Müller, Neumair, Reiser, Dreo-Rodosek (Hrsg.): 10. DFN-Forum Kommunikationstechnologien,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 11

Eine vollautomatisierte e-Learning Plattform am Beispiel eines Universitäspraktikums

Jan Schmidt[1], Nils gentschen Felde[1]

Abstract:

Das Skalierungsverhalten vieler universitärer Praktika ist den derzeitig stetig wachsenden Teilnehmerzahlen häufig nicht gewachsen. Insbesondere die zumeist sehr zeitaufwändige Korrekturarbeit durch das Betreuungspersonal verhindert größere Teilnehmerzahlen, was zu einem Mangel an verfügbaren Praktikumsplätzen führt. Ziel dieser Arbeit ist es, die notwendigen Konzepte und ein System zu entwickeln und zu implementieren, das es ermöglicht, Praktikumsaufgaben vollautomatisiert zu überprüfen. Außerdem soll den Studenten die Möglichkeit gegeben werden, ihre Lösungen bereits während der Implementierung zu überprüfen, um ein eigenständiges Lernen zu unterstützen. In dieser Arbeit werden anhand beispielhafter Praktika Anforderungen an ein solches Prüfungssystem erhoben und daraus ein Systementwurf abgeleitet. Der Systementwurf wird beispielhaft für das Praktikum IT-Sicherheit implementiert. Die Implementierung wird durch eine Kombination aus dem Python-Framework *Flask* und der Task-Queue *Celery* realisiert, um eine skalierbare Web-Anwendung zu erhalten. Die Praxistauglichkeit wird direkt durch den produktiven Einsatz der erhaltenen Lösung an der LMU München belegt.

Keywords: e-Learning, Virtuelle Lehr- & Forschungsumgebungen, Automated Grading

1 Einleitung

In Zeiten stetig steigender Studentenzahlen haben sich im Laufe der Zeit bereits eine Menge technischer Entwicklungen ergeben, die das Skalierungsverhalten von Lehrveranstaltungen (insbesondere von Praktika) erheblich positiv beeinflussen. Im Falle der LMU bedeutet das z. B., dass viele der technisch notwendigen Arbeitsumgebungen und IT-Infrastrukturen virtualisiert wurden [DGK16; Li05; Li06; LRg08; Sc11], so dass eine immer größere Zahl an Studenten vollkommen zeit- und ortsungebunden an einem Praktikum teilnehmen kann. Die Kehrseite der Medaille jedoch ist, dass die persönliche Betreuung und das inhaltliche Anleiten der Studenten – vor allem bedingt durch die exzessive Nutzung der neu gewonnenen Freiheiten bei der Orts- und Zeitwahl und der folglich geringeren Präsenzzeit an der Universität – immer schwieriger wird.

Aus diesem Grund wird in dieser Arbeit ein System entworfen, das die Übungsaufgaben im Rahmen eines Praktikums vollautomatisiert korrigieren kann und so den Korrekturaufwand

[1] MNM-Team, Ludwig-Maximilans-Universität München, Oettingenstr. 67, 80538 München, Germany, Email: {schmidtja,felde}@nm.ifi.lmu.de

für die Betreuer verringert. Auch soll den Studenten die Möglichkeit geboten werden, dass sie ihre Lösungsansätze bereits während der Bearbeitung der Praktikumsaufgaben testen können. Dazu werden im folgenden Kapitel themenverwandte Arbeiten analysiert, bevor Kapitel 3 durch eine Anwendungsfall-getriebene Anforderungsanalyse einen Kriterienkatalog für einen Systementwurf hervorbringt. Auf Basis dieses Kriterienkatalogs wird in Kapitel 4 eine Systemarchitektur abgeleitet und deren Implementierung kurz umrissen, bevor Kapitel 5 die erhaltene Lösung bewertet und Kapitel 6 die Ergebnisse zusammenfasst und einen Ausblick auf weiterführende Arbeiten gibt.

2 Themenverwandte Arbeiten

Es existiert eine Menge Arbeiten und Ansätze zu automatisierten e-Learning Umgebungen, im Rahmen derer diverse Übungsaufgaben durch die Teilnehmer / Studenten bearbeitet und gelöst werden müssen. Die Überprüfung der gelösten Aufgaben erfordert jedoch häufig Humaninteraktion, zumeist von freiwilligen Helfern, was das Skalierungsverhalten der angebotenen Initiativen stark von der Anzahl der Betreuer abhängig macht. Ziel dieser Arbeit ist, genau diese Lücke zu schließen.

Eine bekannte Initiative zur Ausbildung im Bereich IT-Sicherheit ist das sog. *Hacking-Lab*[2]. Das *Hacking-Lab* ist eine Online-Plattform, die eine vernetzte IT-Infrastruktur bereitstellt, innerhalb welcher verschiedene Aufgaben und Herausforderungen zur IT-Sicherheit gelöst werden können. Ein *Hacking-Lab* besteht aus sog. „challenges“, die jeweils aus einer existierenden Schwachstelle bestehen, die es zu finden und beheben gilt. Eine Lösung besteht aus der Beschreibung einer gefundenen Schwachstelle und der Dokumentation einer entsprechenden Absicherung. Eine Bewertung der Lösungen wird von den Betreuern des *Hacking-Lab*s vorgenommen und per Email an die Teilnehmer versendet. Das *Hacking-Lab* basiert technisch auf dem *Open Web Application Security Project*[3] und stellt u. a. auch die Missionen im Rahmen der *European Cyber Security Challenge*[4] bereit. Ähnliche Arbeitsumgebungen, in denen insbesondere Pen-Testing gelehrt und geübt werden kann, sind das *Hacking Dojo*[5], das *Virtual Hacking Lab*[6] sowie *LAMPSecurity*[7].

Viele Vorarbeiten zur automatischen Korrektur von Programmieraufgaben finden sich unter dem Stichwort *Automated Grading* [Bl04; EP08; HPK11; KLC01; SHS15]. Sämtliche Arbeiten beschränken sich allerdings auf die Korrektur von Programmieraufgaben und eignen sich nicht dafür, beliebige praktische Aufgaben, die innerhalb eines universitären Praktikums oder *Hacking-Labs* existieren, zu korrigieren. Ein großer Teil der Lösungen ist jedoch web-basiert und kann als gute Basis für das angestrebte System dienen.

[2] https://www.hacking-lab.com/index.html
[3] https://www.owasp.org/index.php/OWASP_Hacking_Lab
[4] http://www.europeancybersecuritychallenge.eu/
[5] http://hackingdojo.com/lab/
[6] https://sourceforge.net/projects/virtualhacking/
[7] https://sourceforge.net/projects/lampsecurity/

Ein Prüfungssystem für ein virtualisiertes Praktikum ist in [BR13] dokumentiert. Es werden virtuelle Maschinen auf den Systemen der Teilnehmer durch eine Sammlung von Shell-Skripten überprüft und eine entsprechende Bewertung generiert. Zur Überprüfung werden die jeweiligen Tests auf die zu überprüfenden Systeme der Teilnehmer kopiert und ausgeführt. Das führt folglich leider dazu, dass Testroutingen potentiell durch den Teilnehmer eingesehen und im schlimmsten Fall sogar manipuliert werden könnten.

3 Anforderungsanalyse

Auch an der LMU existieren bereits Vorarbeiten und voll-virtualisierte Arbeitsumgebungen im Bereich der Lehre [DGK16; Li05; Li06; LRg08; Sc11], an deren Beispiel eine möglichst weitreichende automatische Korrektur von zu lösenden Übungsaufgaben realisiert werden soll. Diese Arbeiten dienen im Folgenden als Ausgangsbasis für die vorliegende Arbeit. In einem ersten Schritt werden die an den Praktika beteiligten Akteure und ihre Rollen identifiziert. Erwartungsgemäß sind dies die studentischen Teilnehmer, die Lehrstuhlmitarbeiter in ihrer Rolle als inhaltliche Betreuer und Prüfer des Praktikums sowie die technischen Administratoren der unterstützenden IT-Infrastrukturkomponenten.

Die unterschiedlichen Sichtweisen der drei beteiligten Rollen (Student, Betreuer, Administrator) dienen als Ausgangsbasis für eine Anwendungsfall-getriebene Anforderungsanalyse. Die dabei angewandte Methodik folgt der aus dem objektorientierten Software-Entwurf bekannten Vorgehensweise der Analyse von Anwendungsfällen (sogenannte *Use Cases*). Der Ausgangspunkt der Anwendungsfallanalyse ist die knappe und informelle Beschreibung ausgewählter Szenarien. Zur Ableitung von Anwendungsfällen werden die vier Phasen des Lebenszyklus einer Praktikums-Instanz (Planung, Aufbau / Inbetriebnahme, Betrieb inkl. Prüfung, De-Kommissionierung) und zusätzlich die Evolution (Änderung, Erweiterung etc.) des Praktikums über die Jahre hinweg betrachtet. Die Beschreibung der verschiedenen Lebenszyklusabschnitte aus den jeweils unterschiedlichen Blickwinkeln wird als Grundlage für die Analyse der verschiedenen Anwendungsfälle verwendet. Insgesamt leiten sich nach diesem Schema 29 Anwendungsfälle ab [Sc16], die sich auf die vier Lebenszyklusphasen verteilen. Beispiele für Anwendungsfälle sind das Gruppieren von Übungsaufgaben zu Aufgabenblättern, die Zuordnung von Studenten zu Arbeitsgruppen, Änderungen an den Arbeitsgruppen während des Semesters (z. B. Wechsel eines Studenten in eine andere Gruppe, frühzeitiges Ausscheiden eines Studenten, o. ä.), die Archivierung und Reproduzierbarkeit von Testergebnissen oder schlicht die Integration des angestrebten Systems in die bereits existierende Betriebsumgebung. Aus der Summe der Anwendungsfälle werden Anforderungen abgeleitet und in einem Anforderungskatalog gesammelt.

Zusammenfassung der Ergebnisse der Anforderungsanalyse

Zusammenfassend ergeben sich 29 funktionale und 15 nicht-funktionale Anforderungen. Neben den durch die Anwendungsfälle abgeleiteten Anforderungen ergibt sich noch eine weitere Menge an Anforderungen aus formalen und rechtlichen Vorgaben, die z. B. in Prü-

fungsordnungen und sonstigen Studienregularien sowie der Gesetzgebung festgeschrieben sind. Die meisten der hierdurch gegebenen Anforderungen sind Duplikate der bereits im Rahmen der Anwendungsfall-getriebenen Anforderungsanalyse erhobenen Anforderungen (z. B. Reproduzierbarkeit der Ergebnisse, Archivierung etc.). Einige andere Anforderungen hingegen erweisen sich im weiteren Verlauf der Arbeit als nur sehr kompliziert oder gar nicht IT-gestützt umsetzbar (z. B. die eindeutige Zurechenbarkeit von Ergebnissen zu Studenten, insbesondere zum Zwecke der Notengebung).

4 Systementwurf und Implementierung

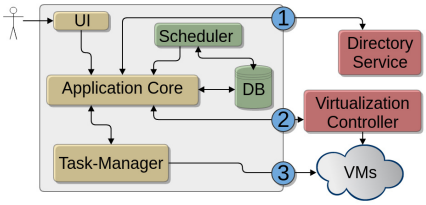

Abb. 1: erster Systementwurf

Um die Bestandteile des Systems und ihre Beziehungen untereinander festzulegen, werden in einem ersten Schritt die erhobenen Anforderungen gruppiert und daraus Komponenten abgeleitet. Abb. 1 zeigt einen ersten groben Überblick über die abgeleiteten Komponenten, Kommunikationsbeziehungen und ihren Schnittstellen zu Bestandssystemen aus der Vogelperspektive.

Aus den Anforderungen an die Datenhaltung (wie z. B. die Verwaltung von Teilnehmerdaten, Terminen und Fristen oder die Zuordnung von Aufgaben zu Übungsblättern) folgt, dass eine zentrale Komponente zur Speicherung relevanter Daten und Informationen (also eine Datenbank (*DB*)) benötigt wird. Zudem beinhaltet der Anforderungskatalog einige Anforderungen an die Interaktionen mit bestehender Infrastruktur, Diensten und den bestehenden virtuellen Maschinen (vgl. rechte Seite in Abb. 1). Der Systementwurf wird diesen Anforderungen in Form von drei Schnittstellen (①, ② und ③) gerecht:

Schnittstelle ① stellt die Kommunikation mit einem existierenden Verzeichnisdienst (*Directory Service*) sicher, der zur Authentifizierung und Autorisierung registrierter Nutzer verwendet werden soll.

Schnittstelle ② realisiert die Kommunikation mit der bestehenden Virtualisierungsumgebung unter Nutzung existierender Management-Komponenten (*Virtualization Controller*). Der notwendige Funktionsumfang der Schnittstelle ergibt sich aus einer Gruppe nichtfunktionaler Anforderungen, die fordern, dass z. B. Testergebnisse reproduzierbar sind, keine Spuren auf den getesteten Systemen hinterlassen werden, und dass Tests nicht durch Dritte beeinflussbar sind. Daraus folgt ebenfalls, dass der Funktionsumfang dieser Schnittstelle zumindest das Sichern des Systemzustandes einzelner VMs sowie die Zugriffssteuerung und das Management von VMs unterstützen muss.

Schnittstelle ③ stellt die eigentliche Kommunikation zwischen dem Prüfungssystem und den virtuellen Maschinen sicher. Zum einen müssen die VMs vor einem Test vorbereitet werden (z.B. einrichten geeigneter Firewall-Regeln), zum anderen müssen Testroutinen Zugriff auf das System erhalten, um dort ausgeführt werden zu können.

Die Kommunikation unter Nutzung der vorgesehenen Schnittstellen sowie die Verwaltung der zentralen Abläufe und das Auflösen potentieller Abhängigkeiten von Aufgaben (*Task*s) untereinander soll durch einen zentralen Applikations-Kern (*Application Core*) übernommen werden. Teil dieses Kerns ist auch ein *Scheduler*, der u. a. den Forderungen nach Skalierbarkeit, Fehlertoleranz und Nebenläufigkeit (z. B. durch die parallele Testausführung mehrerer Übungsgruppen) gerecht wird. Zudem ist im Systementwurf ein eigener *Task-Manager* als *Producer* nach dem *Producer-Consumer*-Schema vorgesehen, der die auszuführenden *Task*s erstellt, die von *Consumern* als eigenständige Prozesse abgearbeitet werden.

Aus einer weiteren Gruppe von Anforderungen ergibt sich schlussendlich die Notwendigkeit einer Sichtenbildung und eines Rollenkonzepts. Das bedeutet für den Systementwurf, dass eine Nutzerschnittstelle (*User Interface (UI)*) mit unterschiedlichen Sichten entsprechend einem festgelegten Rollenkonzept benötigt wird. Zudem beeinflussen viele nicht-funktionale Anforderungen (vor allem an die Sicherheit) die Festlegung von Protokollen und bedingen die Wahl geeigneter kryptographischer Verfahren, wie auch im weiteren Verlauf dieses Kapitels genauer beschrieben.

4.1 Detaillierter Systementwurf am Beispiel des Praktikums IT-Sicherheit

Auf Basis der einleitenden Überlegungen lässt sich eine verfeinerte Systemarchitektur als Grundlage für eine spätere Implementierung leicht an einem Beispiel darstellen. Abb. 2 stellt das Ergebnis graphisch dar.

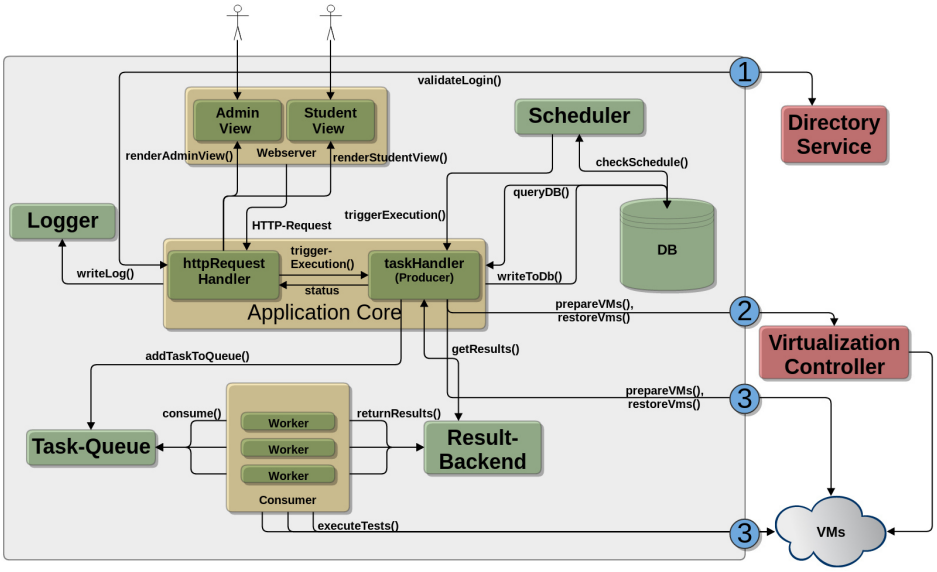

Abb. 2: Verfeinerte Systemarchitektur [Sc16]

Webserver. Die geforderte Sichtenbildung des Systementwurfs wird durch einen Webserver mit unterschiedlichen Sichten für administrative Aufgaben und für Studenten realisiert.

Application Core. Der *Application Core* stellt die zentrale Komponente des Gesamtsystems dar. Es werden alle Anfragen des Webservers verarbeitet, aus denen sogenannte *Task*s erzeugt werden, die wiederum jeweils von einem *Worker* verarbeitet werden. Außerdem wird die Verarbeitung der *Task*s überwacht. Dementsprechend teilt sich der *Application Core* in zwei Teilmodule auf, die die beiden Teilaufgaben funktional abdecken:

`httpRequestHandler`. Neben der Authentifizierung und Autorisierung von Benutzern unter Nutzung eines externen Verzeichnisdienstes (*Directory Service*, wobei „extern“ lediglich bedeutet, dass der Verzeichnisdienst bereits existiert und nicht unter der Ägide des hier entwickelten Systems steht), verarbeitet der *Request Handler* die Anfragen des *Webserver*s.

`taskHandler`. Es gibt zwei verschiedene Quellen, von denen *Task*s vom *taskHandler* entgegen genommen werden können. Zum einen können Nutzer Anfragen stellen, die vom *httpRequestHandler* weitergereicht werden. Zum anderen können geplante oder regelmäßige Aufgaben, die durch einen *Scheduler* zur Ausführung gebracht werden, eintreffen. Regelmäßig ablaufende *Task*s können z. B. das automatische Kontrollieren von Praktikumsaufgaben nach Ablauf der Bearbeitungsfrist für ein Aufgabenblatt sein.

Neben den verschiedenen Initiatoren können ebenfalls zwei verschiedene Arten von *Tasks* unterschieden werden: 1.) Die Überprüfung von Praktikumsaufgaben, 2.) die Kommunikation mit der virtuellen Infrastruktur unter Nutzung des externen *Virtualization Controllers*.

Alle *Task*s werden mit ggf. notwendigen Zusatzinformationen aus der Datenbank angereichert und an eine *Task-Queue* weitergegeben. Dort werden sie von *Worker*n abgearbeitet. Der *taskHandler* überwacht den Status der *Task*s und gibt diesen regelmäßig an den *httpRequestHandler* weiter, um ihn im *User Interface* anzeigen zu können. Nachdem ein *Worker* seinen *Task* abgearbeitet hat, wird das Ergebnis in ein *Result-Backend* geschrieben und kann vom *taskHandler* gelesen und verarbeitet werden.

Scheduler. Um z. B. das automatische Kontrollieren von Praktikumsaufgaben nach Ablauf der Bearbeitungsfrist für ein Aufgabenblatt zu realisieren, kommt ein *Scheduler* zum Einsatz. In regelmäßigen Abständen werden die in der Datenbank erfassten Zeitpläne überprüft und alle ggf. anstehenden Aufgaben als *Task*s zur Ausführung an den *taskHandler* weitergegeben.

Task-Queue und Result-Backend. Die *Task-Queue* dient als Warteschlange für *Task*s, die noch von keinem *Worker* verarbeitet werden. Außerdem werden über die *Task-Queue* Informationen über den Status zwischen *Worker* und *taskHandler* ausgetauscht. Nachdem ein *Worker* seinen *Task* abgeschlossen hat, wird das Ergebnis in ein *Result-Backend* eingetragen, aus dem es vom *taskHandler* ausgelesen und weiterverarbeitet kann.

Worker und Testroutinen. Die *Worker* sind eigenständige Prozesse, die die *Task*s aus der *Task-Queue* abarbeiten. Sie führen die eigentlichen Testroutinen aus. Eine Testroutine

ist eine ausführbare Datei, die lediglich mit ihrem Pfad als Referenz in der Datenbank gespeichert ist und eigenständig (ohne weitere Abhängigkeiten) ausgeführt werden kann.

Bevor eine Testroutine ausgeführt wird, werden die zu testenden virtuellen Maschinen durch den *taskHandler* vorbereitet. Dies beinhaltet, dass den Teilnehmern die Rechte zur Verwaltung der VMs entzogen werden, der aktuelle Zustand der VMs durch Snapshots gesichert und auf jeder VM eine Firewall eingerichtet wird, die sicherstellt, dass nur das Testsystem Zugang während der Testausführung erhält. Nach dieser Vorbereitung kann jeder Test von einem *Worker* ausgeführt werden.

Die Kommunikation mit den Testroutinen erfolgt über wohldefinierte Ein- und Ausgaben. Die Testroutinen erhalten als Eingabeparameter die IP-Adressen der zu testenden VMs. Ein Test muss so entwickelt werden, dass er bestimmte Kommandozeilenparameter (wie z. B. `--eth0-ipv6`) verarbeiten kann. Mit diesen Parametern kann sich der Test (meist per SSH) mit der VM verbinden. Anschließend wird der eigentliche Test durchgeführt. Da die Tests eigenständig sind und auch keine Verbindung zur Datenbank haben, ist der Rückgabewert nicht die erreichte Punktzahl einer Aufgabe, sondern ein entsprechender Prozentsatz der erreichten Punkte, der später mit der für die Aufgabe erreichbaren Punktzahl multipliziert wird. Zusätzliche Ausgaben der Testroutinen beinhalten Hinweise und Fehlermeldungen. Diese werden anschließend vom *taskHandler* an das *User Interface* weitergereicht, um die Teilnehmer bei ihrer Arbeit und Fehlersuche zu unterstützen.

Nach Abschluss aller auszuführenden Testroutinen werden die betroffenen VMs durch Wiedereinspielen der zuvor angelegten Snapshots in ihren ursprünglichen Zustand versetzt und sämtliche Berechtigungen der Teilnehmer werden erneut zugewiesen.

Datenbank (DB). Aus operativen Gründen ist es notwendig, dass die angemeldeten Studenten eines Kurses sowie deren Gruppenzugehörigkeit verwaltet werden können. Außerdem müssen die Aufgaben und Übungsblätter des Praktikums verwaltet werden. Jedes Übungsblatt hat eine Bearbeitungsfrist (*Deadline*) und mehrere Aufgaben, die diesem Übungsblatt zugeordnet werden. Nach Ablauf der Bearbeitungsfrist werden die entsprechenden Testroutinen der jeweiligen Aufgaben ausgeführt und die Bewertung jeder Gruppe in der Datenbank gespeichert. Ein dafür geeignetes und hier eingesetztes Datenmodell ist in [Sc16] zu finden.

4.2 Implementierung

Das entworfene System entspricht im Grundsatz einer Web-Anwendung mit Task-Management. Es bietet sich für die Implementierung an, auf bestehende Bibliotheken und Rahmenwerke zurückzugreifen, um die Grundfunktionalität des Entwurfs zu gewährleisten. Die Implementierung erfolgt in Python auf Basis von *Flask*[8]. *Flask* ist aufgrund seiner Flexibilität und unkomplizierten Entwicklung besonders gut geeignet.

[8] http://flask.pocoo.org/

Der *httpRequestHandler* sowie der *Web-Server* mit seinen *Views* ist in *Flask* realisiert. Das Datenmodell ist durch das SQL-Toolkit *SQLAlchemy*[9] und seinen *Object Relational Mapper* umgesetzt. Für den *taskHandler*, die *Worker* sowie den *Scheduler* wird *Celery*[10] verwendet. *Celery*-Tasks werden über die *Task-Queue* an die *Worker* verteilt, und *Celery* übernimmt die Kommunikation mit der *Task-Queue* und dem *Result-Backend*. *Celery* unterstützt verschiedene *Task-Queues*, *Message-Broker* und *Result-Backends*. *Redis*[11] wird sowohl für die *Task-Queue* als auch für das *Result-Backend* verwendet.

Die Implementierung erfolgt beispielhaft für das Praktikum IT-Sicherheit an der LMU. Dieses Praktikum verwendet als Virtualisierer *VMware ESXi*. Die Kommunikation mit dem entsprechenden *Virtualization Controller* (in diesem Fall also dem *vCenter*) wird durch Powershell-Skripte gelöst. Diese werden unter Nutzung der Python-API von *Ansible*[12] ausgeführt. Die Powershell-Skripte abstrahieren somit die Funktionalität VMs steuern und Snapshots verwalten zu können.

5 Bewertung der Ergebnisse

Zur Bewertung der erzielten Ergebnisse bietet es sich an, a) einen Abgleich mit den abgeleiteten Anforderungen zu führen und b) die (subjektiven) Eindrücke zum produktiven Einsatz des Systems in der Lehre zu betrachten.

Bei einer Überprüfung des Anforderungskatalogs zeigt sich, dass von 29 funktionalen Anforderungen 26 erfüllt werden konnten, 3 nur teilweise. Von den 15 nicht-funktionalen Anforderungen wurden 9 erfüllt, 4 wurden nur teilweise erfüllt. 2 nicht-funktionale Anforderungen konnten gar nicht erfüllt werden. Aus Platzgründen werden nachfolgend lediglich die beiden nicht erfüllten Anforderungen kurz beleuchtet.

Wie aus der Systemarchitektur ersichtlich (vgl. Abb. 2), setzt der Entwurf einen existierenden und gepflegten Verzeichnisdienst voraus. Im konkreten Fall der hier angeführten Implementierung existiert zwar ein solcher Verzeichnisdienst in Form eines Verwaltungssystems für Lehrveranstaltungen, jedoch ist keine Schnittstelle für den automatisierten Datenaustausch vorhanden. Als notwendiges Übel muss also ein Datenexport des Verzeichnisdienstes manuell importiert werden, was inhärent zu einer doppelten Datenhaltung und damit potentiell zu Inkonsistenzen führt. Zwar ist die Nichterfüllung der Konsistenzanforderung nicht durch das System bedingt, aber in der vorliegenden Implementierung unumgänglich.

Weiter konnte die Anforderung, falsch-positive und falsch-negative Ergebnisse der Testroutinen zu vermeiden, nicht erfüllt werden. Dies liegt vor allem daran, dass die Ergebnisse einer Testroutine von der Aufgabenstellung und der konkreten Implementierung der Testroutinen

[9] http://www.sqlalchemy.org/
[10] http://www.celeryproject.org/
[11] http://redis.io
[12] https://www.ansible.com/

abhängig sind. Ein möglicher Lösungsansatz, der auch umgesetzt ist, versucht dieses Problem durch Kontrollgruppen zu verringern. Es existiert je eine Kontrollgruppe mit der korrekten Lösung und eine Kontrollgruppe ohne Lösung, so dass zumindest grobe Fehler in den Testroutinen erkannt werden können. Eine vollumfängliche Erfüllung der Anforderung scheint nur unter Anwendung formaler Methoden machbar, was bisher nicht geschehen ist.

Die neue Arbeitsumgebung und die neuartige Art und Weise, ein Praktikum anzubieten, wurde von den Studenten gut angenommen und hat für eine gute Grundlage für Fragen im Tutorium oder per E-Mail gesorgt. Die Korrekturzeit der Aufgaben wurde durch die automatische Korrektur auf ein Minimum reduziert. Durch die Automatismen und die Ausprägung der Testroutinen sind zudem zusätzliche (Verständnis-) Fragen eingegangen, da die Lösungen nicht mehr nur den eigenen Ansprüchen der Studenten, sondern auch den Testroutinen genügen müssen.

Kritik hingegen gab es vorwiegend bei der Interpretation der Statusmeldungen von Testroutinen, wenn eine Aufgabe nicht mit voller Punktzahl bewertet worden ist. Hier ist die größte Herausforderung, dass auf der einen Seite vielsagende Rückmeldungen wünschenswert sind, aus der Rückmeldung aber auf der anderen Seite nicht direkt auf die richtige Lösung der Aufgabe geschlossen werden können soll.

6 Zusammenfassung & Ausblick

Im Rahmen dieser Arbeit ist ein Automatismus entstanden, der die Korrektur praktischer Übungsaufgaben im Rahmen eines universitären Praktikums übernimmt. Als Ausgangsbasis der Arbeit dienen bereits virtualisierte e-Infrastrukturen zur Ausbildung im Bereich vernetzter Systeme. Durch die Virtualisierung bedingt skalieren die eingesetzten Infrastrukturen bereits sehr gut, jedoch entwickelt sich schnell ein Flaschenhals durch die noch immer notwendige (persönliche) Betreuung von Teilnehmern sowie die Korrektur bearbeiteter Aufgaben. Eine Anwendungsfall-getriebene Anforderungsanalyse auf Basis existierender Praktika bringt einen Katalog an Anforderungen hervor, der systematisch zu einer Systemarchitektur führt, die im Nachgang implementiert und produktiv in Betrieb genommen wurde. Die Erfahrungen im operativen Einsatz sind sehr gut – sowohl objektiv durch einen hohen Grad der Erfüllung existierender Anforderungen, als auch subjektiv im Sinne positiver Rückmeldung der Praktikumsteilnehmer.

Neben Erweiterungen der Implementierung, wie z. B. die Erweiterung des derzeitigen Aufgabenpools oder der Anbindung an existierende Verzeichnisdienste und Dienste zur Veranstaltungsverwaltung, steht die Weiterentwicklung zu einer Art „Praktikum as a Service“ an. Studenten sollen den Startzeitpunkt ihres Praktikums sowie die Bearbeitungsgeschwindigkeit vollkommen frei wählen können. Dazu bedarf es einiger Modifikationen, insbesondere des zugrundeliegenden Datenmodells sowie der prozessualen Abläufe.

Literatur

[Bl04] Blumenstein, M.; Green, S.; Nguyen, A.; Muthukkumarasamy, V.: GAME: a Generic Automated Marking Environment for programming assessment. In: International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. Bd. 1, 212–216 Vol.1, Apr. 2004.

[BR13] Baumstark, L.; Rudolph, E.: Automated Online Grading for Virtual Machine-based Systems Administration Courses. In: Proceeding of the 44th ACM Technical Symposium on Computer Science Education. SIGCSE '13, ACM, Denver, Colorado, USA, S. 477–482, 2013.

[DGK16] Danciu, V.; Guggemos, T.; Kranzlmüller, D.: Schichtung virtueller Maschinen zu Labor- und Lehrinfrastruktur. In: 9. DFN Forum Kommunikationstechnologien. GI–Edition Lecture Notes in Informatics, Rostock, Deutschland, Juni 2016.

[EP08] Edwards, S.; Perez-Quinones, M.: Web-CAT: Automatically Grading Programming Assignments. In: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education. ITiCSE '08, ACM, Madrid, Spain, S. 328–328, 2008.

[HPK11] Hull, M.; Powell, D.; Klein, E.: Infandango: Automated Grading for Student Programming. In: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education. ITiCSE '11, ACM, Darmstadt, Germany, S. 330–330, 2011.

[KLC01] Kurnia, A.; Lim, A.; Cheang, B.: Online Judge. Comput. Educ. 36/4, Mai 2001.

[Li05] Lindinger, T.: Machbarkeitsanalyse zur Virtualisierung des IT–Sicherheit Praktikums, Techn. Ber., LMU, Okt. 2005.

[Li06] Lindinger, T.: Virtualisierung einer Praktikumsinfrastruktur zur Ausbildung im Bereich Sicherheit vernetzter Systeme, Diplomarbeit, LMU, Mai 2006.

[LRg08] Lindinger, T.; Reiser, H.; gentschen Felde, N.: Virtualizing an IT–Lab for Higher Education Teaching. In: Tagungsband zum 1. GI/ITG KuVS Fachgespräch „Virtualisierung". Paderborn, Deutschland, S. 97–104, Feb. 2008.

[Sc11] Schmidt, B.: Konzeptionelle und praktische Erweiterung des Praktikums IT–Sicherheit rund um das Thema IPv6, Techn. Ber., LMU, Nov. 2011.

[Sc16] Schmidt, J.: Vollautomatisiertes Prüfungssystem am Beispiel des Praktikums IT-Sicherheit, Masterarbeit, LMU München, Aug. 2016.

[SHS15] Schlarb, M.; Hundt, C.; Schmidt, B.: SAUCE: A Web-Based Automated Assessment Tool for Teaching Parallel Programming. In: Euro-Par 2015: Parallel Processing Workshops. Springer, Cham, S. 54–65, Aug. 2015.