QuestMapper – Überblick

Ziele

1. Entwicklung eines Werkzeugs zur Visualisierung von NPCs, Quests und Abhängigkeiten eines Rollenspiels.
2. Ermöglichung einer einfachen Darstellung von NPCs, Quests und Abhängigkeiten.
3. Verwaltung und Überwachung der Questreihenfolge.
4. Bereitstellung einer intuitiven Benutzeroberfläche für Spielentwickler.
5. Interaktive Visualisierung von Abhängigkeiten zwischen Quests.
6. Unterstützung der Spielentwickler bei der Planung und Verwaltung ihrer Questreihenfolge.

Zielgruppe

Spielentwickler, die RPGs erstellen und verwalten möchten.

Termine

- Start: KW 06 2023
- Fertigstellung: Sommer 2023

Funktionalität

- Die Darstellung von NPCs und Quests als eindeutige Formen im Programm.
- Die Möglichkeit, Abhängigkeiten zwischen Quests mithilfe eines Graphen darzustellen.
- Eine intuitive Benutzeroberfläche, die einfach zu verwenden ist.
- Eine Möglichkeit, NPCs, Quests und Abhängigkeiten einfach hinzuzufügen und zu bearbeiten.
- Eine Übersicht über den Fortschritt und die Belohnungen der Quests.

Erwartetes Ergebnis

Ein benutzerfreundliches Programm, das Spielentwicklern hilft, ihre Questreihenfolge zu verwalten und zu überwachen. Es soll ihnen eine einfache Möglichkeit bieten, NPCs, Quests und Abhängigkeiten zu visualisieren und sie bei der Planung und Umsetzung ihrer Questreihenfolge unterstützen.

QuestMapper – Analyse Benutzeranforderung

- Funktionale Anforderungen

1. **NPC-Erstellung:**
   - Die Möglichkeit, NPCs mit Namen, Eigenschaften und anderen relevanten Informationen zu erstellen.
2. **Quest-Erstellung:**
   - Die Möglichkeit bieten Quests mit einer Beschreibung, Zielen und Belohnungen zu erstellen.
   - Die Möglichkeit bieten Quest einen bestimmten Typ zuweisen, welcher einen bestimmten farblichen Stil besitzt.
3. **Abhängigkeits-Visualisierung:**
   - Die Fähigkeit, Abhängigkeiten zwischen Quests visuell darzustellen, um die Reihenfolge der Quests zu verstehen und zu überwachen.
4. **Überwachung der Questreihenfolge:**
   - Die Fähigkeit, die Questreihenfolge im Auge zu behalten und gegebenenfalls zu ändern, um sicherzustellen, dass sie für den Spieler logisch und sinnvoll ist.
5. **Benutzerfreundliche Oberfläche:**
   - Eine einfache und intuitiv bedienbare Benutzeroberfläche, um Spielentwicklern die Arbeit mit dem Werkzeug zu erleichtern.
6. **Speicherung und Wiederherstellung von Daten:**
   - Die Fähigkeit, NPCs, Quests und Abhängigkeiten zu speichern und jederzeit wiederherzustellen, um den Spielentwicklungsprozess zu vereinfachen.
7. **Import- und Exportfähigkeit:**
   - Die Möglichkeit, Daten aus anderen Quellen zu importieren und auszugeben, um den Spielentwicklungsprozess noch weiter zu vereinfachen

QuestMapper – Analyse Benutzeranforderung

- Nicht Funktionale Anforderungen

1. **Performance:**
   - Es muss eine hohe Geschwindigkeit und Reaktionsfähigkeit aufweisen, um eine flüssige Nutzererfahrung zu gewährleisten.
2. **Benutzerfreundlichkeit:**
   - Die Benutzeroberfläche muss intuitiv und einfach zu verwenden sein.
3. **Skalierbarkeit:**
   - Das Tool muss in der Lage sein, mit steigenden Anforderungen und Datenmengen umzugehen.
4. **Verfügbarkeit:**
   - Das Tool muss zuverlässig und ständig verfügbar sein.
5. **Wartbarkeit:**
   - Das Tool muss einfach zu warten und zu aktualisieren sein, um kontinuierliche Verbesserungen und Funktionalitäten bereitstellen zu können.
6. **Datenkonsistenz:**
   - Das Tool muss bei Änderungen an einzelnen Daten, diese Änderungen an allen Ansichten übernehmen.

QuestMapper – Akzeptanzkriterien

1. **Funktionalität:**
   - Das Tool soll alle notwendigen Funktionalitäten bereitstellen, um NPCs, Quests und Abhängigkeiten einfach darzustellen und zu verwalten.
2. **Benutzerfreundlichkeit:**
   - Das Tool soll eine intuitiv zu bedienende Benutzeroberfläche bieten, die eine einfache Verwaltung der Questreihenfolge ermöglicht.
3. **Korrektheit:**
   - Das Tool soll korrekt funktionieren und alle interaktiven Visualisierungen von Abhängigkeiten zwischen Quests bereitstellen.
4. **Leistung:**
   - Das Tool soll schnell und stabil laufen und eine hohe Leistung aufweisen, auch wenn es große Datenmengen verarbeiten muss.
5. **Dokumentation:**
   - Das Tool soll über eine vollständige und benutzerfreundliche Dokumentation verfügen, die die Verwendung des Tools erklärt und Spielentwickler unterstützt.
6. **Testbarkeit:**
   - Das Tool soll einfach testbar sein, um sicherzustellen, dass es korrekt funktioniert und alle Anforderungen erfüllt.

QuestMapper – Festlegungen

- Einheitliche Syntax
  - Private Property beginnen mit „_“ und werden klein geschrieben
  - Public Property beginnen mit Großbuchstaben
- Programmiersprache: C#
- Wahl der IDE (ohne Analyse entschieden)
  - Visual Studio
    - offizielle IDE von Microsoft
    - Vielzahl an Funktionen
    - Immer aktuellste C# Version

QuestMapper – Anforderungen SW-Architektur

- Wartbarkeit
  - Aufgaben sollen nicht von einer „Gott-Klasse“ erledigt werden
  - Daten sollen nur an einem Ort gespeichert werden

QuestMapper – Spezifikation SW-Architektur I

- Wahl des .NET Frameworks
  - .NET Framework
  - .NET 8

- Wahl der Benutzersteuerelemente
  - Winform
  - UPW
  - WPF

- Wahl der Entwurfsmuster
  - MCV – Model-View-Controller
  - MVP – Model-View-Presenter
  - MVVM – Model-View-ViewModel
  - Passive View MVP

QuestMapper – Spezifikation SW-Architektur II

- Datenkonsistenz bei Änderungen an den Daten
  - Observer-Patter
    - Property für das Observer Object (GUI-Element)
      - Nur die Ids (QuestID, NPC_ID)
      - Kopie der Objekte (Quest, NPC)
  - Aktualisierung bei Ansichtswechsel

- Arten der Speicherung/ Laden der Benutzerdaten
  - In Dateiformat (wie JSON oder XML)
    - JSON- Format
    - XML-Format
  - In Datenbank

QuestMapper – Analyse SW-Architektur I

- Wahl des .NET Frameworks

.NET Framework	.NET 8
Nur auf Windows	Windows, Linux und MacOS
Besteht aus einer Sammlung von Bibliotheken, Laufzeitumgebungen und Tools	Modular aufgebaut und besteht aus mehreren Komponenten
Hauptsächlich unterstützt C# und Visual Basic .NET	Unterstützt mehr Programmiersprachen wie F#, C++, Python und andere
Monolithisches Deployment-Modell, bei dem alle Komponenten gemeinsam bereitgestellt werden	Modulares Deployment-Modell, bei dem Anwendungen nur die Komponenten enthalten, die sie benötigen
Ergebnis	
Nicht verwendet	Verwendet *Erstmal nur Umsetzen in Windows*

QuestMapper – Analyse SW-Architektur II

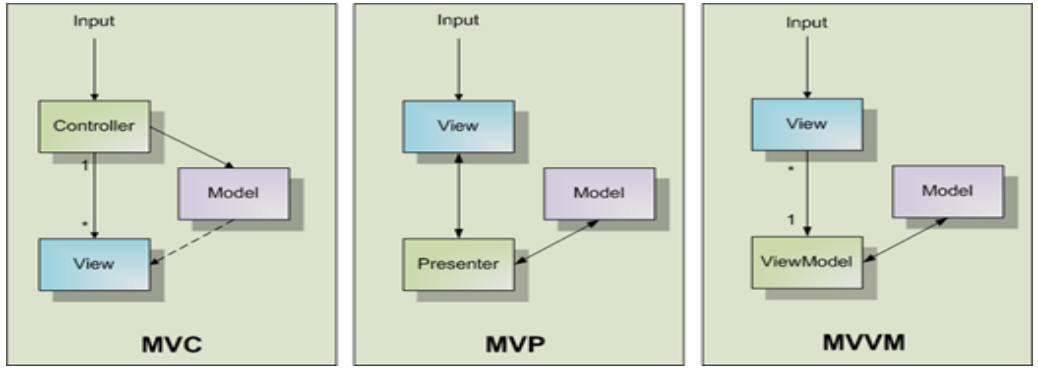
- Wahl der Benutzersteuerelemente

Winform	UWP	WPF
• Basierend auf der älteren Win32-API-Technologie • Verwendet Windows Forms zum Erstellen von Benutzeroberflächen • Einfache und schnelle Entwicklung von Desktop-Anwendungen • Unterstützung für mehrere Plattformen wie Windows, Linux und MacOS • Bietet grundlegende Steuerelemente und Funktionen für Desktop-Anwendungen	• Entwickelt für die Erstellung von plattformübergreifenden Anwendungen für Windows 10 • Verwendet XAML als Markup-Sprache für die Erstellung von Benutzeroberflächen • Bietet ein modernes und einheitliches Design • Enge Integration mit Windows 10-Plattformfunktionen wie Cortana, Live-Kacheln, Ink und mehr • Nur für die Windows-Plattform verfügbar	• Basierend auf der modernen .NET-Architektur • Verwendet XAML als Markup-Sprache für die Erstellung von Benutzeroberflächen • Bietet eine breite Palette von Steuerelementen und Funktionen für die Erstellung anspruchsvoller Desktop-Anwendungen • Unterstützung für mehrere Plattformen wie Windows, Linux und MacOS • Bietet eine flexible Design-Sprache für ansprechende und benutzerdefinierte Benutzeroberflächen zu erstellen
Nicht verwendet	Nicht verwendet	Verwendet

QuestMapper – Analyse SW-Architektur III.a

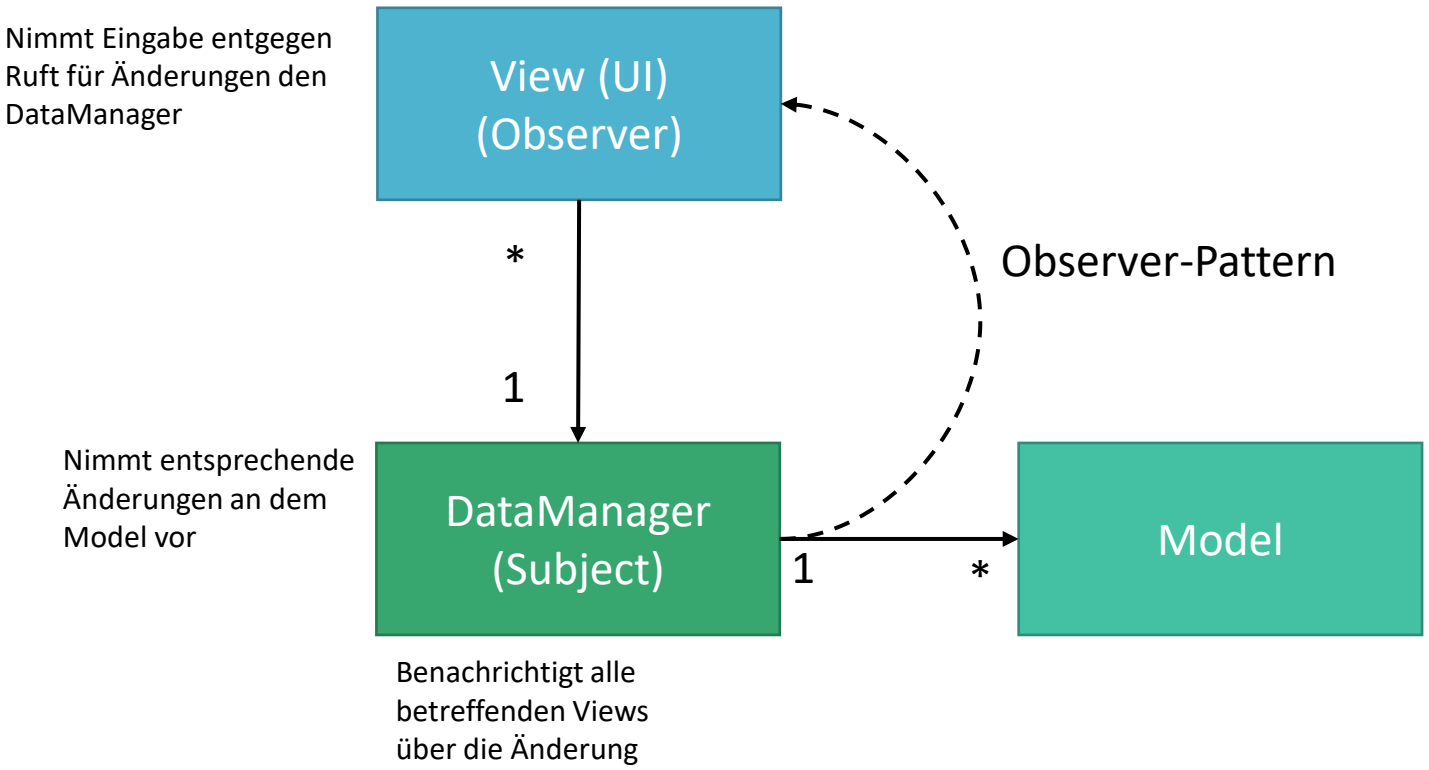
- Wahl der Entwurfsmuster

MVC		MVP		MVVM	
Vorteile	**Nachteile**	**Vorteile**	**Nachteile**	**Vorteile**	**Nachteile**
• Trennung von Geschäftslogik und UI • Ermöglicht eine einfache Wartung und Erweiterbarkeit des Codes • klare Trennung der Verantwortlichkeiten von Daten, Logik & UI • schnelle und einfache Entwicklung neuer Funktionen	• Kann bei größeren Projekten sehr komplex werden • Einige Entwickler können **Schwierigkeiten** haben, das **Konzept** von MVC **zu verstehen** und korrekt zu implementieren	• Trennung von Geschäftslogik und UI • Ermöglicht eine einfachere Testbarkeit, da Presenter von der View entkoppelt sind • Bessere Wartbarkeit durch Entkopplung Einfachere Wiederverwendbarkeit von Presentern	• Kann bei größeren Projekten sehr komplex werden • Einige Entwickler können **Schwierigkeiten** haben, das **Konzept** von MVC **zu verstehen** und korrekt zu implementieren	• klare Trennung von Geschäftslogik & UI und ViewModel • Bessere Wartbarkeit durch Entkopplung Ermöglicht eine einfache Testbarkeit • Einfachere Wiederverwendbarkeit von ViewModels	• Kann bei größeren Projekten sehr komplex werden • Einige Entwickler können **Schwierigkeiten** haben, das **Konzept** von MVC **zu verstehen** und korrekt zu implementieren
Nicht verwendet		Nicht verwendet		Nicht verwendet	



QuestMapper – Analyse SW-Architektur III.b

- Wahl der Entwurfsmuster
  - Passive View MVP



QuestMapper – Analyse SW-Architektur IV

- Datenkonsistenz bei Änderungen an den Daten

Observer-Patter		Aktualisierung bei Ansichtswechsel	
Vorteil	**Nachteil**	**Vorteil**	**Nachteil**
• Lose Kopplung zwischen den beteiligten Klassen • Erweiterbarkeit: neue Observer-Klassen können einfach hinzugefügt werden • Änderung im Subject automatischen Aktualisierung Vermeidung von unnötigen Aktualisierungen	• Komplexität: Die Implementierung des Observer-Patterns erfordert zusätzlichen Code • Verzögerungen: Eine Aktualisierung erfolgt nicht unmittelbar, sondern erst nachdem das Subject die Observer-Klassen benachrichtigt hat	• Einfache Implementierung • Sofortige Aktualisierung der Daten bei Ansichtswechsel	• Hohe Kopplung zwischen den beteiligten Klassen • Ineffizient: Aktualisierung aller Daten, auch wenn nicht benötigt • Schwierigkeiten bei der Skalierung: mehr Ansichten. desto schwieriger wird die Verwaltung der Daten und die Implementierung der Aktualisierungsfunktion
Ergebnis			
Verwendet		Nicht verwendet	

QuestMapper – Analyse SW-Architektur V

- Arten der Speicherung/ Laden der Benutzerdaten

JSON-Format		XML-Format		Datenbank	
Vorteil	**Nachteil**	**Vorteil**	**Nachteil**	**Vorteil**	**Nachteil**
• **Kleinere Dateigröße** als XML • **Schnellere** Übertragung und **Verarbeitung** von Daten • **Einfach zu lesen** und zu schreiben • Verbreitet und gut unterstützt	• **Keine** Möglichkeit, eigene **Markup-Sprachen** zu definieren • **Keine eingebaute Validierung**	• **Verbreitet** und gut unterstützt • **Lesbar** und gut verständlich für Menschen • **Einfach zu validieren** und zu parsen • Bietet die Möglichkeit, **eigene Markup-Sprachen** zu definieren	• **Größere Dateigröße** als JSON • **Langsamer** bei der Übertragung und **Verarbeitung** von Daten	• **Hohe** Skalierbarkeit und **Leistung** • **Strukturierte** Datenverwaltung • **Effiziente Suche** und Abfrage von Daten • **Integrierte Sicherheit** und **Zugriffskontrolle**	• **Komplizierter zu installieren** und zu verwalten als eine einfache Datei • **Höhere Latenzzeiten** als bei reinen Datei-basierten Systemen • **Mehr erforderliche Ressourcen**, wie Speicher und Rechenleistung.
• Bei größeren Datenmengen schneller und platzsparender als XML		• Microsoft verwendet auch XML zur Ablage • Schlecht geeignet für Betriebssystem übergreifenden Datenaustausch • Langsamer in Verarbeitung und mehr Speicherplatz benötigt als JSON		• Wird wegen der schlechten Performance für verhältnismäßig kleinen Dateien verworfen	
• Quelle: https://appmaster.io/de/blog/json-vs-xml-de ,zuletzt verwendet: 14.02.2023					
Verwendet		Nicht verwendet		Nicht verwendet	

QuestMapper – Analyse SW-Komponeten

- Welche GUI-Elemente müssen erweitert werden?
  - Panel
    - Visualisierung von Quests und NPCs
  - Linie mit Pfeil
    - Ggf. mit Text (Bedingung)
  - ListBoxItem
    - Anzeigen alles Quests und NPCs

- Eigene Questtypen sollen erstellt werden können
  - Als Standard sollen Main und Side vordefiniert werden
  - Farbe eines Panel und Text kann individuell angepasst werden
    - Eigenschaft „benutzerdefinierter Typ“

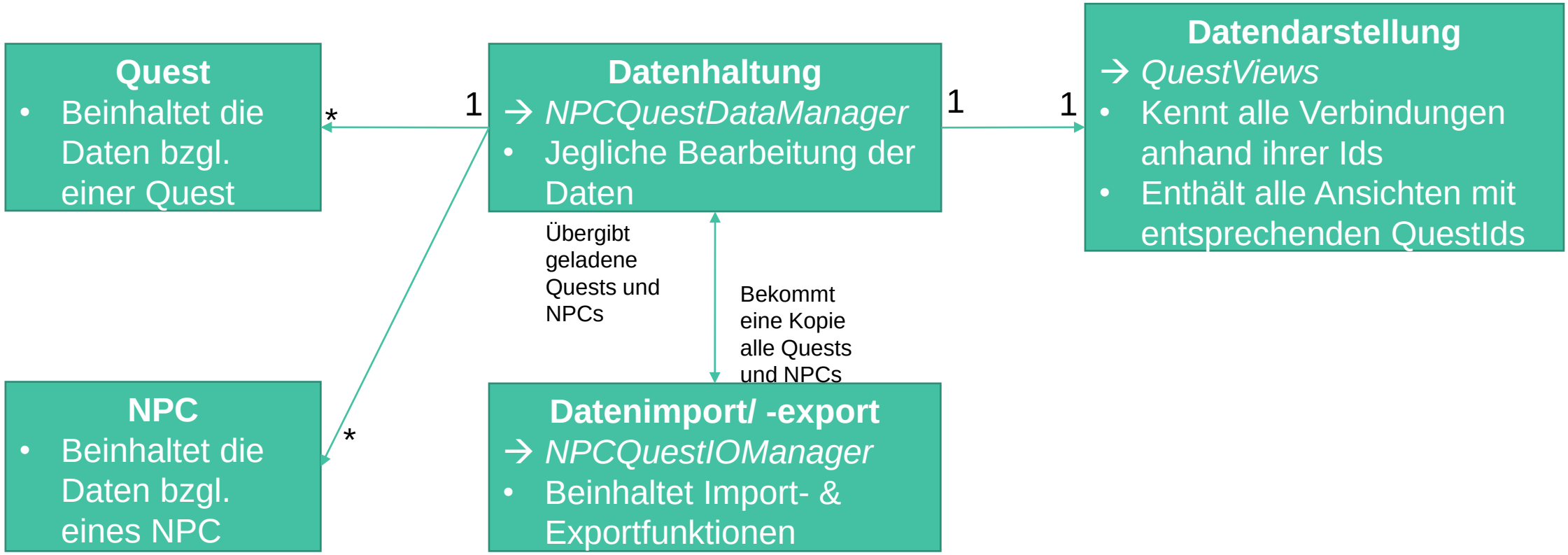
- Eine Komponente soll sich um das Erstellen und Entfernen von QuestPanels kümmern

QuestMapper – Analyse SW-Komponeten

- Getrennte Klassen mit getrennten Aufgaben
  - Datenhaltung – Model
  - Datenhaltung – Views, mittels Ids
  - Datenimport/ -export
  - GUI
  - Ggf. weitere Bibliotheken

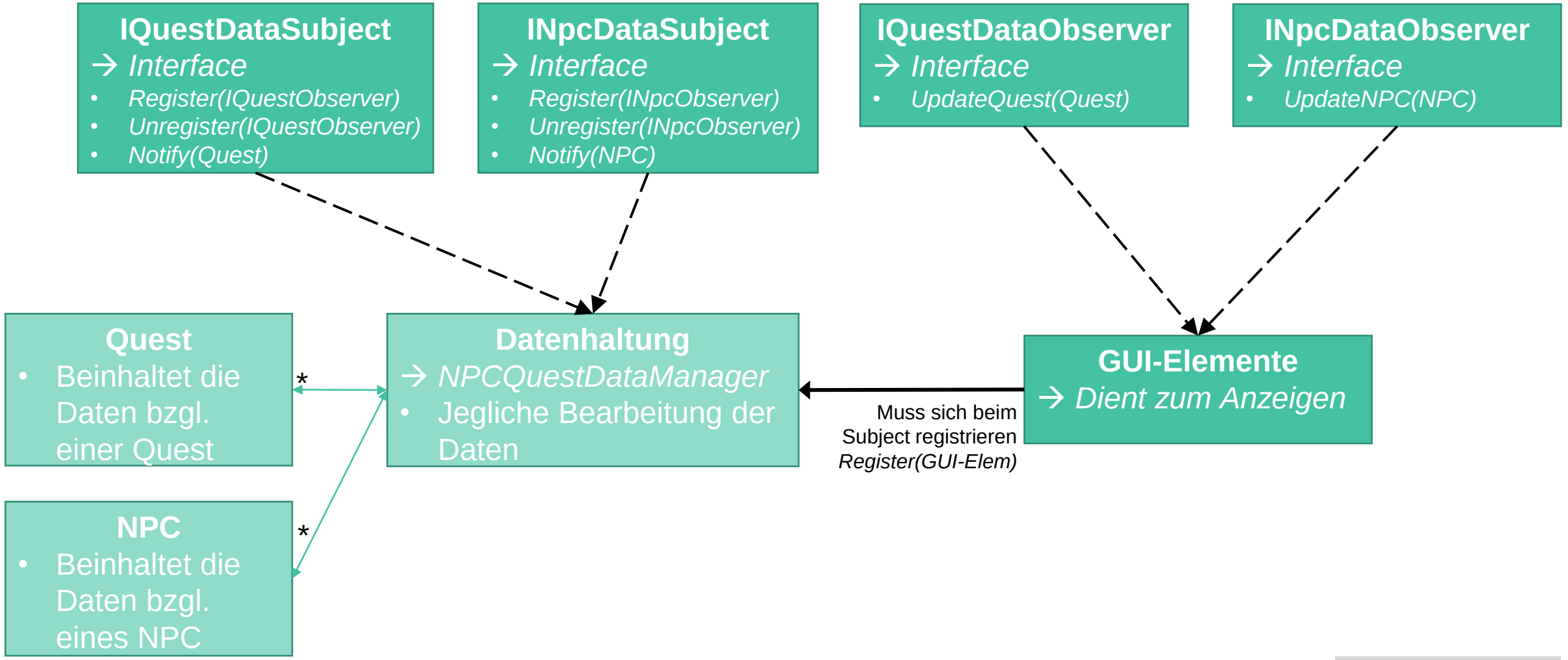
QuestMapper – Spezifikation SW-Komponeten I

- Architektur - Datenflüsse



QuestMapper – Spezifikation SW-Komponeten II

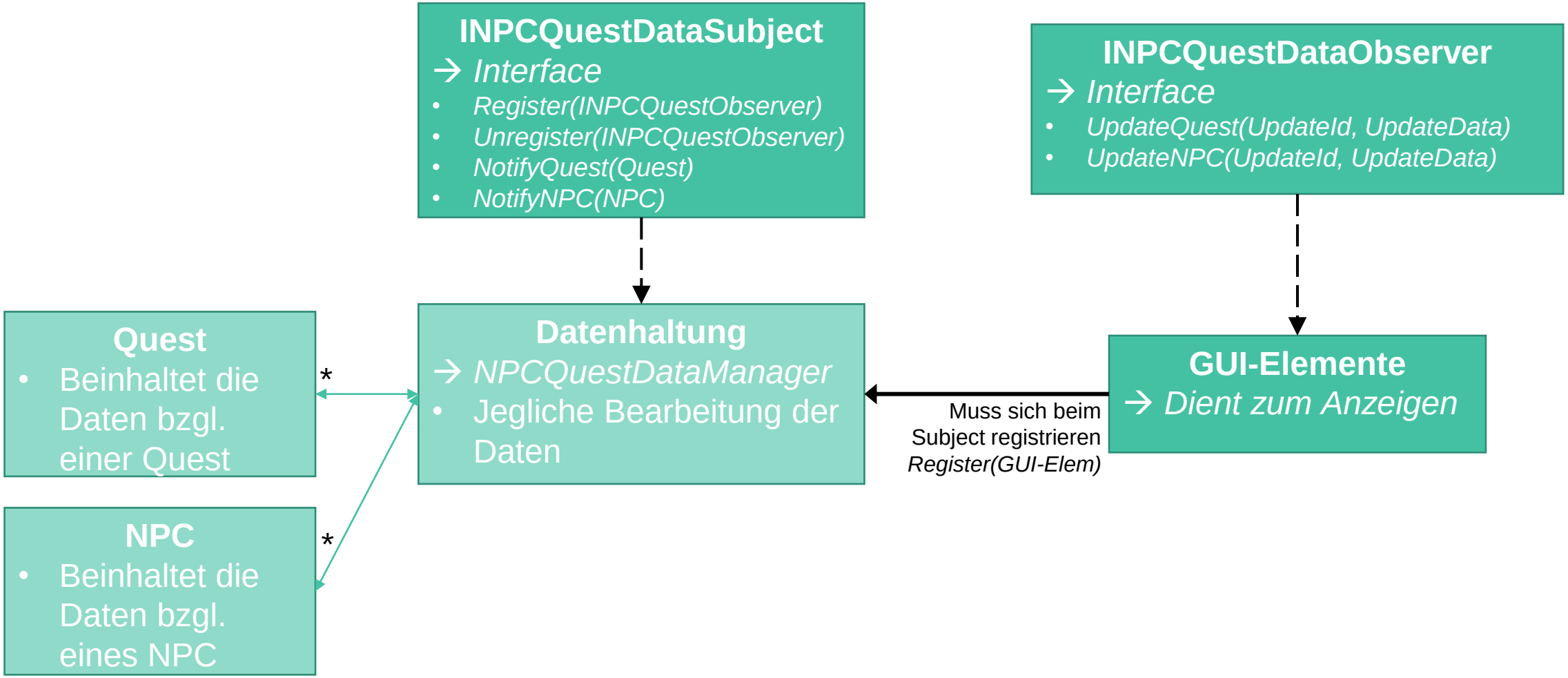
- Architektur – Datenkonsistenz via Observer-Pattern v1



Nicht verwendet

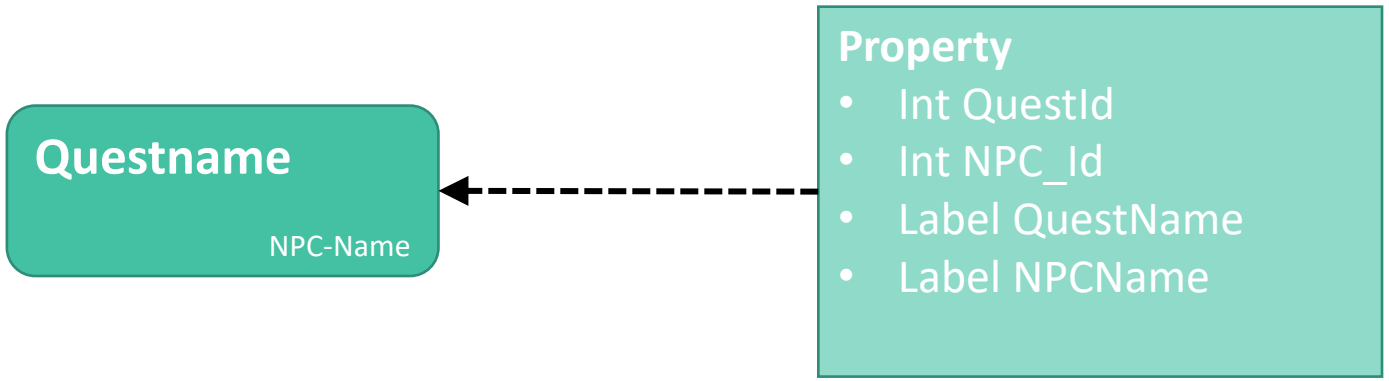
QuestMapper – Spezifikation SW-Komponeten III

- Architektur – Datenkonsistenz via Observer-Pattern v2



QuestMapper – Spezifikation SW-Komponeten IV

- Eine Komponente soll sich um das Erstellen und Entfernen von QuestPanels kümmern



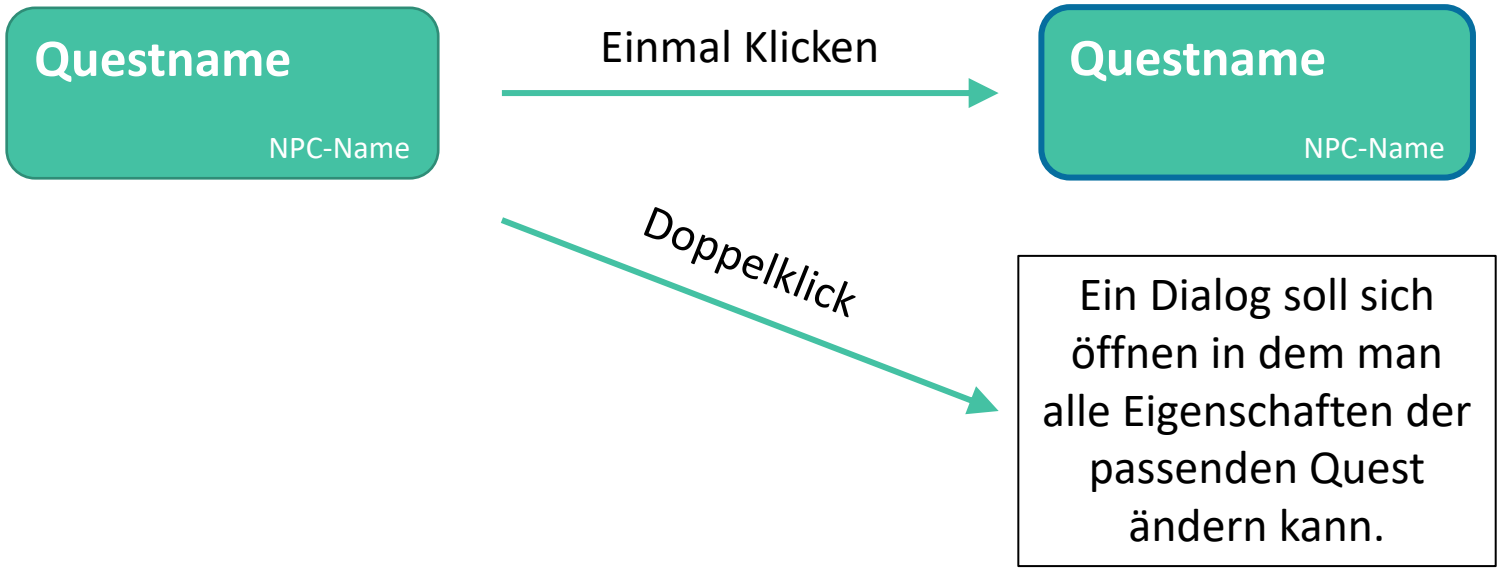
QuestMapper – Spezifikation SW-Komponeten V

- Eigene Questtypen sollen erstellt werden können

- Questtyp
  - Name
  - Farbe Text
  - Hintergrundfarbe Panel

QuestMapper – Spezifikation SW-Komponeten

- Architektur – Welche GUI-Elemente müssen erweitert werden?
  - Panel → Visualisierung von Quests und NPCs



QuestMapper – History

Änderungsdatum	Beschreibung
2023.09.02	Erstellung der PowerPoint Folien
2025.02.07	Hinzufügen einer Historie Folie zur Nachverfolgung von Änderungen, Änderung auf .Net8