Bewertungskriterien für den zweiten Anwendungsfall

Bewertung

Tests:

	Athene-70B	**CodeLlama-34B**	**DeepSeekCoder-32B**	**Llama3.1-70B**	**Qwen2-72B**	**Summe:**
Uniformität und Integration	2	3	2	3	4	**14**
Differenzierung zwischen Design und Code	3	2	3	1	3	**12**
Praktikabilität	3	2	2	3	4	**14**
Konformität	3	3	2	3	4	**15**
Grundlegende Prinzipien (Gerundetes arithmetisches Mittel der oberen Aspekte)	**3**	**3**	**2**	**3**	**4**	**15**
Vertrauenswürdigkeit	**3**	**3**	**2**	**3**	**4**	**15**
Logische Richtigkeit UML Klassendiagramm	**3**	**2**	**4**	**3**	**5**	**17**
Syntaktische Richtigkeit UML Klassendiagramm	**3**	**2**	**3**	**3**	**4**	**15**
Logische Richtigkeit UML Sequenzdiagramm	**2**	**3**	**3**	**2**	**4**	**14**
Syntaktische Richtigkeit UML Sequenzdiagramm	**3**	**3**	**2**	**2**	**3**	**13**
Logische Richtigkeit Dateibaum	**4**	**3**	**4**	**4**	**3**	**18**
SUMME	**21**	**19**	**20**	**20**	**27**	

Athene-70b:

Actor Relationship Game:

Uniformität und Integration: Das Dokument ist recht kurzgehalten. Jedoch werden alle erfragten Bestandteile generiert. Es werden alle Komponenten erkannt. Allerdings hakt die Umsetzung der nahtlosen Zusammenarbeit, was beispielsweise an den jeweiligen UML-Diagrammen zu erkennen ist.

Differenzierung zwischen Design und Code: Es wird kein Code generiert. Die fehlenden Verbindungen und Kardinalitäten in dem UML-Klassendiagramm ausgenommen, bildet das Dokument eine akzeptable Grundlage für das Implementieren des Projekts.

Praktikabilität: Es entstehen keine Widersprüche zwischen den UML-Diagrammen. Jedoch fehlt in dem Dateibaum eine Datei für die „ActorGraph“ Klasse. Es wird in dem kurzen Text auf die UML-Diagramme Bezug genommen.

Konformität: Mit der Einhaltung der Syntax und der Umsetzung von Prinzipien wie „Seperation of Concerns“, stellt dieses Design keinen Widerspruch zu den bewährten Praktiken und Konventionen dar.

Vertrauenswürdigkeit: Trotz der nachfolgenden Mängel an den UML-Diagrammen und dem Dateibaum ist die Vertrauenswürdigkeit mit Einschränkungen gegeben.

Logische Richtigkeit UML-Klassendiagramm: Es wurden alle relevanten Klassen identifiziert und für die meisten Klassen wie zum Beispiel „Actor“ und „Movie“ richtige Attribute gewählt. Lediglich der Datentyp von dem Attribut wird als Integer gewählt und unterschiedet sich somit von der Referenz. Allerdings werden die zuvor genannten Klassen nicht mit der „ActorGraph“ in Beziehung gesetzt. Letzteres stellt einen Fehler dar. Unabhängig von den fehlenden Getter und Setter Methoden, welche in dem Standard der UML keine Erwähnung finden, fehlen beispielsweise für die „ActorGraph“ Klasse viele zentrale Methoden zu dem Hinzufügen oder Entfernen von Instanzen der Klassen „Actor“ und „Movie“. Es fehlt auch die Angabe von Kardinalitäten. Positiv in Erscheinung treten dennoch die erkannten Instanzen und die aus den Eingabedaten richtig extrahierten Klassen und Attribute.

Syntaktische Richtigkeit UML-Klassendiagramm: Syntaktisch erscheint das Diagramm, bis auf die Abwesenheit von Kardinalitäten und dem Fehlen jeglicher Attribute oder Methoden in der „ActorData“ Klasse, richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Anstelle der in der Referenz gegebenen „Main“ Lifeline wird in dem von Athene generierten UML-Sequenzdiagramm der Benutzer dargestellt. Dabei handelt es sich lediglich um einen Unterschied der weder positiv noch negativ hervorsticht. Positiv ist jedoch, dass alle relevanten Lifelines erkannt wurden. Die angeführten Methoden sind in dem Klassendiagramm wiederzufinden und die Abfolge erscheint in den meisten Fällen richtig. Jedoch scheinen einige „return (…)“ Messages zu fehlen in denen die „User“ Lifeline über den Abschluss eines Methodenaufrufs erfährt. Laut dem gegebenen Sequenzdiagramm werden schließlich von der „User“ Lifeline proaktiv Methodenaufrufe getätigt ohne das durch eine „return“ Message bekannt ist, ob die zuvor nötigen Verarbeitungsschritte bereits stattgefunden haben oder nicht. Zuletzt fehlt eine „return“ Message für die „User“ Lifeline, um diese über den erfolgreichen Abschluss der Generierung des Graphs zu informieren.

Syntaktische Richtigkeit UML Sequenzdiagramm: Das Diagramm erscheint, bis auf das Fehlen einiger „return“ Messages, syntaktisch richtig zu sein.

Logische Richtigkeit Dateibaum: In dem Dateibaum fehlt eine Datei in der die „ActorGraph“ zu implementieren wäre. Abgesehen davon erscheint der Dateibaum richtig. Eine „build.gradle“ Datei ist ebenso vorhanden. Letztere wurde jedoch aus dem Anforderungsdokument übernommen. Es wurde aber die Implementierung von Tests bedacht und dementsprechend auch ein dafür vorgesehener Ordner und Dateien angelegt.

Hone:

Uniformität und Integration: Ist teilweise gegeben. Es existieren leichte Widersprüche zwischen dem UML-Sequenzdiagramm und dem UML-Klassendiagramm.

Differenzierung zwischen Design und Code: Es wird kein Code, sondern eine solide Vorlage für die Umsetzung des Projekts geliefert.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen. Jedoch ist der Entwurf modular und effizient.

Konformität: Ist gegeben.

Vertrauenswürdigkeit: Aufgrund der Fehler in dem UML-Sequenzdiagramm nur teilweise erfüllt.

Logische Richtigkeit UML-Klassendiagramm: Die Klassen sind vollständig und sinnvoll gewählt.

Syntaktische Richtigkeit UML-Klassendiagramm: Diagramm ist syntaktisch richtig. So hätten Kardinalitäten angegeben werden können. Allerdings ist durch den Projektrahmen bereits klar, dass in jeder Beziehung eine Kardinalität von „1….1“ besteht.

Logische Richtigkeit UML-Sequenzdiagramm: Ablauf wird sinnvoll dargestellt. Jedoch werden mit „extract column names and data rows“ teilweise Methodenaufrufe in Messages beschrieben die das UML-Klassendiagramm als solches nicht hergibt.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Syntaktische Richtigkeit ist gegeben. Es mussten zudem aus dem generierten PlantUML-Code die „notes“ entfernt werden, da ansonsten das Diagramm aufgrund von Syntaxfehlern nicht angezeigt werden konnte.

Logische Richtigkeit Dateibaum: Ist vollständig.

Hybrid Images:

Uniformität und Integration: Aufgrund des UML-Klassendiagramms nur bedingt gegeben.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen. Design der Komponenten etwas zu kleinteilig.

Konformität: Ist aufgrund der Fehler in den Diagrammen nur bedingt gegeben.

Vertrauenswürdigkeit: Aufgrund der Fehler in den Diagrammen nicht gänzlich kohärent.

Logische Richtigkeit UML-Klassendiagramm: Das Klassendiagramm enthält einige Fehler. So sind zum Beispiel die Klassen „LowPassFilter“, „GaussianBlur“, und „HighPassFilter“ ohne jegliche Beziehung zu den anderen Klassen. Es fehlt die Beziehung zu der abstrakten Klasse „Filter“, da alle zuvor genannten Klassen die „apply“ Methode vererben. Zudem ist es fraglich, ob die drei zuvor genannten Klassen nicht vielmehr Methoden einer nicht abstrakten „Filter“ Klasse sein sollten.

Syntaktische Richtigkeit UML-Klassendiagramm: Dass die Klassen „LowPassFilter“, „GaussianBlur“, und „HighPassFilter“ in dem Diagramm ohne jegliche Beziehungen existieren, stellt einen Fehler dar. Ansonsten syntaktisch richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Erscheint fehlerhaft, da kein richtiger Konversationsfluss dargestellt wird. Als Beispiel kann die „return_hybrid_image()“ Message angeführt werden, welche auf eine nicht vorhandene Lifeline verweist.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Aufgrund des nicht gegebenen durchgängigen Konversationsfluss und der letzten Message, welche auf eine nicht vorhandene Lifeline verweist, fehlerhaft.

Logische Richtigkeit Dateibaum: Es fehlen die Dateien für die Implementierung der „LowPass“ und „HighPass“ Filter Klassen. Zwar wird konkret angegeben, dass die beiden zuvor genannten Klassen in der „filter.py“ Datei zusammen mit der abstrakten „Filter“ Klasse implementiert werden. Jedoch erscheint dieses Vorgehen fehlerhaft und nicht entsprechend der weitläufig akzeptierten Best-Practices.

Login Registration:

Uniformität und Integration: Ist bis auf einige kleine Fehler erfüllt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird aufgrund der fehlenden Strukturen in dem Dateibaum keine umfassende Grundlage für eine spätere Implementierung geliefert.

Konformität: Ist nur bedingt erfüllt.

Vertrauenswürdigkeit: Aufgrund der vielen fehlenden Strukturen in dem Dateibaum nicht erfüllt.

Logische Richtigkeit UML-Klassendiagramm: Klassendiagramm ist vollständig. Methoden und Attribute erscheinen richtig. Es fehlt jedoch eine Beziehung zwischen der „User“ Klasse und der Klasse „Store“. Ebenso fehlt die Beschriftung der Beziehungen und die Kardinalitäten sind ebenfalls nicht vorhanden.

Syntaktische Richtigkeit UML-Klassendiagramm: Es musste der generierte PlantUML-Code angepasst werden, da ansonsten kein UML-Klassendiagramm hätte dargestellt werden können.

Logische Richtigkeit UML-Sequenzdiagramm: Der Kommunikationsverlauf erscheint sinnvoll.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Es werden drei verschiedene Kommunikationsverläufe dargestellt. Jedoch werden hierzu nicht die von der OMG vorgesehenen „alternatives“ genutzt. Ferner musste auch in diesem Fall der PlantUML-Code angepasst werden, um das UML-Sequenzdiagramm überhaupt darstellen zu können.

Logische Richtigkeit Dateibaum: Dateibaum ist nicht vollständig. Es fehlen viele Strukturen wie zum Beispiel Ordner für Unittests und Akzeptanztests.

Readtime:

Uniformität und Integration: Ist erfüllt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird zwar kein Bezug auf die Diagramme genommen, aber das Gesamtkonzept in Verbindung mit den Diagrammen zeigt eine klare und logische Struktur auf.

Konformität: Ist erfüllt.

Vertrauenswürdigkeit: Die Anforderungen der Anforderungsspezifikation werden deutlich erkennbar und richtig umgesetzt.

Logische Richtigkeit UML-Klassendiagramm: Es wird ein sinnvolles UML-Klassendiagramm erstellt. Die Beziehungen enthalten Beschriftungen. Es fehlen lediglich die Kardinalitäten. Auch die vorhandenen Attribute und Methoden werden sinnvoll verknüpft.

Syntaktische Richtigkeit UML-Klassendiagramm: Unter Ausnahme der fehlenden Kardinalitäten ist das UML-Klassendiagramm richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Es werden hier die in der OMG vorgesehenen „alternatives“ zu der Fallunterscheidung richtig eingesetzt (OMG, S. 583). Das UML-Sequenzdiagramm ist schlüssig und enthält keine Fehler.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Das UML-Sequenzdiagramm ist syntaktisch richtig.

Logische Richtigkeit Dateibaum: Der Dateibaum ist vollständig.

CodeLlama-34B:

Actor Relationship Game:

Uniformität und Integration: Es werden, bis auf die „GraphCreation“ Klasse in dem UML-Klassendiagramm, alle Komponenten integriert. Selbiges gilt auch für den Dateibaum. Das Sequenzdiagramm weist jedoch Schwächen auf und stellt aufgrund der vielen rekursiven und teilweise zu wage definierten Messages die nahtlose Zusammenarbeit aller Komponenten in Frage.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird kein weiterführender Text generiert und somit kann auch kein weitergehender Bezug zu den generierten UML-Diagrammen stattfinden.

Konformität: Gemessen an dem Dateibaum und den UML-Diagrammen, ist bis auf die nachfolgend formulierten Schwächen, die Konformität gegeben.

Vertrauenswürdigkeit: Die strikte Übereinstimmung mit der Anforderungsspezifikation ist anhand des Dateibaums und des UML-Klassendiagramms erkennbar. Letzteres gilt jedoch nicht für das UML-Sequenzdiagramm.

Logische Richtigkeit UML-Klassendiagramm: Es werden alle Klassen erkannt. Gleiches gilt auch für die Attribute der Klassen. Es fehlen Kardinalitäten. Jedoch werden die Beziehungen zwischen den Klassen nahezu richtig erkannt. Lediglich die „GraphCreation“ Klasse weist keine Verbindung zu der „ActorGraph“ Klasse auf. Die ist falsch. Ebenso fehlen noch einige Methoden in der „ActorGraph“ Klasse zu dem Hinzufügen und Löschen von Instanzen der Klassen „Actor“ du „Movie“.

Syntaktische Richtigkeit UML-Klassendiagramm: Syntaktisch erscheint das Diagramm bis auf das Fehlen jeglicher Verbindungen zu der „GraphCreation“ Klasse und fehlenden Kardinaltäten richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Das Sequenzdiagramm hat einige Schwächen. So fehlt eine klare „Main“ oder „User“ Lifeline. Zudem erscheinen die vielen rekursiven Messages wenig nachvollziehbar. Zudem werden mit „fetch“ teilweise Methoden angegeben die nicht existieren beziehungsweise zu ungenau beschrieben sind.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Bis auf die übermäßig vielen rekursiven Messages und der fehlenden „Main“ oder „User“ Lifeline ist das UML-Sequenzdiagramm syntaktisch richtig.

Logische Richtigkeit Dateibaum: Dateibaum erscheint richtig. Es wird die Erstellung von Akzeptanztests und Unittests berücksichtigt und passende Ordner und Dateien aufgebaut. Es fehlen auch keine Dateien in dem sonstigen Dateibaum.

Hone:

Uniformität und Integration: Durch das UML-Klassendiagramm nicht wirklich gegeben.

Differenzierung zwischen Design und Code: Es wird kein Code generiert. Mitnichten wird eine gute Grundlage für das Coding geliefert. Aufgrund der Fehler in den jeweiligen UML-Diagrammen kann nur der Dateibaum uneingeschränkt weiterverwendet werden.

Praktikabilität: Es wird kein Bezug auf die Diagramme genommen. Der Entwurf ist dabei fehlerhaft.

Konformität: Ist nur bedingt gegeben.

Vertrauenswürdigkeit: Ist nur mit starken Einschränkungen gegeben

Logische Richtigkeit UML-Klassendiagramm: Ist nur bedingt gegeben. Es existieren keine Verbindungen zwischen den dargestellten Klassen. Die Klassen, Attribute und Methoden selbst ergeben jedoch Sinn. Es kann zusätzlich angemerkt werden, dass potenziell ein Duplikat in einer zu implementierenden Funktionalität gegeben sein könnte. Dies ist zwischen der Methode „parse_csv“ der Klasse „CSVParser“ und der Methode „open_csv“ der Klasse „FileManager“ gegeben. Jedoch ist letzteres von der konkreten Implementierung der zuvor genannten Klassen und Methoden abhängig, weshalb an diesem Punkt nur die Auffälligkeit erwähnt werden kann.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist aufgrund der fehlenden Beziehungen zwischen den Klassen nicht gegeben.

Logische Richtigkeit UML-Sequenzdiagramm: Es fehlt der initiale Aufruf der Klasse „FileManager“ zu dem Öffnen der zu verarbeitenden CSV-Datei. Es wird dann jedoch eine fließende Konversation dargestellt, welche logisch richtig erscheint.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Ist gegeben.

Logische Richtigkeit Dateibaum: Der Dateibaum ist vollständig und richtig. Es wird ebenso an Unittests gedacht.

Hybrid Images:

Uniformität und Integration: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Differenzierung zwischen Design und Code: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Praktikabilität: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Konformität: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Vertrauenswürdigkeit: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Logische Richtigkeit UML-Klassendiagramm: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Syntaktische Richtigkeit UML-Klassendiagramm: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Logische Richtigkeit UML-Sequenzdiagramm: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Logische Richtigkeit Dateibaum: Es wurde nur die Definition des Begriffs „Cross-Correlation“ generiert.

Login Registration:

Uniformität und Integration: Es wurde nichts generiert.

Differenzierung zwischen Design und Code: Es wurde nichts generiert.

Praktikabilität: Es wurde nichts generiert.

Konformität: Es wurde nichts generiert.

Vertrauenswürdigkeit: Es wurde nichts generiert.

Logische Richtigkeit UML-Klassendiagramm: Es wurde nichts generiert.

Syntaktische Richtigkeit UML-Klassendiagramm: Es wurde nichts generiert.

Logische Richtigkeit UML-Sequenzdiagramm: Es wurde nichts generiert.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Es wurde nichts generiert.

Logische Richtigkeit Dateibaum: Es wurde nichts generiert.

Readtime:

Uniformität und Integration: Durch das UML-Klassendiagramm und den Dateibaum erfüllt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen.

Konformität: Durch das UML-Sequenzdiagramm nur bedingt erfüllt.

Vertrauenswürdigkeit: Es sind klar die Anforderungen der Anforderungsspezifikation erkennbar. Durch das Sequenzdiagramm ist die Vertrauenswürdigkeit nur bedingt erfüllt.

Logische Richtigkeit UML-Klassendiagramm: Das Klassendiagramm erscheint sinnvoll und enthält alle notwendigen Klassen, Attribute und Methoden. Es fehlen die Kardinalitäten.

Syntaktische Richtigkeit UML-Klassendiagramm: Abgesehen von dem Fehlen der Kardinalitäten syntaktisch richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Es fehlen die „alternatives“ für die jeweiligen Fälle, dass es sich bei dem Dokument um HTML, Markdown oder Text handelt. Dementsprechend ungenau sind die Beschriftungen der Messages.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Das UML-Sequenzdiagramm ist syntaktisch richtig.

Logische Richtigkeit Dateibaum: Ist vollständig.

DeepSeekCoder-32B:

Actor Relationship Game:

Uniformität und Integration: Einheitlicher Stil aufgrund des UML-Sequenzdiagramms nicht gegeben.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: In dem gegebenen Text wird lediglich auf den Dateibaum Bezug genommen.

Konformität: Aufgrund des UML-Sequenzdiagramms kann nicht von einer Einhaltung der allgemein akzeptierten Konventionen gesprochen werden.

Vertrauenswürdigkeit: Anhand der vollständigen Erkennung aller Klassen in dem UML-Klassendiagramm und dem nahezu vollständigen Dateibaum, ist die Anforderungsspezifikation in dem Softwaredesign wiedererkennbar. Jedoch tut dem das UML-Sequenzdiagramm einen Abbruch.

Logische Richtigkeit UML-Klassendiagramm: Es werden alle Klassen erkannt. Jedoch fehlen die Kardinalitäten. Ebenso fehlen Beziehungen zwischen der „TMDBApi“ Klasse und der „GraphCreation“ Klasse. Selbiges gilt auch für die Klassen „Actor“ und „Movie“ in Bezug auf die Klasse „ActorGraph“. Zusätzlich existiert keine Klasse in der sowohl Attribute als auch Methoden gegeben sind.

Syntaktische Richtigkeit UML-Klassendiagramm: Syntaktisch erscheint das Klassendiagramm richtig. Es existiert keine Klasse in der keine Verbindung gegeben ist.

Logische Richtigkeit UML-Sequenzdiagramm: Nicht gegeben. Lifelines sind lediglich nach Buchstaben benannt, wodurch nicht nachvollziehbar ist, welche Softwarekomponenten miteinander über die dargestellten Messages kommunizieren sollen.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Aufgrund der Benennung der Lifelines nicht gegeben. Erschwerend kommen Duplikate in der Benennung der Lifelines und vier ungenutzte Lifelines hinzu.

Logische Richtigkeit Dateibaum: Ist bis auf die Erstellung nötiger Strukturen für potenzielle Akzeptanztests vollständig.

Hone:

Uniformität und Integration: Es wurde nichts generiert.

Differenzierung zwischen Design und Code: Es wurde nichts generiert.

Praktikabilität: Es wurde nichts generiert.

Konformität: Es wurde nichts generiert.

Vertrauenswürdigkeit: Es wurde nichts generiert.

Logische Richtigkeit UML-Klassendiagramm: Es wurde nichts generiert.

Syntaktische Richtigkeit UML-Klassendiagramm: Es wurde nichts generiert.

Logische Richtigkeit UML-Sequenzdiagramm: Es wurde nichts generiert.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Es wurde nichts generiert.

Logische Richtigkeit Dateibaum: Es wurde nichts generiert.

Hybrid Images:

Uniformität und Integration: Ist gegeben, da die Diagramme, bis auf das Fehlen des „Filter“ Interfaces in dem Dateibaum, alle Komponenten richtig einbindet.

Differenzierung zwischen Design und Code: Es wird kein Code generiert, sondern eine Basis für die Implementierung dieses Projekts geliefert.

Praktikabilität: Es wird kein Bezug auf die Diagramme genommen. Dennoch ist das Design aufgrund der ausführlichen Diagramme und Begriffserklärungen sehr gut lesbar.

Konformität: Ist gegeben.

Vertrauenswürdigkeit: Mit der Ausnahme des Dateibaums erfüllt.

Logische Richtigkeit UML-Klassendiagramm: Für ein trivial erscheinendes Projekt potenziell zu kleinteilige Softwarekomponenten. Jedoch erscheinen die Interfaces Klassen und Attribute sinnvoll.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist erfüllt. Es sind auch Kardinalitäten und die Beschriftung der Beziehungen vorhanden.

Logische Richtigkeit UML-Sequenzdiagramm: Es ist eine „Main“ Lifeline gegeben. Konversationsverlauf ist fließend und erscheint basierend auf das UML-Klassendiagramm logisch.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Ist erfüllt.

Logische Richtigkeit Dateibaum: Es fehlt eine Datei in der das in dem UML-Klassendiagramm aufgeführte "Filter" Interface implementiert wird. Zudem fehlen Ordnerstrukturen in denen Unittests und Akzeptanztests hinterlegt werden können. Es wird eine richtige „requirements.txt“ generiert.

Login Registration:

Uniformität und Integration: Ist aufgrund ausführlicher UML-Diagramme gegeben. Lediglich der Dateibaum schränkt diesen Aspekt etwas ein.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es werden die Diagramme sinnvoll mit dem Hinweis referenziert, dass es sich um die PlantUML-Syntax handelt. Es fehlt aber jegliche Erklärung zu den Dateien in dem Dateibaum.

Konformität: Ist weitestgehend erfüllt.

Vertrauenswürdigkeit: Aufgrund der guten UML-Diagramme und trotz des fehlerhaften Dateibaums, ist dieser Faktor eingeschränkt erfüllt.

Logische Richtigkeit UML-Klassendiagramm: Ist vollständig mit Kardinalitäten, Beziehungen, sinnvollen Klassen, Methoden und Attributen.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist erfüllt. Es könnte lediglich die gegebene 1zu1 Beziehungen zwischen der „User“ und „UserService“ Klasse angeführt werden, welcher der bereits zwischen diesen Klassen gegebenen „1zuN“ Beziehung widerspricht.

Logische Richtigkeit UML-Sequenzdiagramm: Erscheint richtig. Jedoch wird nicht der Fall inkludiert, dass der Benutzer bereits ein bestehendes Konto besitzt. Ansonsten richtig.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Syntaktisch richtig.

Logische Richtigkeit Dateibaum: Es fehlen Ordnerstrukturen für spätere Unittests. Ebenso fehlt eine App.vue Komponente, in der die weiteren Vue Komponenten eingebunden werden können. Es geht aus dem Dateibaum auch nicht hervor, wo der für das Programm elementare Store implementiert wird.

Readtime:

Uniformität und Integration: Ist aufgrund dessen, dass dieses Design aus einer einzigen und leicht verständlichen Komponente besteht erfüllt. Aufgrund des zusätzlich beigefügten Paketdiagramms wird dieser Aspekt besonders erfüllt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird zusätzlich ein Paketdiagramm generiert. Jedoch wird auf keines der generierten UML-Diagramme Bezug genommen.

Konformität: Ist erfüllt.

Vertrauenswürdigkeit: Ist aufgrund der einfachen Architektur aus nur einer Komponente erfüllt. Lediglich das UML-Sequenzdiagramm behindert die Konsistenz.

Logische Richtigkeit UML-Klassendiagramm: Es ist eine Klasse vorhanden, welche sinnvoll erscheint.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist erfüllt.

Logische Richtigkeit UML-Sequenzdiagramm: Es sind mit „HTMLContent“, „Markdown Content“ und „Plain Text Content“ drei ungenutzte Lifelines vorhanden. Andernfalls ist die Kommunikation zwischen der „User“ Lifelane und der „ReadTime“ Lifelane sinnvoll.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Ist aufgrund der drei ungenutzten aber dennoch präsenten Lifelines nicht erfüllt.

Logische Richtigkeit Dateibaum: Der Dateibaum ist vollständig. Es wird zusätzlich noch ein UML-Paketdiagramm generiert. Mit dem „docs“ Ordner wird auch das Verwahren der erstellten UML-Diagramme berücksichtigt.

Llama3.1 70b:

Actor Relationship Game:

Uniformität und Integration: Ist bis auf den Fehler der teilweise fehlenden Beziehungen in dem UML-Klassendiagramm gegeben.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird in dem generierten Text nur auf den Dateibaum eingegangen. Jedoch ist der Entwurf modular und effizient.

Konformität: Einhaltung der von der Open-Source-Gemeinschaft akzeptierten Konventionen und Praktiken sind weitestgehend gegeben.

Vertrauenswürdigkeit: Aufgrund der teils fehlerhaften Diagramme nur bedingt gegeben.

Logische Richtigkeit UML-Klassendiagramm: Es werden alle Klassen erkannt. Ebenso sind viele wichtige Methoden und Attribute vorhanden. So sind in diesem UML-Klassendiagramm für die Klasse „ActorGraph“ Methoden zu dem Hinzufügen und Löschen von Instanzen der Klassen „Actor“ und „Movie“ vorhanden. Allerdings fehlt in dem Diagramm eine Beziehung zwischen den zuvor genannten Klassen und ebenso die Angabe von Kardinalitäten.

Syntaktische Richtigkeit UML-Klassendiagramm: Bis auf die Angabe von Kardinalitäten richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Es fehlt eine „Main“ oder „User“ Lifeline. Zusätzlich erscheint die Message über den Aufruf der „deserializeGraph()“ weniger verständlich. Der Aufruf erfolgt von der „ActorGraphUtil“ Klasse, ohne, dass durch eine Message bekannt ist, dass der Aufruf der „serializeGraph()“ Methode im Vorfeld erfolgt ist. Gleiches gilt für die Message mit dem Aufruf der „findConnections()“ Methode. Es fehlt somit ein fließender Kommunikationsfluss.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Bis auf einige kleinere Fehler richtig.

Logische Richtigkeit Dateibaum: Dateibaum ist vollständig. Es werden ebenso die Unittests und Akzeptanztests berücksichtig.

Hone:

Uniformität und Integration: Integration der unterschiedlichen Komponenten ist gegeben. Jedoch ist die nahtlose Zusammenarbeit aufgrund des fehlerhaften UML-Sequenzdiagramms nicht sichergestellt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen. Jedoch ist das Dokument effizient und modular gehalten.

Konformität: Syntaktisch ist dies in jedem Diagramm der Fall. Logisch jedoch nicht. Das fehlerhafte UML-Sequenzdiagramm und die teilweise vorhandenen Redundanzen in dem Dateibaum schränken die Konformität ein.

Vertrauenswürdigkeit: Aufgrund der genannten Fehler in den Diagrammen nicht gegeben.

Logische Richtigkeit UML-Klassendiagramm: Die erstellten Klassen erscheinen sinnvoll und sind zudem auch vollständig. Es fehlen bei den Beziehungen die Kardinalitäten, welche aber aufgrund des Projektkontexts auch weniger relevant sind. Es fehlt jedoch auch die Beschriftung der einzelnen Beziehungen.

Syntaktische Richtigkeit UML-Klassendiagramm: Bis auf das Fehlen der Beschriftungen und Kardinalitäten der Beziehungen syntaktisch richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Die dargestellten Aufrufe erscheinen bedingt sinnvoll. So wird kein fließender Kommunikationsfluss geboten. Erkennbar ist dies anhand der „run_conversion“ Message, welche erfolgt nachdem die „generate_json()“ Message und somit die eigentliche Umwandlung der gegebenen CSV-Datei in eine adäquate JSON-Datei erfolgt ist. Auch wenn erstmalig die sogenannten „Coregion Areas“ eingesetzt werden, geschieht der Einsatz nicht richtig. So drückt eine „Coregion Area“ gemäß OMG aus, dass die darin enthaltenen Messages in ihrer Reihenfolge beliebig variieren können. (OMG, S.586 und OMG, S.597) Bei genauerer Betrachtung des Diagramms ist dies jedoch nicht der Fall. So kann „get_column_names()“ nicht vor „parse_csv()“ aufgerufen werden, da die Datei zu diesem Zeitpunkt noch gar nicht verarbeitet worden ist. Somit besteht eine temporale Abhängigkeit zwischen den zuvor genannten Aufrufen und gleichermaßen auch Messages.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Syntaktisch ist das Diagramm weitestgehend richtig. Es werden lediglich die „Coregion Areas“ aus semantischer Sicht nicht richtig eingesetzt. Dies tut der syntaktischen Richtigkeit des Diagramms jedoch keinen Abbruch.

Logische Richtigkeit Dateibaum: Der Dateibaum enthält in Bezug auf die Tests einige redundante Dateien und Ordner. Abgesehen davon ist der Dateibaum jedoch vollständig und richtig.

Hybrid Images:

Uniformität und Integration: Ist aufgrund fehlender Beziehungen in dem UML-Klassendiagramm und der teilweise falschen Kommunikation innerhalb des UML-Sequenzdiagramms nicht gegeben.

Differenzierung zwischen Design und Code: Es wird kein Code generiert. Es wird aber auch keine fehlerfreie Basis für eine gute Implementierung geschaffen.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen.

Konformität: Ist nur bedingt erfüllt.

Vertrauenswürdigkeit: Aufgrund der Fehler in den UML-Diagrammen nur mit starken Einschränkungen vertrauenswürdig.

Logische Richtigkeit UML-Klassendiagramm: Klassendiagramm ist fehlerhaft, da zwar passende Klassen generiert, jedoch keine Beziehungen zwischen ihnen dargestellt werden. Bei den Methoden ist zudem auffällig, dass bei einer Implementierung dieses Designs Redundanzen auftreten könnten. Ein Beispiel bildet die „Filter“ Klasse mit einer „apply“ Methode und die jeweiligen Filtermethoden in der „ImageProcessor“ Klasse. Nicht nur die Namen, sondern auch die Typen der Ein- und Ausgabeparameter deuten sogar auf einen Widerspruch hin, da die Filterklasse entweder die in der „ImageProcessor“ Klasse genannten Methoden implementieren müsste oder weitere Klassen gegeben sein müssten, welche aus der „Filter“ Klasse erben und somit die unterschiedlichen Filter anwenden.

Syntaktische Richtigkeit UML-Klassendiagramm: Es fehlen jegliche Beziehungen und somit auch Kardinalitäten und Beschriftungen von Beziehungen zwischen den Klassen.

Logische Richtigkeit UML-Sequenzdiagramm: Es ist kein vollständig fließender Kommunikationsverlauf gegeben. Grund stellt die Message „create_hybrid_image“ der „HybridImageCreator“ Klasse dar, welche nicht über eine Message darüber informiert wird, welche Bilder für das Generieren des hybriden Bilds genutzt werden sollen. Andernfalls erscheint die Konversation logisch.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Bis auf den Fehler in der fließenden Kommunikation hinsichtlich der „HybridImageCreator“ Klasse, richtig.

Logische Richtigkeit Dateibaum: Ist erfüllt. Es wurde auch an den Aufbau von Strukturen für Unittests gedacht. „requirements.txt“ wird generiert und ist richtig.

Login Registration:

Uniformität und Integration: Es wurde nur Code generiert.

Differenzierung zwischen Design und Code: Es wurde nur Code generiert.

Praktikabilität: Es wurde nur Code generiert.

Konformität: Es wurde nur Code generiert.

Vertrauenswürdigkeit: Es wurde nur Code generiert.

Logische Richtigkeit-UML Klassendiagramm: Es wurde nur Code generiert.

Syntaktische Richtigkeit-UML Klassendiagramm: Es wurde nur Code generiert.

Logische Richtigkeit-UML Sequenzdiagramm: Es wurde nur Code generiert.

Syntaktische Richtigkeit-UML Sequenzdiagramm: Es wurde nur Code generiert.

Logische Richtigkeit Dateibaum: Es wurde nur Code generiert.

Readtime:

Uniformität und Integration: Ist aufgrund der gegebenen Diagramme erfüllt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen, dabei wird aber ein modulares Design erstellt in dem, abgesehen von kleineren Fehlern in dem UML-Sequenzdiagramm, alle nötigen Informationen vertreten sind.

Konformität: Ist unter Ausnahme des UML-Sequenzdiagramms erfüllt.

Vertrauenswürdigkeit: Die Anforderungen der Anforderungsspezifikation werden nahtlos in das gegebene Softwaredesign eingepflegt.

Logische Richtigkeit UML-Klassendiagramm: Klassen sind klar nach dem Prinzip der „Seperation of Concerns“ aufgeteilt. Methoden und Beziehungen ergeben Sinn. Es fehlen jedoch die Beschriftung der Beziehungen und die Kardinalitäten, welche aufgrund des Projektkontexts aber alle bei der Kardinalität „1zu1“ sein sollten.

Syntaktische Richtigkeit UML-Klassendiagramm: Bis auf das Fehlen der Beschriftungen und Kardinalitäten an den Beziehungen, ist das UML-Klassendiagramm syntaktisch richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Es werden die möglichen drei Fälle richtig berücksichtigt. Allerdings geschieht dies nicht in Form der von der OMG vorgesehenen „alternatives“.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Es werden nicht die zu diesem Beispiel passenden „alternatives“ genutzt. Abgesehen davon aber syntaktisch richtig.

Logische Richtigkeit Dateibaum: Die mitgelieferte „requirements.txt“ ist richtig. Der Dateibaum ebenso.

Qwen2 72B:

Actor Relationship Game:

Uniformität und Integration: Nahtlose Zusammenarbeit der Komponenten wird aufgrund der Fehler in den UML-Diagrammen nur teilweise widergespiegelt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird ausführlich auf die gegebenen UML-Diagramme eingegangen. Es werden mit weiteren Diagrammen wie zum Beispiel Paketdiagrammen, zusätzliche Diagramme der UML generiert.

Konformität: Ist bis auf die Fehler in dem UML-Klassendiagramm und UML-Sequenzdiagramm erfüllt.

Vertrauenswürdigkeit: Ist teilweise erfüllt. Die Erstellung weiterer, jedoch nicht geforderter, UML-Diagramme und die intensive textuelle Auseinandersetzung mit diesen steigern die Vertrauenswürdigkeit in das Softwaredesign. Jedoch bewirken die Fehler innerhalb des UML-Klassendiagramms und des UML-Sequenzdiagramms das Gegenteil.

Logische Richtigkeit UML-Klassendiagramm: Es werden alle nötigen Klassen erkannt. Gleiches gilt für die Attribute und Methoden. Es fehlt jedoch die Beziehung zwischen den Klassen „Actor“ und „Movie“ zu der „ActorGraph“ Klasse. Ferner fehlt auch die Beziehung zwischen der „TMDBApi“ Klasse und der „GraphCreation“ Klasse. Es sind auch die Kardinalitäten oder Beschriftungen der Beziehungen nicht vorhanden.

Syntaktische Richtigkeit UML-Klassendiagramm: Syntaktische Richtigkeit durch das Fehlen der Beschriftung und Kardinalitäten bei den Beziehungen eingeschränkt.

Logische Richtigkeit UML-Sequenzdiagramm: Es fehlt eine „Main“ oder „User“ Lifeline. Des Weiteren fehlt die initiale Abfrage der Schauspieler- und Filmdaten mithilfe einer Message an die „TMDBApi-Klasse“. Ferner erscheint das Diagramm unvollständig, da das Speichern bzw. die Ausgabe des erstellten Graphen nicht abgebildet wird. Letztendlich wird aber zumindest ein fließender Kommunikationsverlauf dargestellt.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Ist syntaktisch richtig.

Logische Richtigkeit Dateibaum: Dateibaum ist richtig und nahezu vollständig. Es fehlen lediglich Strukturen zu dem Erstellen eines Akzeptanztests.

Hone:

Uniformität und Integration: Ist besonders durch die UML-Diagramme gegeben.

Differenzierung zwischen Design und Code: Es wird kein Code generiert, sondern eine gute Grundlage für das Entwickeln von Code geliefert.

Praktikabilität: Es wird kaum Bezug auf die UML-Diagramme genommen und bei dem bestehenden Bezug ein Fehler vorgenommen.

Konformität: Ist gegeben.

Vertrauenswürdigkeit: Die strikte Übereinstimmung mit der Anforderungsspezifikation ist gegeben.

Logische Richtigkeit UML-Klassendiagramm: Das UML-Klassendiagramm ist logisch vollkommen richtig. Es werden zudem die Beziehungen beschriftet und Kardinalitäten angegeben.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist gegeben.

Logische Richtigkeit UML-Sequenzdiagramm: Das UML-Sequenzdiagramm ist logisch richtig. Ein möglicher Kritikpunkt dieses Diagramms liegt in der letzten Message. Für die CLI-Klasse wäre es eigentlich sinnvoll in Form eines Booleans mitgeteilt zu bekommen, ob der „write_json()“ Aufruf erfolgreich abgeschlossen werden konnte oder nicht. Jedoch ist die Angabe des Werts „None“ konsistent zu dem angegebenen Rückgabetyp dieser Methode in dem UML-Klassendiagramm. Somit handelt es sich bei dem zuvor angesprochenen Kritikpunkt nicht um einen logischen Fehler des Diagramms.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Ist richtig.

Logische Richtigkeit Dateibaum: Erneut Redundanzen in Bezug auf Ordner und Dateien für die Durchführung von Tests. Ansonsten vollständig. In der Erklärung zu der „setup.py“ Datei existiert jedoch ein Fehler. So wird angegeben, dass diese Datei unter anderem UML- Diagramme enthalten würde.

Hybrid Images:

Uniformität und Integration: Ist basierend auf den UML-Diagrammen gegeben.

Differenzierung zwischen Design und Code: Es wird teilweise in den textuellen Ausführungen des Designs auf potenzielle Imports eingegangen. Auch wird gezeigt wie das Programm über die CLI starten könnte. Jedoch findet in Bezug darauf eine klare Trennung zwischen Design und Code statt.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen.

Konformität: Ist erfüllt.

Vertrauenswürdigkeit: Design entspricht weitestgehend den Vorgaben aus der Anforderungsspezifikation.

Logische Richtigkeit UML-Klassendiagramm: Der Komplexität des Projekts entsprechend einfacher gehalten. Erscheint logisch richtig. Es fehlt die Beschriftung der Beziehungen und die Kardinalitäten ebendieser. Attribute und Methoden erscheinen richtig.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist bis auf das Fehlen von Beschriftungen für die Beziehungen und Kardinalitäten erfüllt.

Logische Richtigkeit UML-Sequenzdiagramm: Konversation erscheint fehlerhaft. So sollten die ersten beiden Messages rekursiv sein, da die benannten Methodenaufrufe in der „ImageProcessor“ Klasse selbst stattfinden. Abgesehen davon, wird ein fließender Konversationsverlauf dargestellt.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Ist erfüllt.

Logische Richtigkeit Dateibaum: Dateibaum ist fast vollständig. Es fehlen jedoch Strukturen für spätere Unittests und Akzeptanztests. Mitgelieferte „requirements.txt“ ist ebenso richtig.

Login Registration:

Uniformität und Integration: Ist aufgrund des UML-Diagramms und des fast vollständigen Dateibaums gegeben.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird kein Bezug auf die UML-Diagramme genommen.

Konformität: Entspricht den Best-Practices weitestgehend.

Vertrauenswürdigkeit: Die Anforderungen der Anforderungsspezifikation sind definitiv erkennbar.

Logische Richtigkeit UML-Klassendiagramm: Klassen erscheinen sinnvoll. Kardinalitäten und Beziehungen sind vorhanden. Es fehlt lediglich die Beschreibung der jeweiligen Beziehungen.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist bis auf die fehlende Beschreibung der Beziehungen vorhanden. Es musste jedoch der PlantUML-Code angepasst werden, um das Diagramm anzeigen zu können.

Logische Richtigkeit UML-Sequenzdiagramm: Unvollständig. Es fehlt die Unterscheidung zwischen den drei möglichen Szenarien „Login“, „Register“ und „View/Delete Users“ mithilfe von „alternatives“. Kommunikationsverlauf erscheint jedoch sinnvoll.

Syntaktische Richtigkeit UML-Sequenzdiagramm: Ist erfüllt.

Logische Richtigkeit Dateibaum: Fast vollständig. So fehlen die Strukturen zu der Implementierung von der Benutzerauthentifizierung.

Readtime:

Uniformität und Integration: Ist aufgrund der Diagramme erfüllt.

Differenzierung zwischen Design und Code: Es wird kein Code generiert.

Praktikabilität: Es wird sinnvoll Bezug auf die erstellten UML-Diagramme genommen und zusammengefasst, welchen Sinn diese in dem Rahmen des Softwaredesigns erfüllen.

Konformität: Ist erfüllt.

Vertrauenswürdigkeit: Die Anforderungen bezüglich der zu unterstützenden Dateiformate sind aufgrund der Abstrahierung der Methodenaufrufe in dem UML-Klassendiagramm nicht so gut zu erkennen wie bei anderen Softwaredesigns. Letzteres ist jedoch nicht von großer Relevanz, da der Dateibaum mithilfe der darin angelegten Strukturen für die Unittests ohnehin Aufschluss über die zu unterstützenden Formate gibt.

Logische Richtigkeit UML-Klassendiagramm: Es werden drei Klassen dargestellt, welche sinnvolle Beziehungen aufweisen. Allerdings fehlen in jeder Klasse Attribute. Ferner fehlen auch Kardinalitäten und Beschriftungen der Beziehungen. Es werden die dargestellten Methoden abstrahiert, sodass die Berechnung der Lesezeit unter Angabe des Formats stattfindet. Ob dies richtig ist, hängt gänzlich von der schlussendlichen Implementierung dieses Designs ab.

Syntaktische Richtigkeit UML-Klassendiagramm: Ist, bis auf das Fehlen der Kardinalitäten und Beschriftungen der Beziehungen, richtig.

Logische Richtigkeit UML-Sequenzdiagramm: Kommunikationsverlauf ist sinnvoll dargestellt. Es werden hier nicht die Fälle unterschieden, welches aber aufgrund der abstrahierten Methoden keinen Fehler darstellt.

Syntaktische Richtigkeit UML-Sequenzdiagramm: das UML-Sequenzdiagramm ist syntaktisch richtig.

Logische Richtigkeit Dateibaum: Der Dateibaum ist vollständig und die „requirements.txt“ ist ebenso vollständig und richtig.