Abschlussbericht

Team: SpeziRangers /NR.3

Mitglied 1: (Martin Beyer, 11909749)

Mitglied 2: (Nicolas Marte, 11909113)

Mitglied 3: (Islam Mechtijev, 11910366)

Mitglied 4: (Martin Neuner, 11917314)

Mitglied 5: (Clemens Prosser, 11907449)

Proseminargruppe: 6

Datum: 18.06.2021

1. Analyse des Projektablaufs

1.1 Meilensteine und zeitlicher Ablauf

Das Projekt wurde zunächst von uns sehr akribisch geplant. Alle Meilensteine wurden möglichst früh angesetzt, um gegebenenfalls auf einen Zeitpuffer zurückgreifen zu können. Bereits nach Ostern war dieser Zeitplan nur noch schwer einzuhalten, da während der lehrveranstaltungsfreien Zeit nur zwei Teammitglieder einen in anderen Wochen durchschnittlichen Wochenaufwand erbracht haben.

Am 20. April ist dann noch zusätzlich ein Teammitglied unvorangekündigt aus dem Projekt ausgestiegen, ohne eine Zeile Code beigetragen zu haben. Zu diesem Zeitpunkt wäre der ursprüngliche Zeitplan bereits nur noch mit großem Einsatz von sechs Teammitgliedern einzuhalten gewesen und Issues waren dementsprechend verteilt. Da alle fünf verbleibenden Teammitglieder durch ihr Studium nur begrenzte Stunden pro Woche in das Projekt investieren konnten, mussten alle Meilensteine ab Meilenstein 1 und Features zeitlich verschoben werden.

In der folgenden Tabelle sind die Meilensteine aufgelistet: (vgl. Konzeptbeschreibung, Projektplan)

Bezeichnung	**Ursprünglich Geplante Fertigstellung**	**Tatsächliche Fertigstellung**
Fertigstellung Konzept	18.03.2021	18.03.2021
Project Setup / Konfiguration	28.03.2021	28.03.2021
Meilenstein 1	11.04.2021	20.04.2021
Meilenstein 2	25.04.2021	02.05.2021
Meilenstein 3	13.05.2021	16.05.2021
Fertigstellung aller Features	23.05.2021	18.06.2021
Projektabschluss	13.06.2021	18.06.2021

Die Verzögerung über Ostern ist hier gut zu sehen. Bei Meilenstein 1 muss jedoch hinzugefügt werden, dass der Aufwand eines Issues falsch eingeschätzt wurde. Währenddessen wurde bereits an Meilenstein 2 gearbeitet.

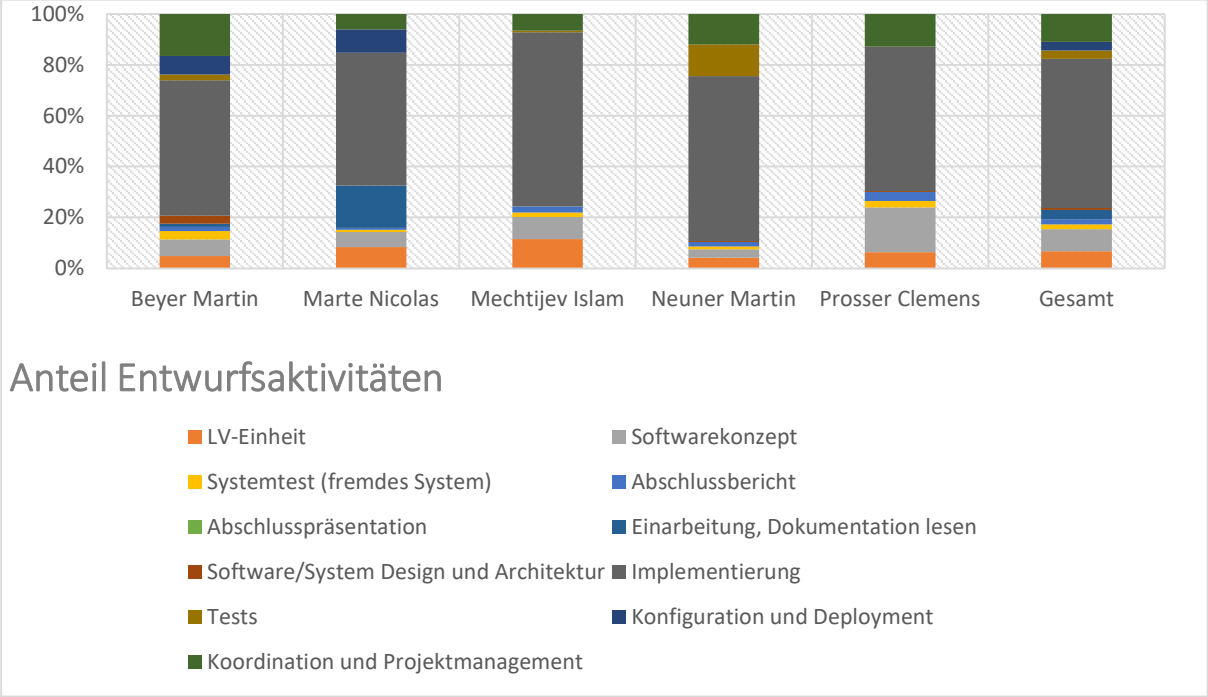
Die Verzögerung bei Meilenstein 3 ging einher mit der Verschiebung des Statistik Features. Trotzdem traten am Abend der geplanten Fertigstellung massive Bugs auf, die das Verpassen der Abgabe für den Abnahmetest zur Folge hatten.

Nach dieser Abgabe war das Team endgültig eingespielt und auftretende Verzögerungen konnten durch eine bessere Kommunikation kompensiert werden.

1.2 Analyse der geleisteten Stunden

Zunächst die Auflistung der geleisteten Stunden tabellarisch und graphisch (vgl. Summary.xlsx):

Geleistete Zeit [hh:mm]	Beyer Martin	Marte Nicolas	Mechtijev Islam	Neuner Martin	Prosser Clemens	Gesamt
Gesamt	199:08	201:50	148:20	214:25	226:25	990:08
LV-Einheit	9:35	16:50	17:05	8:50	14:20	66:40
Softwarekonzept	13:07	11:50	12:55	6:55	39:55	84:42
Systemtest (fremdes System)	6:25	2:00	2:30	2:30	5:45	19:10
Abschlussbericht	3:20	1:30	3:30	3:40	7:50	19:50
Abschlusspräsentation	0:00	0:00	0:00	0:00	0:00	0:00
Einarbeitung, Dokumentation lesen	2:30	33:30	0:00	0:00	0:00	36:00
Software/System Design und Architektur	6:20	0:00	0:00	0:50	0:50	8:00
Implementierung	98:50	105:30	97:55	126:30	124:45	553:30
Tests	1:00	0:00	1:00	26:30	0:00	28:30
Konfiguration und Deployment	14:36	18:30	0:00	0:00	0:00	33:06
Koordination und Projektmanagement	31:30	9:10	9:00	22:20	27:30	99:30



Hier fällt zunächst auf, dass die Implementierung den Großteil der Zeit einnimmt. Das liegt daran, dass vier Teammitglieder bereits praktische Erfahrung mit den eingesetzten Tools hatten und das meiste Recherchieren bei Bedarf, während dem Implementieren, stattfand. Ebenso ist es schwierig, Testen von der tatsächlichen Implementierung zu trennen.

Die aufgebrachte Zeit ist bei allen Teammitgliedern weit über den vorgesehenen 125h Arbeitsaufwand (5ECTS = 5 * 25h).

2. Analyse des implementierten Systems

2.1 Stabilität des Konzepts

Das initiale Konzept war nicht während der gesamten Entwicklung stabil. Da das Konzept im Vorfeld erstellt wurde, wurden nicht alle Usecases, Klassen, ... abgedeckt und mussten somit nachgetragen werden.

Viele Umgestaltungen gab es bezüglich der Abläufe der Use Cases. So wurden unter anderem im initialen Konzept andere Abfolgen beschrieben als in der finalen Version. Ein konkretes Beispiel ist die Abfolge der Registrierung. So war zu Beginn geplant, dass der User nach der erfolgreichen Registrierung zur Startseite weitergeleitet wird. Da das direkte automatische Einloggen jedoch nicht möglich war, entschieden wir uns den User zur Login-page weiterzuleiten.

Hinsichtlich dem fachlichen Klassendiagramm gab es auch signifikante Änderungen. Nicht nur die Beziehungen der Klassen haben sich wesentlich verändert, sondern auch die Klassen selbst. So wurden unter anderem Attribute und weitere Klassen ergänzt. Zu den ergänzten Klassen zählen zum Beispiel: *VirtualUser*, *VirtualTeam*, *UserRole*.

2.2 Stabilität der SW-Architektur

Präsentationsschicht:

Hier haben sich keine Änderungen ergeben. Vue.js war für das Frontend geplant und wurde somit auch verwendet.

Anwendungsschicht:

In dieser Schicht gab es wesentliche Änderungen. Grundsätzlich hat man sich für eine persistente Speicherung aller Daten entschieden. Während des Projektes hat man das Potential von In-Memory Klassen erkannt, diese wurden schließlich realisiert.

Persistenzschicht:

Hier haben sich keine Änderungen ergeben.

2.3 Funktionalität

Die gesamte geplante Funktionalität des Softwarekonzepts konnte realisiert werden. Diese wurden mittels modern gestalteter GUI möglichst benutzerfreundlich gestaltet und umgesetzt. Dies bietet dem Benutzer implizit die Möglichkeit, die Applikation auf jedem Endgerät zu verwenden.

Die Ausfallsicherheit des Systems entspricht nicht vollständig dem vom Projektteam angestrebten Qualitätsstandard.

2.4 Qualitätsmanagement

Die Verwendung von GitLab erleichterte das Arbeiten im Team erheblich. Besonders GitLabs‘ Tags haben unsere Organisation signifikant erleichtert (Todo, Doing, …). Durch unseren kontinuierlichen Review-Prozess konnte Code Quality sichergestellt werden. Besprochene Kriterien konnten so validiert werden, ebenso konnten Bugs frühzeitig gefunden und eliminiert werden.

In Kombination mit unseren mehrmals pro Woche stattfindenden Jour Fixe Terminen konnten wir unsere derzeitige Qualität erreichen.

3. Ursachenanalyse

3.1 Ursachen der Probleme

Zeitprobleme

Wie bereits beschrieben lässt sich die Verschiebung vieler Meilensteine erklären. Später wurden Meilensteine absichtlich früher definiert, damit jeder Meilenstein auch ein Puffer besitzt. Auch sind wir mit einer Verspätung in das Projekt gestartet. Nur zwei der damals noch 6 Mitglieder arbeiteten über die Osterferien an dem Projekt. Wir hätten vermutlich schon in den Arbeits-Rhythmus kommen können, indem wir die ersten Programmier-Tickets bereits vor Ostern erledigt hätten. So wären wir hoffentlich nicht allzu groß in der Versuchung geraten, eine Oster-Ruhe einzulegen.

Der erhöhte Zeitaufwand lässt sich nur bedingt erklären. Die Ursache liegt zum einen an der Reduktion der Teamgröße, zum anderen aber an der Proseminarorganisation. Ohne diesem erhöhten zeitlichen Einsatz hätten wir nicht alle Punkte aus der Aufgabenbeschreibung erfüllen können.

Kommunikationsprobleme

Zum Beginn des Semesters war teilweise unklar, wie der Fortschritt mancher Teammitglieder ist. Manche Teilnehmer führten laufende Änderungen offline durch (ohne sie zu synchronisieren; bis das Issue abgeschlossen ist).

Nach Ostern wurden daher strengere Kommunikationsregeln (checkin – checkout) eingeführt. Diese dienten als Workaround, da manche Teammitglieder nicht auf Offline Änderungen verzichten wollten. Diese Strategie wirkte sich auch signifikant auf die Motivation aus: Teilnehmer waren aufgrund der (fast) täglichen Push-Benachrichtigungen wesentlich mehr motiviert, am Projekt zu arbeiten. Diese Kommunikationsregeln hätten demnach schon zu Beginn des Projektes eingeführt werden sollen.

Fehlendes Wissen zu Java Spring

Eine weiters Problem war die geringe Wissensgrundlage bezüglich Java Spring. Die Grundlagen aus Softwarearchitektur waren bereits bekannt, jedoch konnten wir selten dieses Wissen anwenden, da wir oft in Spezialfälle geraten sind, die nur durch aufwändiges Lesen der umfassenden Spring Dokumentation gelöst werden konnten.

3.1 Findings

Lockere Planung
Ein lockerer Zeitplan sollte gewählt werden. Unsere Meilensteine waren sehr optimistisch angesetzt – sofern manche Mitglieder ihre Aufgaben nicht zeitgerecht erledigen konnten, führte das implizit zur Verzögerung eines Meilensteins.

Vor allem der spontane und unerwartete Wegfall unseres 6. Teammitgliedes hat uns endgültig zu dieser Erkenntnis gebracht. Sämtliche Aufgaben mussten dieser Ressourcenveränderung angepasst werden.

Genauere Einschätzung der Aufgaben
Auch wenn unsere Issues sehr genau definiert wurden (inkl. hilfreicher Referenzen, …), wurden benötigte zeitliche Ressourcen mancher Aufgaben falsch eingeschätzt (Bsp.: Authentification / Authorization). Grundsätzlich muss man allerdings mit falschen Einschätzungen wie diesen rechnen, sofern man, den Entwicklern unbekannte, Systeme nutzt (in unserem Fall konkret Spring).

Diversität der Aufgabenverteilung
Manche Aufgabengebiete wurden gewissen Mitgliedern während des gesamten Projektablaufs zugewiesen. Das Hauptbeispiel stellt die Implementierung des Cubes dar, da dieser nur mit erhöhtem zeitlichen Aufwand an andere Mitglieder übergeben werden konnte. Entwickler (abgesehen vom Ersteller der Implementierung) hatten somit Probleme bei Nachvollziehbarkeit, Implementierung und Testen der jeweiligen (Cube-) Methoden. Sofern keine Limitierungen vorliegen (Bsp.: in der Hardware), sollten nicht ganze Aufgabengebiete einem Entwickler zugesprochen werden.

Bessere Wahl des CSS-Frameworks
Es wurde Tailwind als CSS-Framework benutzt. Dies gab uns mehr Freiheiten im Design. Gleichzeitig wurden unerfahrene Teilnehmer aufgrund der Komplexität eingeschränkt. Ein CSS anfängerfreundliches Framework wie vuetify wäre eine deutlich bessere Wahl gewesen.

Tests, Tests, Tests
Manche Teammitglieder haben Systemtests (explizit Unit-Tests) als zweitrangig eingestuft und demnach vernachlässigt. Mergerequests hätten abgelehnt werden sollen, sofern nicht ausreichend Unit Tests vorhanden sind. (Das impliziert allerdings gegebenenfalls eine Erhöhung der erforderlichen Zeit je Issue.)

4. Erfahrungen mit den eingesetzten Werkzeugen

Boten die verwendeten Werkzeuge adäquate Unterstützung bei der Entwicklung?

Zur Realisierung haben wir auf ein drei-Schichten-Modell gesetzt. Vor allem unsere Vue.js Frontend Single Page Application (SPA) hat uns stark unterstützt, da einige Projektmitglieder bereits Fachwissen aufweisen konnten. Spring war für uns relativ neu, allerdings konnten wir gängige Programmierparadigma anwenden und die Knowledge Gap umgehen.

Auch unsere WebSocket Implementierung, welche Echtzeit-Kommunikation zwischen verschiedenen Teilnehmern ermöglicht, war ein Erfolg. Um einen möglichst reibungslosen Ablauf zu gewährleisten, haben alle Projektmitglieder möglichst ähnliche technische Voraussetzungen realisiert (ähnliche Versionen von Drittsystemen, Environment, ...).

Anfangs wurde ein hybrider Kommunikationsfluss gewählt - schriftlicher Austausch erfolgte mittels Slack sowie mündlicher Austausch mittels Discord (für Bildschirmübertragungen, ...). Probleme konnten so bestmöglich besprochen und behoben werden, ohne dass ein Punkt übersehen wurde. Zuletzt schließt unser Vorgehen ein konsistentes, sauberes und branchenübliches GIT Versionierungsmodell ab. Änderungen konnten zu jedem Zeitpunkt nachvollziehbar eingesehen und beurteilt werden. Ebenso konnten Fehler durch Reviews behoben und Änderungsvorschläge eingepflegt werden.

Mit welchen Werkzeugen hatten Sie Probleme? Welche haben gut funktioniert?

Vor allem der Umgang mit Spring war aufgrund der damit verbundenen mangelnden Erfahrung unsererseits kritisch. Die Kommunikation via Discord / Slack war anfangs zu selten (1x pro Woche), wurde allerdings erhöht und somit stark verbessert.

Besonders gut funktioniert haben In-Memory Implementierungen. Nach initialen Implementierungsentscheidungen konnten wir ohne weitere Probleme dieselben Grundgedanken (für In-Memory Klassen) auf andere Klassen anwenden.

Auch das angewendete 3-Schichten-Modell, in diesem Fall konkret bestehend aus einer SPA im Frontend, Spring im Backend und einer SQL Datenbank in der Persistenzschicht, hatte (bis auf das mangelnde Wissen seitens Spring) ausschließlich Vorteile.

5. Feedback zur Proseminar-Organisation

Die Durchführung solcher Projekte finden wir grundsätzlich sinnvoll, allerdings sollte die Vorgehensweise **stark** überarbeitet werden. Nachfolgend finden sich einige Probleme und (sofern nicht trivial) Lösungsansätze:

- **Fehlende Informationen zu Spring**
  Im Projekt wird Spring verwendet (zwingend vorgeschrieben). Im restlichen Bachelorstudium Informatik werden allerdings generell nur grundlegende Kenntnisse in Java vermittelt – Spring wird demnach **nicht** (ansatzweise ausreichend) behandelt. Auch wenn Spring für den Anwendungsfall geeignet ist, stellt das einen kritischen Aspekt dar.
  Lösung: Ausreichend Informationen zur Verfügung stellen (Beispielsweise 4-6 Stunden Einführung in Spring, in welchen typische Szenarien beispielhaft gelöst werden; Einführung in Softwarearchitektur nicht ausreichend)
- **Vorlesung vom Seminar entkoppelt**
  Grundsätzlich dient ein Proseminar dazu, Inhalte der Vorlesung zu vertiefen. Das ist hier nicht möglich, da die Vorlesung erst sehr spät im Semester stattfindet. Inhalte der Vorlesung werden wesentlich weniger vertieft. Zusätzlich ist den Studierenden nicht klar, auf welche granularen (!) Eigenschaften wertgelegt wird (Bsp.: „Was wird am Code beurteilt? Wie sollten Bestandteilte des Konzeptes richtig realisiert werden?“)
  Lösung: Vorlesung an Seminar koppeln und gegebenenfalls kleine wöchentliche Übungen ansetzen
- **Zeitintensivität**
  Das Projekt überschreitet bei allen Teammitgliedern den vorgesehenen Stundenaufwand von $125h$ (5ECTS $\rightarrow 5 * 25h$) sehr. Das ist äußerst fatal, da Studierende so andere Lehrveranstaltungen zwingend (aus zeitlichen Gründen) vernachlässigen müssen. Das der effektive Stundenaufwand vom vorgesehenen Stundenaufwand grundsätzlich abweicht, ist nicht vermeidbar – eine dermaßen hohe Differenz kann allerdings sehr wohl verhindert werden.
- **Fachwissen wird intuitiv bestraft**
  Personen, welche Fachwissen aufweisen, müssen wesentlich mehr Zeit für das Projekt aufwenden. Zu den zusätzlichen Tätigkeiten gehört das Einpflegen von Standards, umfangreiche Reviews von Merge Requests, Beratung bei Fragen von Teammitgliedern, Behebung besonders schwieriger Bugs, erweiterte Verwaltung des Projektes, …. Da man diese Tätigkeiten meist in Kombination mit anderen Tätigkeiten erledigt (Fragen könnten beispielsweise $24/7$ gestellt werden), können keine genauen Arbeitsstunden ermittelt und notiert werden.
  Da grundsätzlich (lt. Kommunikation) keine individuelle Beurteilung pro Teammitglied vorgenommen wird, sind Personen mit Fachwissen intuitiv dazu angehalten, mehr Aufgaben zu übernehmen. Würde eine Person ohne Fachwissen dieselbe Aufgabe erledigen, muss die

jeweilige Aufgabe meist nachbearbeitet werden (ansonsten wäre auch die eigene Beurteilung von der gegebenenfalls schlechteren Qualität anderer Mitglieder stark betroffen). Die Nachbearbeitung benötigt gegebenenfalls wesentlich mehr Zeit, als wenn die jeweilige Person mit Fachwissen diese Aufgabe direkt erledigt hätte.
Lösung: Individuelle Beurteilung pro Teammitglied (inkl. Sauberer Aufstellung / Zusammensetzung)

- **Teamgebundene Noten**
  Sofern manche Teammitglieder geringere Qualitätsvorstellungen haben, erledigen diese gegebenenfalls weniger Arbeit. Darunter leidet allerdings das gesamte restliche Team. Aus Sicht des jeweiligen Proseminarleiters ist das leider nur sehr schwer zu erkennen.
  Lösung: Individuelle Beurteilung pro Teammitglied (inkl. Sauberer Aufstellung / Zusammensetzung)
- **Maximale Teamgröße nicht kommuniziert**
  Die maximale Teamgröße wurde nicht (ausreichend) kommuniziert.
  Konkret bestand unser Team grundsätzlich aus 5 Personen, wobei am Anfang des Proseminars eine 6. Person uns zugewiesen wurde (Diese Person war keinem Teammitglied zuvor bekannt). Die 6. Person wurde vollständig in die bereits existenten Pläne eingeweiht und eingeplant. Kurz vor der ersten Abgabe des Projektes hat diese Person das Seminar ohne Ankündigung und Vorlaufzeit aufgrund einer dubiosen Geschäftsreise verlassen – bis heute befindet sich von dieser Person keine Codezeile im Main-Branch, zusätzlich wurde nach Absprache der Seminarleiter keine (uns bekannte) Nachsicht ausbesprochen.
  Wäre die maximale Teamgröße sauber kommuniziert worden, hätten wir diese befüllt und das Problem intuitiv vermeiden können.
  Lösung: Teamgröße eindeutig bestimmen und kommunizieren
- **Lehrveranstaltungsfreie Zeit**
  Seitens der Aufgabenstellung wurde ganz klar vorausgesetzt, dass man auch während der Lehrveranstaltungsfreien Zeit (Bsp.: Ostern) überproportional aktiv am Projekt arbeitet. Aus Sicht der Studenten ist das fatal, da besonders die lehrveranstaltungsfreie Zeit für Aufarbeitung / Wiederholung von Vorlesungen von Vorteil ist.
  Lösung: Zeitintensität verringern
- **Proseminarzeit**
  Die Proseminarzeit wird leider sehr ineffektiv genutzt. Größtenteils finden Status Quo Besprechungen statt – diese haben für den Großteil der Kursteilnehmer allerdings nur einen kleinen (wenn überhaupt) Mehrwert. Der interne Status ist dem jeweiligen Teammitglied grundsätzlich bekannt, der Status anderer Teams ist den eigenen Teammitgliedern vermutlich gleichgültig. Sofern zumindest ein Mitglied pro Team anwesend ist, sollte ein angemessener Kommunikationsfluss mit dem jeweiligen Seminarleiter problemlos möglich sein.
  Lösung: Seminar als optionalen Termin gestalten (außer min. 1 Teammitglied pro Team), oder wichtige Informationen am Anfang der jeweiligen Seminarstunde bekanntgeben und die Status Quo Besprechungen als optional einstufen. Sofern ein Teammitglied den Status eines anderen Teams wissen will, kann dieser optional an der Besprechung teilnehmen oder gegebenenfalls nachträglich die jeweilige Mitschrift / Präsentation einsehen.
- **Word Vorlagen**
  Grundsätzlich werden Vorlagen für verschiedene Dokumente zur Verfügung gestellt. Es ist eine Voraussetzung, diese auch zu benutzen. Das ist in einem Informatikstudium nicht zeitgemäß. Solche Dateien können schlecht versioniert, zusammengeführt, gleichzeitig bearbeitet, … werden. Zusätzlich wird beispielsweise Word in keiner anderen Lehrveranstaltung im Bachelorstudium Informatik vorausgesetzt. Latex wäre wesentlich sinnvoller, da sich Studierende so unter anderem auf Syntax zukünftiger wissenschaftlicher Arbeiten vorbereiten können.

Lösung: Word Vorlagen als optional einstufen – ein Team könnte, sofern es sich so entscheidet, Latex einsetzen; andere könnten genauso Word nutzen.

- **Dokumente in Deutsch**
  Wichtige Dokumente wie Konzept, Abschlussbericht, … müssen in deutscher Sprache verfasst werden. Das führt zu wiederkehrenden Problemen, da Code grundsätzlich in Englisch verfasst wird. Zusätzlich stellt das eine optimale Möglichkeit dar, Englisch-Kenntnisse mancher Studierender für zukünftige wissenschaftliche Arbeiten zu verbessern.
  Lösung: In den angeführten Dokumenten sowohl Englisch als auch Deutsch erlauben.
- **Angabe ungenau / fragwürdig**
  Manche Stellen der Angabe erweisen sich als äußerst ungenau (weswegen im Seminar häufig Fragen gestellt wurden), ebenso sind einige Stellen sehr fragwürdig (Bsp.: Andere Teams müssen bestätigen, sofern ein Wort *nicht* erraten wurde).

  Der letzte Hinweis in der Aufgabenstellung besagt, dass alle „Arbeitsabläufe, Features und Anforderungen […] zwingend, vollständig und in angemessener Qualität“ abzugeben sind, um eine positive Beurteilung zu bekommen. Wir finden diesen Hinweis sehr streng und bezweifeln, dass sich die Proseminarorganisation exakt danach richten wird. Dies war auch in Status Quo Präsentationen anderer Teams deutlich zu erkennen.
  Lösung: Finetuning der Angabe
- **Standards kommunizieren**
  Im Proseminar kam es während der Status Quo Besprechungen oft zu offensichtlichem verstoßen gegen Standards. Diese Verstoße wurden leider nicht hinreichend behandelt, weswegen sie für viele Studenten nach wie vor unbekannt sind. Beispielsweise denken viele Studierende, eine Schnittstelle (API), welche wer HTTP aufgerufen wird, sei implizit eine REST-API. Das ist aber offensichtlich nicht der Fall.
  Lösung: Fehler korrigieren
- **Kommunikation per Olat**
  Von den Seminarleitern wurden Fragestellungen per Olat Forum gewünscht, da so Fragen für alle Teilnehmer einsehbar sind. Offensichtlich hat sich der Delay zwischen Fragestellung und Antwort gegenüber einer direkten Mail an den jeweiligen Seminarleiter stark erhöht. Ein entsprechendes Monitoring (Olat ermöglicht E-Mail-Benachrichtigungen bei Erstellung eines Beitrags) ist erwünscht, da der Delay so stark reduziert werden kann.
  Lösung: Benachrichtigungen aktivieren
- **Unzureichende Hardware**
  Das Verhalten des Würfels ist nicht für dieses Spiel adäquat. Eine Benachrichtigung erfolgt nicht, wenn der Würfel gewürfelt wird und nochmals auf derselben Seite landet. Weiters muss der Würfel umständlich konfiguriert werden, damit das Mapping aus der Aufgabenbeschreibung passt. Zusätzlich müssen fortgeschrittene Algorithmen zur Sicherstellung einer angebrachten Stabilität verwendet werden.
  Lösung: Es sollte ein angebrachtes Framework für die Interaktion mit dem Würfel zur Verfügung gestellt werden. Alternativ kann auch die Aufgabenbeschreibung entsprechend angepasst werden.
- **Workshops**
  Grundsätzlich sind Workshops sehr hilfreich und sollten weiterhin angeboten werden. Allerdings minimiert die Teilnehmerbeschränkung und die oft fehlende Aufzeichnungen die Sinnhaftigkeit.