The Microservice Dungeon: Realitätsnahe Lehre komplexer Softwarearchitekturen

Philipp Felix Schmeier[1], Stefan Bente[2]

Abstract: Microservices bieten sich als Lösung für große, lose gekoppelte Softwarelandschaften an, die durch weitgehend autonome Teams entwickelt werden. Dies setzt einen geeignetem fachlichen Schnitt, asynchrone Kommunikation und eine Container-orchestrierte Infrastruktur voraus - alles erhebliche Komplexitätstreiber. In der Hochschullehre hat man kaum die Chance, diese Komplexität den Studierenden anschaulich und praxisnah zu vermitteln, da hier aus didaktischen Gründen häufig eher kleinere Software-Einheiten betrachtet werden. Das Verbundprojekt »The Microservice Dungeon« mit 59 Studierenden aus vier verkoppelten Lehrveranstaltungen in drei Informatik-Studiengängen versucht diesen Widerspruch aufzulösen. Die Studierenden konzipieren eine große Microservice-Landschaft und implementieren darin selbst Services, die in einer Spielwelt Roboterschwärme steuern. So entsteht eine Lernplattform für das Software Engineering, die nunmehr in der dritten Iteration weiterentwickelt wird. Anhand detaillierter Feedback-Befragungen und einer Analyse von ca. 3500 Posts im Messenger des Projekts konnte gezeigt werden, dass dieser Ansatz die Studierenden erfolgreich zu einer aktiven Beschäftigung mit Softwarethemen wie REST-Calls, Eventing und DevOps hinführt.

Keywords: Microservice; Lernplattform; Software Engineering; Eventing; REST; lose Kopplung; DevOps; Infrastruktur; Didaktik

1 Einführung

Bei der Vermittlung von Softwarearchitektur-Kompetenzen hat die projektorientierte Hochschullehre mit einem inhärenten Widerspruch zu kämpfen. Einerseits muss die Aufgabenstellung und die entstehende Code-Basis klein genug sein, damit Studierende im Rahmen einer typischen Veranstaltungsgröße - zwischen 5 und 10 ECTS - in einem Semester das Lernziel erreichen können. Andererseits werden sowohl Vorzüge wie auch Mehraufwand einer modernen, cloud-kompatiblen Architektur (mit auf Domain-Driven Design [Ev04] beruhenden Microservices in einer Container-orchestrierten Infrastruktur) erst ab einer gewissen Systemgröße deutlich.

[1] TH Köln, CIDE - Cologne Institute for Digital Ecosystems, Steinermüllerallee 1, 51643 Gummersbach, philipp.schmeier@gmail.com
[2] s.o., stefan.bente@th-koeln.de

Wenn die Studierenden lernen sollen, hochperformante und skalierbare Systeme wie bei Netflix oder Amazon [Hi21] zu bauen, dann brauchen sie ein spezielles Projektformat mit hinreichender Größe. In der Literatur gibt es hierzu nur wenige dokumentierte Beispiele[3].

Aus diesem Grund wurde durch die Autoren ein großes Verbundprojekt mit insgesamt 59 Studierenden in vier integrierten Lehrveranstaltungen konzipiert, das eine praxisnahe Microservice-Architektur umsetzt. Ein wesentliches Ziel von Microservices ist es, Teams ein weitgehend autarkes Arbeiten ohne dauernde Abstimmungen über die Teamgrenzen hinweg zu ermöglichen. Das Verbundprojekt war mit der großen Menge an teilnehmenden Studierenden bewusst darauf angelegt, ein im Studium sonst nicht übliches Maß an Komplexität zu schaffen. Die Studierenden sollten den »Schmerz« dauerhafter Abstimmungsprozesse erfahren. Daraus, so die Intention der Lehrveranstaltung, sollte sich die Motivation speisen, in Tests, Dokumentation und geeignete Architekturen (Redundanz, asynchroner statt synchroner Nachrichtenaustausch, etc.) zu investieren, um Abstimmungsbedarfe zu reduzieren und ungestörter arbeiten zu können. Mit anderen Worten: Das Projekt hatte den Anspruch, reale Berufsbedingungen in puncto Komplexität zu simulieren.

Gleichzeitig sollte das Projekt die Studierenden ermuntern, fachliche und technische Schwierigkeiten zu diskutieren und im Konsens zu Lösungen zu kommen. Im Sinne echter »self-organized teams« wurde seitens der Betreuer bewusst darauf verzichtet, zu viele Vorgaben im Detail zu machen. Dadurch sollte den Studierenden Raum für die eigene Lösungsfindung geschaffen werden. Darüber hinaus war es das Ziel, alle Teilnehmenden möglichst gleichermaßen zu beteiligen (keine »Trittbrettfahrer«). Kompetenzen sollten mindestens auf Stufe 3 (*Applying*), besser 5 (*Evaluating*) der Revised Bloom's Taxonomy [AK01] erworben werden. Das Verbundprojekt wurde im Rahmen einer Masterarbeit [Sc22] empirisch begleitet, auf deren Ergebnissen das vorliegende Paper zu großen Teilen beruht.

2 Aufbau und Phasen des Verbundprojekts

Unter dem Titel »The Microservice Dungeon« wurden für das Projekt im Wintersemester 21/22 vier Lehrveranstaltungen aus drei Informatik-Studiengängen zusammengekoppelt. Die Entwicklung eines Spiels wurde gewählt, um den Studierenden eine interessante und ihrer Lebenswirklichkeit nahestehenden Domäne zu bieten. Dies half erkennbar bei Diskussionen über Fachlichkeit - so gut wie alle Beteiligten waren »Gamer« und konnten Erfahrung zu bestimmten Mustern und Regelsystemen beisteuern [Sc22].

Das Verbundprojekt befindet sich zum Zeitpunkt, an dem dieses Paper geschrieben wurde, bereits in der dritten Iteration. Im Folgenden wird aber nur die erste Projektiteration

[3] Eine erwähnenswerte Ausnahme ist das »Laboratory of Complex Computational Systems« an der University of São Paulo (USP), bei dem eine Microservice-Landschaft mit einer großen Gruppe von Studierenden durchgeführt wurde (4-mal als zweiwöchiger Kurs mit insgesamt 65 Studierenden und einmalig einsemestrig mit 18 Studierenden) [Co20]. Das aus dem Projekt entstandene Paper fokussiert allerdings eher auf erreichte »Hard Skills« und weniger auf eine detaillierte Analyse von Projektformat und Aufgabenstellung aus didaktischer Sicht.

beschrieben und analysiert, da es sich bei dieser um die zahlenmäßig größte Version des »Microservice Dungeon« handelt und sie damit die besten Möglichkeiten für Analysen liefert. Die nachfolgenden Iterationen werden in Kap. 5 kurz umrissen.

Das Projekt wurde vom 13.09.2021 bis zum 28.01.2022 durchgeführt. Die verschiedene Module hatten teilweise unterschiedlichen Aufgaben und Rollenzuweisungen. Die Teilnehmenden des einzigen Master-Moduls »Domain Driven Design of large Software Systems« übernahmen beispielsweise die Rolle von Architekt:innen und Product Ownern. Abb. 1 zeigt eine Übersicht über alle Teilnehmenden und deren Rollen.

Kürzel	Studiengang	Module	ECTS	Personenzahl
DDD	Digital Sciences (Master)	Domain Driven Design of large Software Systems	6	10
MSA	Code and Context (Bachelor)	Microservices Architecture	3	29
PL	Code and Context (Bachelor)	Project Launch	12	5
IP	Informatik (Bachelor)	Informatikprojekt	10	15

Rollen	DDD	MSA	PL	IP
Architekt:in	x			
Product Owner	x			
Gameplay Designer			x	
Software Developer (Service)	(x)		x	x
Software Developer (Player)	(x)	x	x	x

Abb. 1: Übersicht der Teilnehmenden

Das Projekt begann als Greenfield-Projekt und ermöglichte dadurch den Studierenden große Freiräume bei der Gestaltung, von der Systemlandschaft bis hin zu den verwendeten Technologien. Die Studierenden hatten weitgehende Technologiefreiheit, was auch als Vorteil von Microservice-Systemlandschaften gilt [Ne15]. Dies führte zu einer gewissen Technologie-Vielfalt in den fünf Core-Services.

Die Lehrenden beschränkten die eigene Einflussnahme weitgehend und gaben nur Rahmen und Ziele vor. Im Vordergrund stand die Moderation von Veranstaltungen und Statustreffen. Die genauen Inhalte und Regeln des entstehenden Spiels wurde den Studierenden, genauer der Subgruppe der »Gameplay Designer/Dungeon Master« (Gruppe PL in Abb. 1) überlassen, die über das gesamte Projekt hinweg als Ansprechpartner und Expert:innen für die Spielregeln agierten. Der Ablauf des Projekts wurde in verschiedene Phasen unterteilt, wie in nachfolgender Abbildung dargestellt.

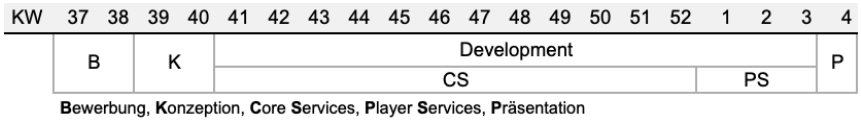

Abb. 2: Projektphasen nach Kalenderwoche (2021/22)

Bei drei der vier teilnehmenden Module handelte es sich um Wahlpflicht-Module, nur MSA ist ein Pflichtmodul. Bei Modul IP wurde eine Teilnehmer-Obergrenze von 15 festgelegt. Die Bewerber:innen hatten die Möglichkeit, mittels Bewerbungsvideos ihre Motivation für die Projektthematik darzustellen, was als Auswahlkriterium diente. Daher kann von einem gewissen intrinsischem Interesse der teilnehmenden Studierenden an dem Thema ausgegangen werden.

2.1 Konzeption

Die Gameplay Designer (PL) hatten vor Beginn Zeit, die Spielregeln festzulegen, die im Semester umgesetzt werden sollten. Bis auf wenige Eingriffe der Lehrenden (etwa um den Umfang der Spielmechaniken zu reduzieren) stand es den Designern frei, welche Form die Regeln annehmen sollten. Die Phase wurde mit einer Präsentation des geplanten Gameplays für alle[4] Projektteilnehmenden abgeschlossen. Ebenfalls in diesem Rahmen wurde ein *Event Storming* durchgeführt, dessen Ergebnisse in Abb. 3 dargestellt sind.

Abb. 3: Übersicht der Ergebnisse des Event Storming Workshops

Die von Alberto Brandolini entwickelt Methodik des Event Storming ermöglicht die explorative Erforschung einer komplexen Fachdomäne. Ursprünglich mit dem Hintergrund der Softwareentwicklung entworfen, bietet dieses Vorgehen Einsatzmöglichkeiten bis hin zu Organisationsentwicklung [Br21]. Für das Projekt diente das zweitägige Event Storming mehreren Zwecken. Einerseits wurde die Fachlichkeit (Spielablauf und Regeln) von den Studierenden durchdrungen, hinterfragt und verinnerlicht. Andererseits konnten aus den Ergebnissen heraus die Domänengrenzen der Core-Services identifiziert werden.

Durch das interaktive Format konnte der Herausforderung durch die große Zahl und Diversität der Teilnehmenden begegnet werden. Besonders angesichts der ansonsten aufgrund der Pandemiesituation häufigen Remoteveranstaltungen wurde das Präsenzformat als lebendig und inspirierend wahrgenommen, und bot allen Beteiligten die Möglichkeit, sich persönlich kennenzulernen. Besonders aus didaktischer Sicht kann vermutet werden, dass dieses Treffen zu einem frühen Projektzeitpunkt zu der positiven, lebendigen Stimmung in den Projektdiskussionen (siehe Kap. 4) beigetragen hat.

Die konzipierten und durch das Event Storming »ausgereiften« Spielregeln wurden für alle Teilnehmenden öffentlich einsehbar über eine Website dokumentiert [Mia]. Die Spiellandschaft stellt eine zweidimensionale Karte dar, die den teilnehmenden Playern unbekannt ist. Jedes Feld der Karte spiegelt einen Planeten wider, der verschiedene Ressourcen enthalten kann. Ein Player ist kein Mensch, sondern ein autonom agierender und in der Infrastruktur deployter Service. Hinter einem Player steht ein Studierenteam, das ihn implementiert.

Teilnehmende Player erhalten ein Startguthaben und können sich damit erste Roboter kaufen, die auf definierten Stellen auf der Karte starten. Die Player kennen zu Beginn nur die Planeten, auf denen ihr Roboter startet. Ziel der Player ist es, durch geschicktes Erkunden,

[4] MSA war nicht Teil der Einführungsveranstaltung, da MSA in einer Blockveranstaltung gelehrt wird. Die MSA-Studierenden stießen daher erst gegen Ende des Projektes (Playerentwicklung) dazu.

Abbauen und Verkaufen von Ressourcen sowie durch Kämpfe einen Roboterschwarm aufzubauen und Punkte zu sammeln, die für jede Interaktion im Spiel verteilt werden.

2.2 Development

Die Gestaltung der Spielregeln, also der Karte, der Überwachung der Roboterbewegung, der Anmeldung zum Spiel und die Verwaltung der Guthaben der Spieler war die erste Aufgabe für die Studierenden. Damit entstanden fünf »Core-Services«. Die Entwicklung der eigenen Robotersteuerung und Strategien wurde zeitlich und thematisch davon abgegrenzt, dies sind die »Player-Services«. Diese Abgrenzung hatte im Wesentlichen drei Gründe:

1. Alle Teilnehmenden, die sowohl Core- wie auch Player-Services entwickeln, lernten durch die Wiederholung, essenziellen Aspekte noch besser zu verinnerlichen.
2. Alle Teilnehmenden mussten gegen vorgegebene REST- und Event-Spezifikation entwickeln (für die Player-Services), wie auch solche Spezifikationen selbst entwickeln (Core-Services).
3. Teilnehmende aus dem MSA-Modul, die aus Lehrplan-Gründen erst spät hinzukommen konnten, konnten zumindest noch in der Player-Entwicklung dabei sein.

Die Entwicklung der Systemlandschaft mit den Core-Services stellte die zeitlich längste Phase des Projektes dar. Die im Serviceschnitt im Event Storming entstandenen Services wurden Studierendengruppen von 2 bis 5 Softwareentwickler:innen zugeordnet. Jedem so entstandenen Development Team aus Bachelor-Studierenden wurden zwei Master-Studierende als Product Owner und Architekt:in zugeordnet. Über einen Zeitraum von 12 Wochen entstand in einer durch agiles Vorgehen inspirierten Projektstruktur die Servicelandschaft. Die entstandenen Services können in GitHub [Mic] (1. Iteration) und GitLab [Mib] (aktueller Stand) eingesehen werden. An Scrum orientiert wurden 2-wöchentliche Sprint-Review- und -Planning-Meetings abgehalten, die durch die Product Owner geleitet wurden. Lehrende begleiteten alle Meetings als Beobachter und Antwortgeber für Fragen zu Prozessen oder organisatorischen Themen[5].

Nach Fertigstellung der Core-Services wurde die Playerentwicklung angestoßen, die über zwei Wochen im Januar stattfand. Dieser Prozess konnte aufgrund zeitlicher Verzögerungen und Entwicklung der Pandemie nicht wie geplant als durchgängiger einwöchiger Hackathon am Campus stattfinden. Der Abschluss der Playerentwicklung gipfelte in einem *Codefight*, in welchem die Player-Services gegeneinander antraten.

Das Projekt wurde mit Kurzpräsentationen der Studierenden abgeschlossen. Aufgabe war die Reflektion des eigenen Handelns im Projekt. Die Themen konnten dabei von den

[5] Die agile Arbeitsweise war kein definiertes Lernziel der Veranstaltung, um die Veranstaltung nicht zu überfrachten. Die gewählte Form hat sich aber in zahlreichen Lehrforschungsprojekten der Autoren bewährt.

Studierenden frei gewählt werden, sofern es einen Bezug zum Projekt gab. Von Ideen und Konzepten für eine weitere Projektiteration über technische Aspekte bis hin zu Darstellung der eigenen Arbeitsweise im Team war das Ergebnis breit gefächert. Dies war einer von mehreren möglichen Feedbackkanälen für die Studierenden. Weitere werden in den folgenden Kapiteln näher erläutert und analysiert.

3 Feedback der Studierenden

Wie bei vielen realen Software-Projekten traten auch im »Microservice Dungeon« Herausforderungen aller Art auf, von Schwierigkeiten im eigenen Team über unterschiedliche Code-Qualität bis hin zur projektweiten Kommunikation. Um den Projekterfolg und die Vermittlung der Learning Outcomes zu prüfen sowie Hinweise für das Format zu sammeln, wurden verschiedene Feedbackschleifen bereitgestellt, die es den Studierenden ermöglichten, Vorschläge, Kritik und Wünsche zu äußern.

Die erste und dauerhafte Möglichkeit bestand im direkten Feedback, entweder nur an die Lehrenden oder an das gesamte Projektteam. Für den Großteil der Kommunikation wurde die Plattform Discord[6] verwendet, die ein Instant-Messaging sowohl als Privatnachricht als auch über Gruppenkanäle ermöglicht. Eine dedizierte Auswertung der über Discord stattgefundenen Kommunikation ist in Kap. 4 zu finden. Zusätzlich dazu wurden zwei Umfragen durchgeführt, eine Zwischenumfrage zur Mitte des Projektes sowie die Abschlussumfrage unmittelbar nach Abschluss der Projektarbeiten. Zuletzt konnten die Studierenden noch durch die Abschlusspräsentation mit ihrer weitgehend freien Themenwahl Vorschläge einbringen und Herausforderungen beleuchten. Ergänzend lieferte ein Teil der Masterstudierenden eine ausführliche individuelle Reflektion des Projekts als Wahlleistung.

Das Feedback durch die Abschlussumfrage fiel sehr positiv aus, insbesondere mit Blick auf erlernte Hard-Skills. So gaben über zwei Drittel der Teilnehmenden an, ein wenig oder sogar viel besser im Programmieren geworden zu sein [Sc22]. Dies ist besonders bemerkenswert, da seitens der Lehrenden aus Zeitgründen kaum Hilfestellung zu konkreten Programmieraufgaben gegeben werden konnte. Weiterhin ist die Verständlichkeit von Microservices im generellen sowie die Nachvollziehbarkeit von Vorteilen und Bedeutung von Microservices fast durchgängig bei allen Teilnehmenden stark gestiegen, wie in Abb. 4 zu sehen[7].

[6] Discord ist ein Onlinedienst für Kommunikation und hat ihren Ursprung im Bereich des Gamings und ermöglicht die Kommunikation via Textkanälen, Sprachkonferenzen (inkl. Screenshare und Webcam) und Instant Messaging.

[7] Das Feedback wird aus Platzgründen hier zusammenfassend dargestellt. Für eine weiter ausdifferenzierente Analyse sei auf [Sc22] verwiesen.

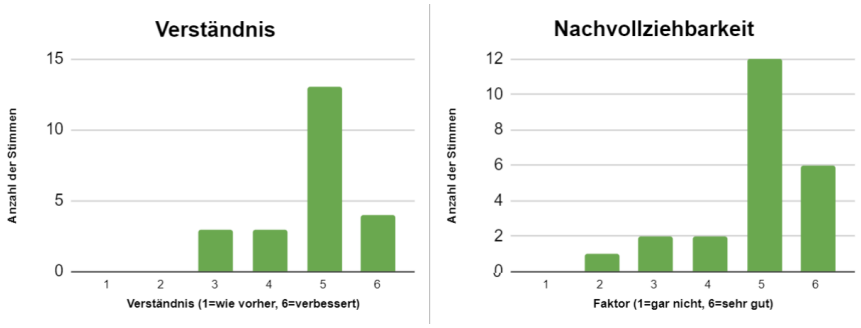

Abb. 4: Verständnis und Nachvollziehbarkeit von Microservices und ihren Vorteilen [Sc22]

Die Erwartungen an das im Projekt Gelernte wurden für einen großen Teil der Studierenden deutlich übertroffen, wie Abb. 5 (links) zeigt. Die meisten Teilnehmenden gaben an, mehr (Stufe 4) oder deutlich mehr (Stufe 5) gelernt zu haben als erwartet. Neu erworbene Kompetenzen konnten dabei auch wirksam in der Implementierung der Core-Services genutzt werden. Dadurch war das eigene Qualitätsempfinden bei den Core-Services allgemein sehr hoch (Abb. 5, rechts).

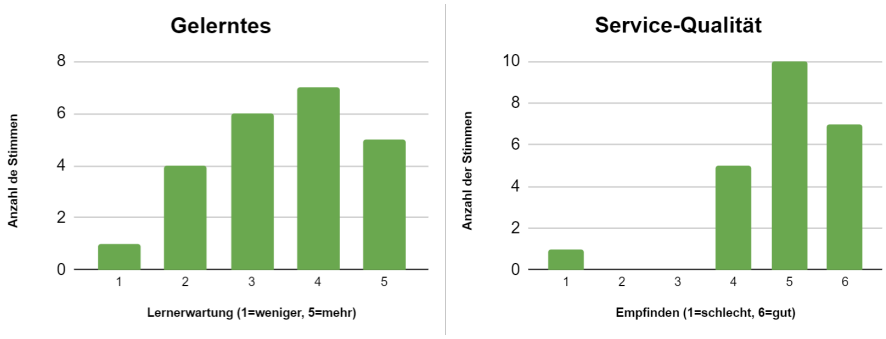

Abb. 5: Übersicht der studentischen Selbsteinschätzung bzgl. Lernerwartung und der produzierten Code-Qualität [Sc22]

Trotz verschiedener Probleme und Herausforderungen in Abstimmung, Arbeitsverteilung und anderen Aspekten wurde das Projekt allgemein als großer Erfolg wahrgenommen. Dies gilt aus Sicht der Studierenden sowohl für die Freude an der Projektarbeit als auch für das Endergebnis, wie Abb. 6 zeigt. Die Spaßtreiber sind dabei breit aufgestellt, die Studierenden nannten beispielsweise das »Event Storming«, »die fachlichen Diskussionen«, das »Entwickeln des eigenen Microservices« oder den Moment, »wenn die Services angefangen haben miteinander zu interagieren« als Höhepunkte der Projekts [Sc22].

Besonders die Zufriedenheit mit dem Projektergebnis ist hervorzuheben. Sie ist um so bemerkenswerter, da der gegen Ende des Projektes geplante Hackathon ausfallen musste und die entwickelten Player-Services angesichts der Projektverzögerungen nicht so weit ausgereift waren, um wirklich spannende Codefights zu ermöglichen. So wurde auch die Projektgröße

als etwas überdimensioniert wahrgenommen. Obwohl also das nominelle, zu Beginn ausgegebene Projektziel »Wir machen einen großen Codefight« nur sehr eingeschränkt erreicht werden konnten, nahmen die Studierenden das Projekt trotzdem als Erfolg wahr. Aus Lehrendensicht war das Betreuungsaufwand durchaus hoch, aber nicht höher als die Summe der Aufwände bei einer »normalen« Durchführung als isolierte Einzelveranstaltungen.

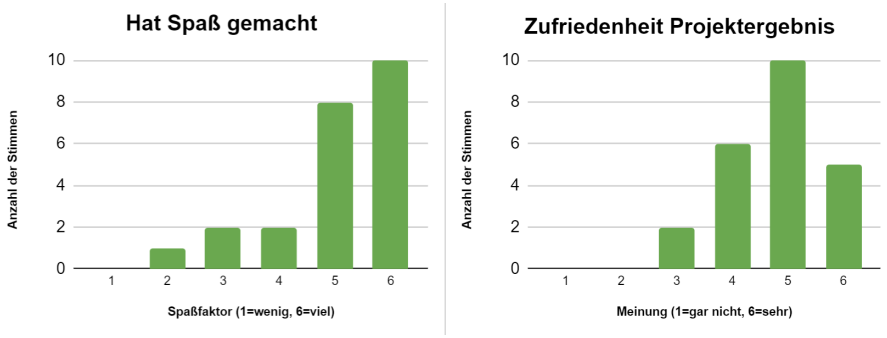

Abb. 6: Spaßfaktor und Zufriedenheit der Studierenden mit dem Projektergebnis [Sc22]

Die Umfrageergebnisse als Ganzes bestätigen den Lernerfolg der Studierenden, besonders im Bereich der Hard-Skills. Fortschritte und Lernerfolge in Reflektionsfähigkeit, Kommunikationsverhalten und Architektur-Knowhow war in den Umfragen kaum greifbar zu gestalten und lässt sich daher nicht in belastbaren Zahlen darstellen. Sie waren in den Studierenden-Präsentationen zur Abschlussreflektion dennoch ablesbar, die ein durchgehend hohes Niveau hatten.

4 Kommunikationsanalyse

Im Projekt wurde Discord als Messenger-Dienst verwendet. Dafür wurden acht dedizierte Kanäle angelegt (je einer pro Core-Service, einer für DevOps, einer für Player-Entwicklung und einer für allgemeine Fragen). Da Discord kein offizielles Tool der Hochschule ist, wurde die Kommunikation via Discord den Studierenden nicht vorgeschrieben. Es wurde dennoch rege genutzt, da davon ausgegangen werden kann, dass Discord durch die Nähe zur Gaming-Community einem hohen Prozentsatz der Teilnehmenden vertraut ist. Viele der Studierenden kennen es auch aus ihrem 3. und 4. Bachelor-Semester, wo es im Modul Softwaretechnik erfolgreich als Diskussionsforum eingesetzt wird [BIK22].

Insgesamt wurden in der Laufzeit des Projekts 3485 Posts abgesetzt, das entspricht einem Mittel von ca. 25 am Tag. In der Spitze war es die zehnfache Zahl, 252 (wenig überraschend: am letzten Tag vor dem Codefight). Diese Posts wurden für dieses Paper aus Discord exportiert, thematisch codiert und ausgewertet, um besseren Einblick in die diskutierten Themen und die Beteiligung der Studierenden zu erhalten.

4.1 Thematische Verteilung

Die inhaltliche Lebendigkeit und Vielfalt der Diskussion lässt sich an Abb. 7 ablesen. Auf der X-Achse sind jeweils die Projektwochen aufgetragen, auf der Y-Achse die Anzahl Posts pro Woche in einer bestimmten thematischen Ausprägung. Discord wurde keineswegs nur für organisatorische Themen wie etwa Terminabsprachen genutzt (links, gestrichelte Linie). Dies machte nur zu Beginn, in einer "Findungsphase", die Mehrzahl der Posts aus. Gegen Ende, als es auf die Reflektions-Präsentation mit zugehöriger Benotung zuging, nahm die Anzahl der nicht-inhaltlichen Posts wieder leicht zu. Es überwogen aber ab etwa der Mitte des Projekts die inhaltlichen Themen.

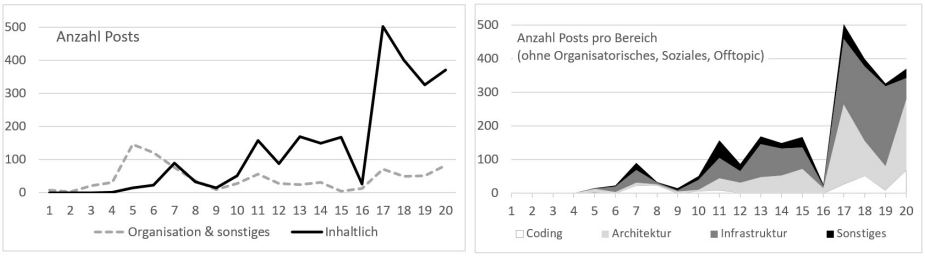

Abb. 7: Verlauf der Posts nach thematischen Bereichen (»Knick« bei Woche 16 = Weihnachtswoche)

Dabei dominierten überraschenderweise nicht unbedingt die reinen »Coding«-Themen, die das Programmieren im Kleinen (mit den damit einhergehenden Verständnisfragen) abdecken (rechts, unterstes weißes Band). Hier scheinen die Studierenden die eingesetzten Technologien gut zu beherrschen. Stattdessen wurde außerordentlich viel über Architektur (hellgrau) und Infrastruktur (dunkelgrau) diskutiert - ein erfreuliches Zeichen, dass die Studierenden offensichtlich eines der Projektziele aktiv angenommen hatten, nämlich sich mit der Tauglichkeit von Architektur- und Hostingansätzen auseinanderzusetzen.

Schaut man sich die Schwerpunkte bei Architektur-Diskussionen genauer an (Abb. 8 links), so werden die Themen »Eventing« (Struktur und Payload der auf Kafka-Topics veröffentlichten Events) und »REST-API« (Aufbau und Implementierung der REST-Endpunkte) mit annähernd gleichbleibender Intensität von der Mitte des Projekts an bis zum Ende diskutiert. Die Intensität beim Thema »Choreographie« (dunkelgraues Band: Welche Event-Konsumierungen und synchronen Aufrufe müssen in welcher Reihenfolge erfolgen, um gewisse Transaktionen zu ermöglichen?) nimmt hingegen zum Ende des Projekts hin zu. Dies korreliert mit der Phase der Player-Entwicklung (Phase PS in Abb. 2), wo sich die Player-Services der Fertigstellung nähern und sich Integrationsprobleme beim komplexen Synchron-Asynchron-Zusammenspiel zeigen.

Bei den Infrastruktur-Themen (Abb. 8 rechts) nimmt das Thema »CI/CD« (Anbindung des Service an das Deployment) in der Projektmitte den größten Raum ein, wo alle Services in einer ersten »Hello-World«-Ausbaustufe vorlagen. Die Infrastruktur selbst (Aufbau der

Serverlandschaft, Zuteilung von Ressourcen, Monitoring) blieben eher ein Randthema bis ganz zum Schluss, als die Integrationsphase kurz vor dem Codefight die bislang unentdeckten Probleme aufzeigte.

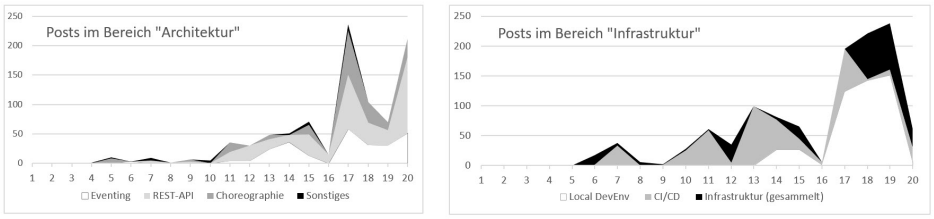

Abb. 8: Verlauf der Posts in den Bereichen »Architektur« und »Infrastruktur«

Das Thema »Local DevEnv« (weißes Band in Abb. 8 rechts) dominiert - auf den ersten Blick überraschend - die Endphase des Projekts. Dies zeigt ein Problem bei der Entwicklung verteilter Systeme auf: Um den eigenen Service lokal testen zu können, muss ein ungleich höheren Aufwand betrieben werden als bei monolithischer Entwicklung (da ja für einen lokal laufenden Player-Service alle Core-Services ansprechbar sein müssen). Dies wurde im Projekt durch eine spezielle, auf Docker-Images und Scripten beruhende lokale Umgebung ermöglicht, in der die fünf Core-Services zur Verfügung gestellt wurden.

Diese Umgebung führte immer wieder zu Nachfragen, weil die Studierenden sich mit einer Menge an komplexer Technologie auseinandersetzen mussten. Das wurde aber erst in dem Moment auf breiter Front relevant, als die Player-Services aus der Unit-Test-Phase in die Integration übergingen. Dies unterstreicht anschaulich, dass das Verbundprojekt seinen Praxisnähe-Anspruch gut einlöst: Die Studierenden stoßen auf dieselben Probleme, von denen auch Microservice-basierte Produktivprojekte aus der »realen Welt« berichten.

4.2 Diskussionsauslöser

Auslöser für Diskussionen waren häufig konzeptionelle Fragen, aber oft auch Verständnisprobleme und Hinweise auf Probleme (Laufzeitfehler, HTTP-Fehlercodes als Returnwerte von REST-Calls, Fehler beim Konsumieren von Events, Inkonsistenzen zwischen Dokumentation und Implementation, etc.). Diese zweite Art von Auslösern wurden in der Codierung der Posts mit einem pauschalen »Problem«-Flag berücksichtigt.

Man erkennt in Abb. 9 links, dass bei den Architekturthemen (schwarze Linien) die Problem-getriggerten Diskussionen erst gegen Ende in der Integrationsphase einsetzen. Zuerst werden Probleme mit der synchronen Kommunikation via REST evident (schwarze gestrichelte Linie) - vermutlich, weil diese aus den Veranstaltungen im 4. Semester besser bekannt ist [Be22]. Die Probleme mit asynchroner Kommunikation (schwarze durchgezogene Linie) treten erst später zutage. Wie oben in Abb. 8 zu erkennen ist, beginnt die konzeptionelle

Diskussion zu diesen Themen deutlich früher, nur die aktive Inanspruchnahme durch Player-Services wird durch die »Fehler-Peaks« markiert.

Auch im Bereich Infrastruktur sind die »Ausschläge« mit hohem Anteil an Problem-Auslösern konsistent mit dem Zeitpunkt, an dem das Verbundprojekt das jeweilige Thema in die Breite brachte. Pipelines für Core-Services und erste, halbfertige Player-Services wurden vom DevOps-Team schon relativ früh im Projekt vorbereitet, daher treten auch die ersten Probleme im Bereich CI/CD (graue gestrichelte Linie) schon früh auf. Probleme in der Infrastruktur kommen erst kurz vor Produktivsetzung bei den Testläufen zum Codefight ans Licht, als das Zusammenspiel aller Komponenten erprobt wurde. Auch hier verhält sich das Verbundprojekt sich »praxisnah« und zeigt ähnliche Muster wie bei Realprojekten.

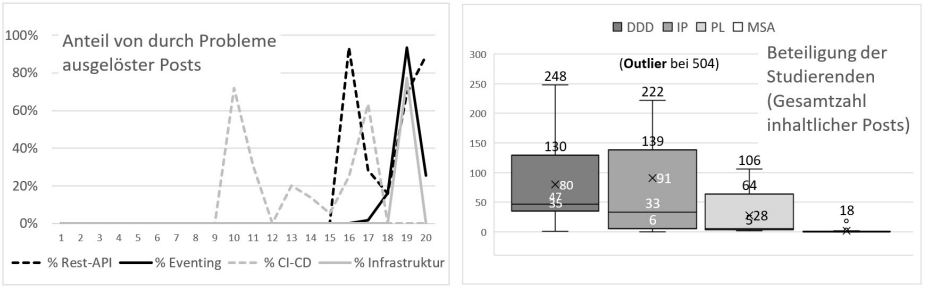

Abb. 9: Anteil der durch Probleme ausgelösten Posts in den Bereichen (links) und Beteiligung der Studierenden (rechts)

4.3 Beteiligung und Diversität

Die bisherige Analyse belegt, dass die Diskussionskultur in dem Verbundprojekt als lebendig, fachlich geprägt und praxisnah angesehen werden kann. Rechts in Abb. 9 sieht man einen Box-Plot der Beteiligung (gesamte Zahl Posts pro Teilnehmendem, aufgeschlüsselt nach Kurs). Sieben Teilnehmende (IP(BA): 4, PL(BA): 1, DDD(MA): 2), steuerten jeweils mehr als 4% der gesamten Menge von 2590 inhaltlichen Posts bei, also jeweils mehr als 100 Posts. Ein Teilnehmer aus dem Informatikprojekt stellte mit 504 inhaltlichen Posts ca. 20% des Gesamtvolumens. Es war immer wieder festzustellen, dass diese *Student Leaders* [AJ22] die allgemeine inhaltliche Diskussion anstießen, bei Problemen halfen und dadurch ein lebendiges Interaktionsklima entstehen ließen.

Die Diskussion fand aber durchaus auch in der Breite der Teilnehmerschaft statt. Der Master-Kurs DDD und das Informatikprojekt IP haben hier ein vergleichbares Profil. Median und oberes/unteres Quartil liegen bei ähnlichen Werten, wobei der Master-Kurs homogener ist. Das aus fünf Teilnehmenden bestehende PL hat ein sehr aktives Mitglied und vier weitere, die sich in Workshops durchaus beteiligten, aber kaum an der Fachdiskussion teilnahmen. Insgesamt haben knapp 2/3 der Teilnehmenden aus DDD, IP und PL (18 von 30) mindestens

30 inhaltliche Posts verfasst. Bei den nur abschnittsweise teilnehmenden MSA-Studierenden haben sich immerhin noch knapp 1/3 (9 von 29) mindestens ein Mal zu Wort gemeldet.

5 Fazit und Ausblick

Ziel des Verbundprojekts war es, den Studierenden über ein breit angelegtes Projekt die Möglichkeit zu geben, die Entwicklung einer Microservice-Architektur in einem realitätsnahem Rahmen aktiv mitzugestalten. Dieses Ziel kann nach der Analyse des Feedbacks als erreicht betrachtet werden. Die Analyse der Projektkommunikation zeigt darüber hinaus eine hohe Aktivierung der Studierenden für die erfolgreiche Auseinandersetzung mit Architektur- und Infrastrukturthemen.

Dennoch wurde durch große Teile des Feedbacks auch Vieles aufgezeigt, das in weiteren Iterationen zu verbessern wäre. Der größte Vorteil der weiteren Projektiterationen liegt in der Tatsache, dass durch die Vorgängersemester bereits eine Systemlandschaft bereitsteht, die verstanden, erweitert und genutzt werden kann.

Die zweite und dritte Iteration im SoSe 2022 und WiSe 2022/23 wurde in etwas kleinerem Rahmen mit je zwei Modulen durchgeführt. Der Fokus lag in der zweiten Iteration zuerst auf einer Analyse und Weiterentwicklung, besonders hinsichtlich Robustheit der Servicelandschaft. In der dritten Iteration lag das Ziel auf einem Refactoring hin zu einer echten *Event-Driven Architecture* (EDA) der Systemlandschaft. Dabei fokussierten sich die Bachelorstudierenden auf die Entwicklung von Player-Services, ihre Master-Kommilitonen auf Core-Services. Besonders die Player-Entwicklung ermöglicht es, auch in zukünftigen Iterationen stetig neue Aspekte für die Studierenden zu schaffen. Weiterhin lassen sich auch Spielregeln und damit einhergehend Core-Services erweitern, so dass stetig neue Anreize und Herausforderungen für die Studierenden geschaffen werden können.

Der aktuelle Stand des Projekts ist unter [Mia] dokumentiert. Alle Sourcen sind öffentlich unter [Mib] zugänglich. Interessierte Lehrende sind herzlich eingeladen, das Konzept für die eigene Lehre zu adaptieren. Dabei ist sowohl ein umfassendes Lehrforschungsprojekt wie hier beschrieben denkbar, wie auch die Nutzung als fertige »Sandbox« für studentische Experimente mit Microservice- und EDA-Ansätzen.

An der TH Köln ist ein Forschungsantrag in Arbeit, um die Sourcen weiter zu entwickeln und das Projektformat damit auch langfristig bereitstellen zu können. Für eine Pflichtveranstaltung ist das Format vermutlich weniger geeignet, da wenig Fokus auf Prüfungsdesign liegt. Als Wahlveranstaltung für an Coding und Architektur interessierte Studierende erreicht es aber nachgewiesenermaßen die große Mehrheit der Teilnehmenden, und ist damit eine große Bereicherung des Curriculums.

Literaturverzeichnis

[AJ22] Al-Jarf, Reima: Interaction Analysis in Online Learning Communities: The Student Leader. Journal of Learning and Development Studies, 2(2):22–38, Jul. 2022.

[AK01] Anderson, Lorin W.; Krathwohl, David R., Hrsg. A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives. Allyn & Bacon, New York, 2. Auflage, December 2001.

[Be22] Bente, Stefan: , Learning Outcome der Veranstaltung Softwaretechnik 2 (ST2), 3 2022. Abgerufen 31.10.2022.

[BIK22] Bente, Stefan; Intveen, Jann; Krampe, Fabian: Divekit — Digitalisierung und Individualisierung als Schlüssel für eine moderne Softwaretechnik-Ausbildung. In (Thurner, Veronika; Kleinen, Barne; Siegeris, Juliane; Weber-Wulff, Debora, Hrsg.): Software Engineering im Unterricht der Hochschulen (SEUH 2022). Gesellschaft für Informatik, Bonn, S. 29–41, 2022.

[Br21] Brandolini, Alberto: Introducing EventStorming: An act of Deliberate Collective Learning. Leanpub, 2021.

[Co20] Cordeiro, Renato; Rosa, Thatiane; Goldman, Alfredo; Guerra, Eduardo: Teaching Complex Systems based on Microservices. GROUP, 1:1, 2020.

[Ev04] Evans, Eric: Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional, 2004.

[Hi21] Hillpot, Jeremy: , 4 Microservices Examples: Amazon, Netflix, Uber, and Etsy. `https://blog.dreamfactory.com/microservices-examples`, 2021. Abgerufen 25.01.2022.

[Mia] Microservice Dungeon: Aktuelle Dokumentation. `https://the-microservice-dungeon.gitlab.io/docs`. Abgerufen 05.11.2022.

[Mib] Microservice Dungeon: Git-Repositories. `https://gitlab.com/the-microservice-dungeon`. Abgerufen 05.11.2022.

[Mic] Microservice Dungeon: GitHub-Repositories (1. Projektiteration). `https://github.com/The-Microservice-Dungeon`. Abgerufen 08.01.2023.

[Ne15] Newman, Sam: Microservices: Konzeption und design. MITP-Verlags GmbH & Co. KG, 2015.

[Sc22] Schmeier, Philipp Felix: Moderne lose gekoppelte Softwarearchitekturen: Wie die Hochschullehre komplexe Systeme „hands-on“ vermitteln kann. Masterarbeit, Technische Hochschule Köln, Steinmüllerallee 1 51643 Gummersbach, Februar 2022.