeigt. Während des Projekts haben die meisten Teammitglieder ihre ersten Erfahrungen mit Softwaretests gemacht. Der initiale Aufwand zur Einrichtung der Testumgebung war sehr hoch. Die Aufgabe hat ihnen aber auch einen guten Einblick in die Integrationsprobleme gegeben. Nicht alle Werkzeuge ließen sich in einfacher Weise in den Jenkins Integrationsserver einbinden.

Abzüglich des einmaligen Aufwands für den Aufbau der Testumgebung beschreiben die Studierenden, dass die Software schneller und in einer besseren Qualität erstellt wurde (ähnliche Aussagen sind in [La05] enthalten). Aus ihrer Sicht führte die testgetriebene Softwareentwicklung mit dem Continuous Integration und der Testautomatisierung im Hintergrund zu einer besseren Qualität in ihrer Gebäudeverwaltung. Gerade die beiden Teams, die eine sehr umfangreiche Lösung erarbeitet haben, beschreiben, dass sie ihre erstellte Entwicklungs- und Testumgebung auch für weitere Projekte einsetzen werden.

Die Ausarbeitungen zeigen jedoch auch deutlich, dass ein sehr hohes Engagement der Studierenden und eine bestimmte Größe der Teams erforderlich sind, um alle Anforderungen der Aufgabenstellung erfüllen zu können. Hinzu kommt, dass einzelne Teammitglieder umfassende Programmierkompetenzen mitbringen müssen. Kleinere Teams waren mit der Aufgabe überfordert. Hier kann eine Vorgabe einer installierten Werkzeugkette zur Testautomatisierung sinnvoll sein, damit sich diese Teams auf eine systematische Erstellung von Testfällen im Rahmen von ATDD konzentrieren können.

Literaturverzeichnis

[Az14] Maximilian Azimi, Jan-Henrich Mattfeld, Pascal Schneider, Nils Techau: Qualitätsmanagement im Softwareentwicklungsprozess - Entwicklung einer Gebäudeverwaltung mit Test-Driven Development und Continuous Integration. Hochschule Bremerhaven, Ausarbeitung im Rahmen des Moduls Grundlagen Qualitätsmanagements, 2014.[19]

[Am06] Scott W. Ambler: Introduction to Test Driven Development. 2006. http://www.agiledata.org/essays/tdd.html (letzter Zugriff: 2014/10/15)

[CT11] ISTQB Certified Tester Foundation Level Syllabus. http://www.german-testing-board.info/fileadmin/gtb_repository/downloads /pdf/lehrplan/2011-ctfl/ ISTQB_CTFL_Lehrplan_2011_germ_V101.pdf (letzter Zugriff: 2014/10/15)

[Fe14] Jens Felske, Sebastian Lauber, Krischan Stapelfeldt, Timm Suhl: Grundlagen Qualitätsmanagement. Hochschule Bremerhaven, Ausarbeitung im Rahmen des Moduls Grundlagen Qualitätsmanagements, 2014.[4]

[La05] J. Langr: Agile Java. Crafting Code with Test-Driven Development. Prentice Hall 2005

[HK12] A. Heidt, S. Kleuker: Kontinuierliche Prozessverbesserung durch Testautomatisierung. In: H. Brandt Pook, A. Fleer, T. Spitta, M. Wattenberg (Hrsg.): Nachhaltiges Softwaremanagement. Proceedings Softwaremanagement 2012, LNI 209, 2012, S. 69-78

[SBB11] R. Seidl, M.; Baumgartner, T. Bucsics: Basiswissen Testautomatisierung dpunkt 2011

[SL14] A. Spillner, T. Linz: Basiswissen Softwaretest. 5. Auflage, dpunkt 2012.

[VS14] K. Vosseberg, A. Spillner: The Dark Side of TDD. 6th World Congress on Software Quality, 1st-3rd July 2014, London.

[19] Die Ausarbeitung ist öffentlich nicht zugänglich, kann jedoch bei Interesse zugeschickt werden