Einzelreflexion Turnierauswertungssoftware Team C

Inhaltsverzeichnis

1. Maximilian Lohr

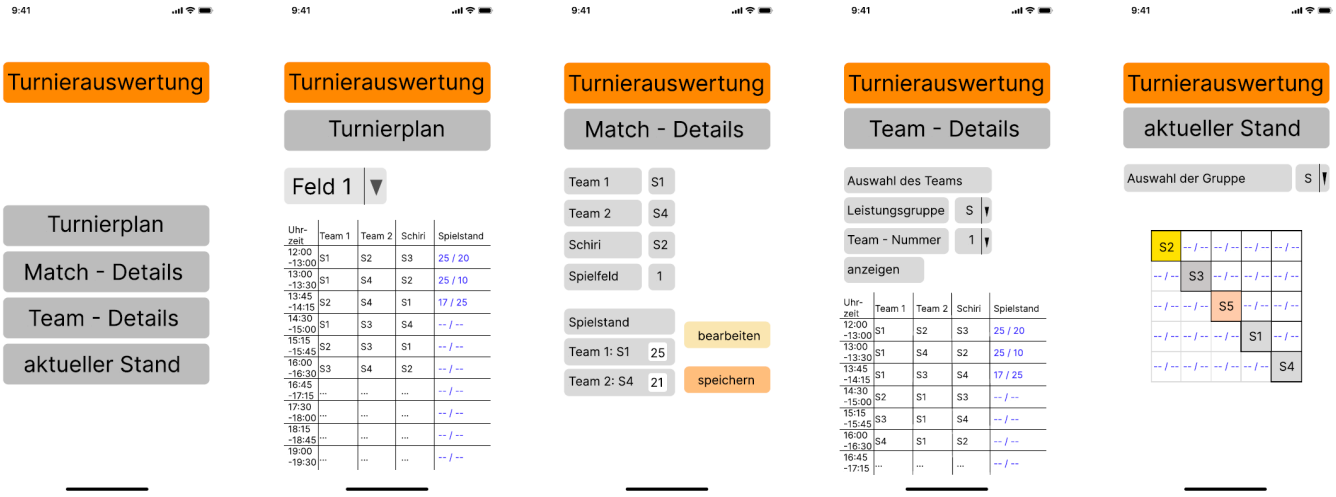
Konzeption und iterative Verbesserung der Nutzeroberfläche mithilfe von Wireframes

Ausgangssituation

In der ersten Projekthälfte lag der Fokus auf der Analyse- und Entwurfsphase im Sinne des OpenUP-Prozesses, um eine solide Grundlage für die spätere Implementierung der Turnierplanungssoftware zu schaffen. Neben der Anforderungserfassung in den OpenUP-Dokumenten und User Stories spielen dabei Wireframes eine zentrale Rolle in der UI/UX-Entwicklung.
Nachdem Alexander bereits einen ersten Entwurf dazu auf Papier skizziert hatte, kümmerte ich mich um die Weiterentwicklung und Umsetzung in digitalem Format. Im Zuge der iterativen und inkrementellen Entwicklung, wie sie in OpenUP und agilen Prozessen empfohlen wird, haben wir im nächsten Auftraggeber-Meeting (Issue #40) konkretes Feedback zu den Wireframes eingeholt und in den nächsten Entwurfsschritt integriert. Dazu zählte unter Anderem eine Verbesserung der Übersichtlichkeit von Informationen zum aktuellen Spielstand in der Seite „Match – Details“ und das Einführen einer klareren Strukturierung im Spielplan.

Lösungsweg

Basierend auf diesem Feedback habe ich mit Tim zusammen einen Wireframe für die mobile Darstellung aus Spielersicht entwickelt:



Dazu haben wir einige Änderungen vorgenommen. Zuerst wurde eine Start/Home-Seite eingerichtet, welche alle weiteren Seiten verlinkt. Im Turnierplan wurden Filter für das Feld hinzugefügt, um die Übersichtlichkeit zu steigern. Außerdem führt die Einführung der Spalte „Uhrzeit“ zu mehr Struktur und einer verständlichen Abfolge. Die Seite „Match – Details“ wurde so verändert, dass ein zusätzlicher Button zum Bearbeiten hinzugefügt, um versehentliche Änderungen zu vermeiden.
Die Anpassungen basierten auf den zuvor ausgearbeiteten User Stories, Personas und Taskflows. Dies entspricht dem nutzerzentrierten Ansatz der Requirements-Analyse, bei dem konkrete Nutzungsszenarien als Grundlage für Entwurfsentscheidungen dienen. Somit konnten wir sicherstellen, dass der Nutzungskontext immer beachtet wurde.
Nach weiteren Absprachen mit dem Auftraggeber entstand daraus die finale Version der Wireframes aus Spieler- sowie Organisatorsicht. Dabei wurde klar, dass das aktuelle Bedienkonzept immer noch zu komplex ist und es zu viele Buttons und Informationen gibt. Vor Allem die Seite „Match – Details“ soll nun weniger Informationen enthalten. Des Weiteren sollen die Farben vom StuRa deutlicher verwendet werden. Mit diesem Feedback entwickelte ich einen überarbeiteten Wireframe für die Nutzeransicht.
Da sich die Arbeit mit Figma als Tool allerdings für uns als sehr zeitaufwendig und wenig intuitiv herausstellte entschieden wir uns den Fokus, wie in der Vorlesung empfohlen, auf der Funktionalität zu belassen anstatt unsere Ressourcen für eine grafisch ästhetische Oberfläche zu verwenden, welche dann in der späteren Implementierung per HTML und CSS eh erneut Aufwand macht. Zudem ist leider der Bearbeitungslink zu dem oben eingefügten Screenshot verloren gegangen, so dass ein kompletter Neuanfang nötig gewesen wäre. Für mich war das ein gutes Learning, da ich nun genauer darauf achte, dass alle entwickelten Ressourcen auf GitHub im Issue verlinkt werden. Seitdem gab es keinen ähnlichen Vorfall mehr. Dieses Learning entspricht dem in Retrospektiven üblichen Vorgehen, Arbeitsprozesse regelmäßig zu hinterfragen und zu verbessern.
Zudem habe ich in Zusammenarbeit mit Tim auf Discord (Issue #81) ebenfalls einen finalen Wireframe für die Organisatoren-Sicht erstellt. Dieser enthielt nun zusätzlich eine Konfigurationsseite zum Einstellen der Turnierparameter, einen hochskalierten Turnierplan mit direkter Änderungsmöglichkeit der Ergebnisse ohne Extra-Seite, eine intuitivere Ansicht für die Ergebnisliste und eine Historie-Ansicht für zuvor gespeicherte Turniere. Auch die Farben des StuRa wurden klarer eingesetzt, da nun aus der von Lotte zugeschickten Farbpalette die Petrol-Abstufung #58AAA0 als Kontrastfarbe verwendet wird.
Die Anforderungen dazu konnten wir ebenfalls anhand des Feedbacks zu unserem ersten Prototypen ableiten. In dem Wireframe zur Nutzeransicht wurde zudem die Dialogdynamik genau beschrieben, damit nachvollziehbar wird, welche Inhalte der Seite wie bearbeitbar sind bzw. welche Aktionen ein Klick auf einen Button auslöst. Beide Entwürfe (Nutzer- und Organisatoransicht)

wurden in dem folgenden Teammeeting vorgestellt und besprochen sowie im UX-Concept unter Punkt 3.2 „Wireframes“ eingebettet. In den folgenden Auftraggeber-Meetings wurden die Entwürfe mit minimalen Ergänzungen abgenommen.
Der Fokus in der Implementierung lag vorerst darauf die Organisatorsicht umzusetzen, da wir uns bereits bewusst waren, dass wir zeitlich wahrscheinlich nicht alle erfassten Anforderungen umsetzen werden können und die Organisatorsicht dabei höhere Priorität hat, da damit theoretisch das Turnier ebenfalls ausgetragen werden kann, weil die Grundfunktionalitäten alle verwirklicht sind.
Als Frontend-Entwickler war es im weiteren Verlauf auch meine Aufgabe die Wireframes in der Webanwendung abzubilden (Issue #184). Die vorentwickelte Visualisierung reduzierte Missverständnisse in der Implementierung und erleichterte die spätere komponentenbasierte Entwicklung im Frontend. Schlussendlich wurden die Wireframes damit zu 90% so auch in der finalen Anwendungsoberfläche angewandt. Kleinere Änderungen gab es nur, wenn beim Benutzen der Anwendung aufgefallen ist, dass es unkomfortabel zu bedienen ist oder das Design noch verschönert werden könnte.

Bewertung

Abschließend betrachtet war die Wireframe-Entwicklung ein Erfolg. Das iterative Vorgehen und ständige Feedback in den Auftraggeber-Meetings haben zur Vermeidung von Unzufriedenheit und Kritik bei der finalen Abnahme beigetragen. Auch wenn am Ende nicht alle Funktionalitäten in der Software umgesetzt werden konnten, sind die übernommenen Entwürfe für die Frontend-Entwicklung essenziell gewesen.
Auch die teaminterne Absprache in den wöchentlichen Meetings und die direkten Zusammenarbeit mit Tim hat dabei geholfen. Gerade die gemeinsame Entwicklung des letzten Wireframes aus Organisatorsicht hat mir dabei gezeigt, dass gleichzeitiges und gemeinsames Arbeiten an einer Aufgabe keine Zeitverschwendung ist, sondern die Qualität erhöhen und zur Vermeidung späterer Nacharbeiten beitragen kann.
Dennoch muss man anmerken, dass dieses iterative Vorgehen mit insgesamt fünf Entwürfen viel Zeit in Anspruch genommen hat, welche am Ende des Projektes mehr in die Implementierung hätte einfließen können. Wenn Projekte also stark zeitlimitiert sind, kann es ggf. in Zukunft von Vorteil sein kleinere Änderungswünsche direkt in Form der Frontend-Entwicklung dem Auftraggeber zu präsentieren oder zu ändern.

2. Kareen Khouri

3. Jonas Grießhaber

4. Alexander Zajcev

Entwurf und Implementierung der Request-Verarbeitungskette unter Verwendung des Model-View-Presenter Architekturmusters.

Ausgangssituation

Im Prototyp war die Code der Benutzerobeflächengenerierung stark mit der Code der Anwendungslogik vermischt. Es hat ausgereicht um die gewünschte Funktionalität der Anwendung bereitzustellen. Aber das hat die Zusammenarbeit an der Code mit den anderen Teammitglieder stark verhindert. Die Störfaktoren waren unter anderem:

- Große Anstrengung für jeden Teammitglied beim Einstieg in die Programmierarbeit, weil jeder sich mit der kompletten Anwendungslogik auseinandersetzen musste.
- Viele potenzielle Konflikte im Git, da die zuerfüllende Aufgaben mehrerer Programmierer sich oft auf die gleiche Quelltext-Datei bezogen.
- Situationen, wenn ein Entwickler auf die Arbeit eines Anderen warten muss , bevor er eigene Aufgaben erledigen konnte.
- Eventuelle zukünftige Verwendung eines alternativen Datenbankmanagementsystems wäre mit zu großen Änderungen verbunden.

Lösungsweg

Um die Schwierigkeiten zu adressieren haben wir uns im Team in der **Entwurfsphase** entschieden eine saubere "Separation of Concerns" durchzusetzen. Den MVP-Architekturmuster fand ich dafür am besten geeignet und habe mit der **Implementierung** angefangen.

View-Presenter-Verbindung

Die Daten, die durch eine View-Komponente dem Nutzer angezeigt werden sollen, waren großenteils noch aus der Analyse-Phase des Projekts bekannt. Für jede View-Komponente musste eine Klasse für die anzuzeigende Daten entworfen werden. Die Daten-Klasse diente dann als Schnittstelle zwichen der Controller-Komponente und dem View.

Somit waren für jede durch den Nutzer aufgerufene Ansicht drei Klassen notwendig: die View-Klasse, wo der endgültige HTML-Response erstellt wird, die Controller-Klasse, die die Daten für das View bereitstellt und die Datenschnittstellen-Klasse selbst.

Als erstes habe ich View-, Controller- und Datenschnittstellen-Basisklasse implementiert. In den Basisklassen waren die Initialisierungsvorgänge und andere für alle Kindklassen notwendige Prozeduren versteckt. Damit habe ich erreicht, dass zum Beispiel ein Entwickler, der für eine bestimmte View-Komponente zuständig ist, sich fast ausschließlich mit dem Rendern der HTML-Ausgabe beschäftigen kann.

Die von der Datenschnittstellen-Basisklasse geerbte Datenschnittstellen-Klasse konnte sowohl der View-Entwickler als auch der Cotroller-Entwickler anlegen. Damit war die Abhängigkeit der beiden Entwickler voneinander abgekoppelt und sie mussten nicht aufeinander warten.

Nutzeraktionen

Eine besondere Herausforderung war einen Weg zu finden, wie die View-Komponente die Benutzereingabemöglichkeiten in den HTML-Response integriert. Dafür habe ich Nutzeraktion-Klasse eingeführt. Controller übergab der View-Komponente Instanzen dieser Nutzeraktion-Klasse und View-Komponente konnte die Nutzeraktion-Instanzen auf die HTML-Benutzereigabemöglichkeiten (Hyperlinks, Eingabefelder, Buttons, …) abbilden.

Modell

Die für die Anwendung erforderliche Datenstrukturen wurden im Prototyp im Programm-Speicher verwaltet. Es gab Bedenken, dass die Leistung der InMemory-Datenverwaltung irgendwann nicht mehr ausreicht und deswegen ein externes Datenbanksystem (z.B SQLite) notwendig wird. Der Austausch des Datenverwaltungmoduls ist nur möglich, wenn die Datenabfragen und Datenmanipulationen in der Anwendung über eine genau definierte Schnittstelle durchgeführt werden. Die Datenbankschnittstelle habe ich als eine abstrakte Klasse realisiert. In der abstrakten Klasse sind die Funktionen nur deklariert und die Klassen der Datenstrukturen definiert. Konkrete Implementierung der Funktionen findet in den von der abstrakten Klasse abgeleiteten Datenbank-Backend-Klassen statt.

Documentation

Um den anderen Teammitglieder die Anwendungsstruktur zu erklären habe ich eine graphische Skizze des Requestsablaufs erstellt. Für einige Views habe ich die View-, Datenschnittstelle- und Controller-Klassen komplett implementiert, damit sie als Beispiel bei der Erstellung weiterer Klassen dienen. In jeder Datenschnittstellenklasse habe ich Platzhalterdaten definiert, damit die View-Entwickler die Beispiele bei noch nicht abgeschlossener Datenbankimplementierung nachvollziehen können.

Bewertung

Das Vorhaben ist meiner Meinung nach gelungen. Großer Teil der User-Stories kann mit der Requestverarbeitungskette realisiert werden. Auf die implementierte Struktur kann man sich verlassen. Die Arbeitsschritte für die Benutzung des Systems durch einen neuen Entwickler sind erlernbar. Aufgaben für die Entwickler können damit genau definiert und in einer vorhersehbaren Zeit erledigt werden.

In unserem Team war sowohl ein Entwickler, der von Anfang an da war als auch jemand, der später dazugekommen ist. Sie beide konnten sich gut in das Requestbearbeitungsprozess einarbeiten.

Aber es gibt noch ein Verbesserungsbedarf, der im Zeitrahmen des SE-Projekts nicht realisierbar war. Einige Schritte beim Anlegen der Requestverarbeitungskette für eine neue View sind zu umständig und müssen optimiert werden. Einstieg eines neuen Entwicklers dauert noch zu lange und die Ursache dafür ist noch nicht ganz klar. Das könnte zum Beispiel an nicht intuitiven Schritten im Gesamtvorgehen liegen oder vielleicht an der nicht ausreichender Dokumentation.

Eine positive Entdeckung möchte ich noch erwähnen: in der Testphase ließen sich die Unit-Tests für die umgesetzte Struktur gut definieren.

5. Radmir Mullagaliev

1. Ausgangssituation / Problemstellung

Im Rahmen unseres Projekts zur Entwicklung einer Turnierverwaltungssoftware habe ich die Verantwortung für die Erstellung eines Algorithmus zur automatischen Spielplan-Generierung übernommen.

Im Dokument `Anforderungen.adoc` wurden folgende Anforderungen definiert:

- Alle Teams einer Gruppe sollen gegeneinander spielen.
- Jedes Spiel benötigt ein drittes Team als Schiedsrichter.
- Es gibt eine begrenzte Anzahl an Feldern und Timeslots.
- Die Schiedsrichterrollen sollen fair verteilt werden.
- Bestimmte Muster (z. B. **Schiri–Pause–Schiri** oder **Schiri–Schiri**) sollen vermieden werden.
- Kein Team soll mehr als eine Pause hintereinander haben.
- Kein Team darf gleichzeitig spielen und Schiedsrichter sein.

Zu Beginn des Projekts gab es keinen passenden Algorithmus, der all diese Regeln gleichzeitig erfüllen konnte. Deshalb war eine vollständige **Anforderungsanalyse**, ein eigener **Systementwurf** sowie Tests nach den Prinzipien der **Verifikation und Validierung** notwendig.

2. Lösungsweg

Zuerst habe ich die grundlegende Logik in einem Dokument namens `Algorithmus_Turnierplangenerator.adoc` beschrieben. Dort habe ich die Abläufe wie in Use-Case-Szenarien dargestellt – von der Spielgenerierung bis zur Zuweisung von Rollen und Slots.

Für die Umsetzung habe ich mich an folgenden Prinzipien orientiert:

- Iteratives Vorgehen gemäß Unified Process (Construction Phase)
- Architekturprinzipien wie **Separation of Concerns**, **Loose Coupling** und **High Cohesion**
- SOLID-Prinzipien, insbesondere das **Single Responsibility Principle (SRP)** und das **Dependency Inversion Principle (DIP)**
- **Modultests** und **Systemtests** zur Qualitätssicherung

Die zentrale Logik wurde in der Datei TurnierplanGenerator.java implementiert.

Zum Testen habe ich folgende Dateien verwendet:

- TurnierTest.java: Konsolen-Ausgabe des Plans
- TurnierSimulation.java: Simulation mit verschiedenen Parametern

Im Rahmen des **Objektentwurfs** habe ich außerdem Hilfsklassen erstellt, die den Prinzipien von **Information Hiding** und geringen Abhängigkeiten folgen.

3. Bewertung des Ergebnisses

Der entwickelte Algorithmus erfüllt die meisten Anforderungen und beachtet wichtige Einschränkungen. Der Spielplan wird Slot-für-Slot und Feld-für-Feld aufgebaut. Die Schiedsrichter werden möglichst fair verteilt.

Trotzdem gab es einige Herausforderungen:

- Bei vielen Teams und wenigen Slots ist es schwer, alle Regeln gleichzeitig einzuhalten.
- In Einzelfällen mussten Kompromisse gemacht werden (z. B. zwei Pausen hintereinander oder zu viele Schiri-Einsätze).
- Die Rollenverteilung ist nicht immer vollständig gleichmäßig.

Durch einen **evolutionären Architekturansatz** und ständiges **Refactoring** konnte ich jedoch die Struktur und Lesbarkeit des Codes deutlich verbessern.

Ich habe in dieser Arbeit gelernt:

- den Code iterativ zu verbessern
- **Clean-Code-Prinzipien** wie **KISS** und **DRY** anzuwenden
- **Testgetriebene Entwicklung** praktisch umzusetzen
- Architekturentscheidungen gut zu dokumentieren und verständlich zu erklären

4. Zusammenarbeit mit anderen Teammitgliedern

Mein Beitrag war eine wichtige Grundlage für andere Projektbereiche. Der Spielplan, den ich generiert habe, wurde später im Interface dargestellt. Zusätzlich habe ich die Möglichkeit zur Ergebnisbearbeitung direkt im Turnierplan umgesetzt. Dafür habe ich folgende Dateien geändert:

- RoleAdmin_TaskTurnierplan_Controller.java
- RoleAdmin_TaskTurnierplan_Renderer.java

Damit habe ich nicht nur an der **Systemlogik**, sondern auch an der **UI-Integration** und dem **User Interaction Flow** mitgearbeitet.

5. Bezug zu den Lernzielen des Moduls SE II

"Die Studierenden verfügen über Fähigkeiten und Fertigkeiten zur arbeitsteiligen Entwicklung modularer, wiederverwendbarer und erweiterbarer Softwaresysteme." (Quelle: **Einführung und Organisatorisches, SE II, Foliensatz, S. 5, Übergeordnetes Qualifikationsziel von SE II**)

Durch diese Aufgabe habe ich konkret zu den Lernzielen des Moduls beigetragen, insbesondere in folgenden Bereichen:

- Arbeitsteilige Entwicklung im Team
- Modularer und wiederverwendbarer Code
- Systemarchitektur mit klaren Schnittstellen

- Anwendung von Konzepten aus der Vorlesung:
  - **Architekturprinzipien** (z. B. Separation of Concerns)
  - **Entwurfsmuster** (z. B. SRP, DIP)
  - **Testmethoden** (z. B. Modultests)
  - **Codequalität** (z. B. Clean Code, Refactoring)