

paluno  
The Ruhr Institute for Software Technology  
Institut für Informatik und Wirtschaftsinformatik  
Universität Duisburg-Essen

**Bachelorprojekt**

**Entwicklung eines Satzbau-Prüfmoduls für das E-Assessmentsystem JACK**

Felix Schmidt

3039031

Wilfried Florentin Wakeu Kontchipo

3027726

Essen, 00.00.2018

Betreuung: Dr. Michael Striewe

Erstgutachten: Prof.Dr Michael Goedicke

Studiengang: Angewandte Informatik – Systems Engineering

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Essen, am 00.00.2018

**Inhaltsverzeichnis**

[Eidesstattliche Erklärung i](#_Toc508268590)

[**Inhaltsverzeichnis** ii](#_Toc508268591)

[1 Einführung 1](#_Toc508268592)

[2 Theoretischer Hintergrund 2](#_Toc508268593)

[3 Auswahl der Aufgabenstellungen und Aufgabentypen 3](#_Toc508268594)

[4 Konzept für Umsetzung 4](#_Toc508268595)

[4.1 Auswahl der Systeme 4](#_Toc508268596)

[4.2 Komponentendiagramm 4](#_Toc508268597)

[5 Beschreibung der Umsetzung 5](#_Toc508268598)

[6 Tests 6](#_Toc508268599)

[6.1 Junit-Test 6](#_Toc508268600)

[6.2 Scenario-test 6](#_Toc508268601)

[6.3 Integrationstest 6](#_Toc508268602)

[7 Konkretes Beispiel 7](#_Toc508268603)

[8 Bewertung der Umsetzung und Fazit 8](#_Toc508268604)

# Einführung

Den Studierenden der Universität Duisburg-Essen steht das E-Assessment-System JACK zur Verfügung. Das ist ein server-basiertes System für die Durchführung computergestützter Übungen und Prüfungen mit automatischer Bewertung und Feedback-Generierung. Die Architektur von Jack ermöglicht es neue Softwarekomponenten zur Aufgabengenerierung oder Aufgabenprüfung in das bestehende System zu integrieren. So sind beispielsweise schon einige Komponenten, wie eine Komponente zur Bewertung von Programmieraufgaben oder zur Generierung und Bewertung mathematischer Übungsaufgaben, integriert. Für dieses E-Assessment-System ist die Integration einer neuen Softwarekomponente im Bereich des Sprachtrainings geplant.

Im Rahmen einer Seminararbeit wurden bereits Softwarekomponenten identifiziert, die im Bereich Satzbau für eine Integration in JACK in Frage kommen. Der nächste Schritt besteht darin, geeignete Aufgabenstellungen für Aufgaben zum Satzbau zu ermitteln und zu bestimmen auf welche Art und Weise diese Aufgaben mit den Möglichkeiten, die Jack anbietet, am besten präsentiert werden können. Für die Generierung und Bewertung dieser Aufgaben soll anschließend ein Prüfmodul entwickelt werden.

Ziel dieses Bachelorprojekts ist es also, auf Grundlage der Ergebnisse der Seminararbeit „Softwarekomponenten zum Satzbau“ ein weiteres Prüfmodul für JACK zu entwickeln, das Aufgabenstellungen aus dem Bereich Satzbau im Kontext des Sprachtrainings anbietet.

# Geplante Vorgehensweise, Verfahren und Werkzeuge

In einem ersten Schritt ist eine Analyse des E-Assessmentsystems JACK notwendig. Zum einen dient diese Analyse dazu die Codestellen für eine Integration eines neuen Prüfmoduls auszumachen und festzustellen auf welche Art und Weise ein neues Prüfmodul konfiguriert werden muss. Zum anderen ist es das Ziel dieser Analyse zu ermitteln welche Aufgabentypen (z.B. Lückentexte, Multiple Choice, etc.) JACK anbietet.

Der nächste Schritt besteht darin verfügbare E-Assessmentsysteme aus dem Bereich Sprachtraining zu untersuchen, um festzustellen wie andere Systeme Aufgaben zum Satzbau darstellen. Hierbei wird der Fokus auf typische Aufgabenstellungen und Aufgabentypen gerichtet. Es soll also ermittelt werden, mit welchen Problemen aus dem Bereich des Satzbaus Lernende typischerweise konfrontiert werden und wie diese Probleme den Lernenden präsentiert werden.

Auf dieser Basis und unter Berücksichtigung der Ergebnisse der Seminararbeit „Softwarekomponenten zum Satzbau“ soll danach festgelegt werden welche Aufgabenstellungen zum Satzbau das neue Prüfmodul anbieten soll und wie diese durch die von JACK angebotenen Aufgabentypen präsentiert werden sollen.

Zur Umsetzung des Prüfmoduls wird dann eines oder mehrere der durch die Seminararbeit identifizierten Systeme ausgewählt und die Implementierung des Prüfmoduls wird geplant. Die entstandene Lösung soll anschließend durch eine ausreichende Anzahl geeigneter Tests geprüft werden.

Als Abschluss der Arbeit wird die Lösung einer kritischen Bewertung unterzogen und es soll ermittelt werden welche Anpassungen oder Erweiterungen für die entstandene Lösung in Frage kommen.

# Theoretischer Hintergrund

Grundlage für das Projekt bildet die Seminararbeit „Softwarekomponenten zum Satzbau“ [Referenz]. Eine Auswahl in dieser Arbeit identifizierten Systeme sollen verwendet werden um das Prüfmodul umzusetzen. Die identifizierten Systeme setzen Methoden der Computerlinguistik um und lassen sich in die Kategorien Parser und Generatoren natürlicher Sprache unterteilen. Computerlinguistik ist ein interdisziplinärer Bereich zwischen Informatik und Linguistik, der die Interaktion zwischen Computern und der natürlichen Sprache von Menschen untersucht. Parsing und die Textgenerierung sind Teilbereiche der Computerlinguistik. Parsing beschäftigt sich mit der Zerlegung von natürlich-sprachlichen Sätzen in ihre Bestandteile und der Analyse der grammatikalischen Eigenschaften dieser Bestandteile. Ziel der Textgenerierung hingegen ist es natürlich-sprachliche Texte durch die Verwendung von Algorithmen automatisch zu produzieren.

In der Seminararbeit wurden anhand von definierten Kriterien Systeme identifiziert, welche zur Generierung von Sätzen oder zur Analyse von eingegebenen Sätzen in das E-Assessmentsystem JACK integriert werden können. Es wurden jeweils vier Generatoren natürlicher Sprachen und vier Parser gefunden und in Detail analysiert, die die Kriterien teilweise erfüllten. Diese waren SimpleNLG, KPML, FUF/SURGE sowie OpenCCG für die Generatoren und OpenNLP, Standford Core NLP, Lucence sowie NLTK für die Parser.

Bei der Untersuchung der Generatoren natürlicher Sprache stellte sich heraus, dass sich die identifizierten Systeme stark in Benutzung und Anbindbarkeit an das E-Assessmentsystem JACK unterscheiden. Außerdem sind sie nicht in der Lage, die Wörter für die zu generierenden Sätze automatisch auszuwählen. D.h. um einen Satz mit Hilfe dieser Systeme zu generieren, müssen alle Wörter, die der zu generierende Satz enthalten soll, den Generatoren als Input eingegeben werden. Für die Systeme zur Analyse von Sätzen ergab sich dagegen, dass die für diese Aufgabe identifizierten Systeme sich in ihrer Funktionalität und Benutzung stark gleichen. Allerdings wurde festgestellt, dass der Einsatz der identifizierten Systeme für eine detaillierte Analyse von Sätzen im Bereich des Sprachtrainings nicht ausreichend ist. Z.B. das Prüfen von Sätzen in Hinblick auf Fehler oder das Erkennen von zusätzlichen grammatikalischen Informationen (Handlungsform, Deklinationen, etc.) zu den einzelnen Wörtern ist von keiner dieser Parser-Systeme abgedeckt.

# Auswahl der Aufgabenstellungen und Aufgabentypen

Da die genauen Aufgabenstellungen, die das neue Prüfmodul anbieten soll noch nicht feststehen, werden diese basierend auf vorhandener Sprachtrainingssysteme erstellt. Anschließend werden die Aufgaben anhand der vom E-Assessmentsystem JACK angebotenen Aufgabentypen dargestellt. Für Sprachtraining-Aufgaben kommen formularbasierte Aufgaben häufig in Frage. JACK unterstützt schon die Erstellung solcher Aufgaben.

Zu den Aufgabentypen gehören Fill-In-Aufgaben oder Drop-Down-Aufgaben, Multiple-Choice-Aufgaben sowie SCList-Aufgaben. Bei Fill-In-Aufgaben muss der Lernender seine Lösungen in die dafür vorgesehenen Kätschen eintragen. Die Fill-In-Kätschen können entweder freie Eingabefelder oder Drop-Down-Menü sein. Ist das Fill-In-Kätschen ein freies Eingabefeld, muss der Lernender seine Lösung darin eintippen. Wenn das Fill-In-Kätschen ein Drop-Down-Menü ist, muss der Lernender eine passende Antwort aus den im Menü vorgegebenen Antworten auswählen. Bei Multiple-Choice Aufgaben wird für einen Fragetext mehrere Antwortmöglichkeiten vorgegeben, von denen der Lernender die richtigen Antworten finden und auswählen muss. Der SCList-Aufgabentyp ist dem Multiple-Choice-Aufgabentyp ähnlich. Im Unterschied werden hier für eine Fragestellung mehrere Fragetexte und Antwortmöglichkeiten mit Hilfe einer Tabelle mit variabler Zeilen- und Spaltenanzahl dargestellt. Die Fragentexte stehen in der ersten Spalte und die Antwortmöglichkeiten in der ersten Zeile. Der Lernender muss jede Frage die richtigen Antworten finden und auswählen. [Referenz]

Als Aufgabenstellungen können generelle Aufgaben wie z.B. Aufgaben zur Bestimmung einer Eigenschaft für einen Satzteil vorkommen. Dabei wird dem Lernender gefragt, eine Eigenschaft wie z.B. das Subjekt, das Verb, das Akkusativ usw. in einem Satz zu bestimmen. Dafür sind Drop-Down-, Multiple-Choice- sowie SCList-Aufgabentypen geeignet, um solche Aufgaben darzustellen. Weitere Aufgabenstellungen wären Aktiv-/Passiv-Transformationen, Zeitformentransformationen oder noch W-Fragen.

Bei Aktiv-/Passiv-Transformationen wird ein Satz in aktiver bzw. passiver Form generiert. Es kann dem Lernenden gefragt, den Satz in passiver bzw. aktiver Form zu transformieren, in dem er seine Antwort mittels Fill-In- oder Drop-Down-Aufgabentypen in ein dafür vorgesehenes Fill-In-Kätschen einträgt oder auswählt. Diese Aufgabe kann auch mit Hilfe von Multiple-Choice- sowie SCList-Aufgabentypen vorgestellt werden. Dabei werden für einen Satz in aktiver bzw. passiver Form mehrere mögliche Antworten für die passive bzw. aktive Form des Satz vorgegeben.

Die Aufgaben über Zeitformentransformationen geschieht genauso wie die Aktiv-/Passiv-Transformationen. Hierbei wird ein Satz in einer bestimmten Zeitform generiert und dem Lernender muss den Satz in die vorgegebene Zeitform transformieren oder die richtigen Antworten aus vorgeschlagenen Antworten auswählen. Diese Aufgabe ist also mittels Fill-In-, Drop-Down- Multiple-Choice- sowie SCList-Aufgabentypen umsetzbar.

Bei W-Fragen-Aufgaben werden Fragen gestellt, welche sich auf einen vorgegebenen Satz beziehen. Ausgehend von einem Satz kann dem Lernenden gefragt, dem Satz in eine Frage umzuwandeln. Dafür sind Fill-In- oder Drop-Down-Aufgabentypen geeignet. Als Aufgabe kann auch eine Frage zu den Inhalten eines Satzes gestellt werden, auf die der Lernender antworten muss. Dadurch wird sein Verständnis über den Satz geprüft. Hierbei wären Fill-In- sowie Multiple-Choice-Aufgabentypen für die Darstellung der Aufgaben passend. Außerdem kann basierend auf vorgegeben Sätzen Ja/Nein-Fragen gestellt werden, die der Lernende beantworten muss. Für solche Aufgaben macht am besten Sinn, Multiple-Choice-Aufgabentypen zu nutzen.

# Konzept für Umsetzung

Welche Aufgabenstellungen und welche Aufgabentypen wurden gewählt?

Satz und richtige Lösung generieren

Satz anzeigen, Eingabefeld für Lösung bereitstellen,

User-Lösung mit richtiger Lösung vergleichen

Bei freien Eingabefelder wird bei falschen Antworten die Eingabe von Nutzer analysiert, um detaillierte Feedback zu generieren

Bewertung….

## Auswahl der Systeme

(Begründung)

## Komponentendiagramm

# Beschreibung der Umsetzung

# Tests

Das implementierte Prüfmodul soll auf verschiedenen Ebenen mit jeweils für die Ebene geeigneten Testverfahren getestet werden. Auf der detailliertesten Ebene soll das Modul mit Unit-Tests getestet werden. Das Zusammenspiel der einzelnen Komponenten soll auf einer höheren Ebene durch Integrationstests geprüft werden und auf der höchsten Ebene soll die Interaktion des Nutzers mit dem System über die Benutzeroberfläche getestet werden. Die Anzahl der Tests nimmt dabei mit dem Detaillierungsgrad zu.

## Modultests

[

Zunächst ist es wichtig, dass die innersten Komponenten des Prüfmodul isoliert voneinander wie erwartet funktionieren. Das bedeutet, dass die implementierten Klassen und Methoden isoliert in Hinblick auf ihre Funktionalität geprüft werden müssen. Dazu werden für jede zu testende Funktionalität die Äquivalenzklassen identifiziert und die Funktionalität wird mit je einem Repräsentanten dieser Äquivalenzklasse als Eingabe getestet. Bei der prüfen der inneren Komponenten wird dabei ein hoher Abdeckungsgrad angestrebt. (umformulieren und genauer beschreiben was bedeutet Äquivalenzklasse, was sind das für Vorteile, welche Arten von Abdeckung gibt es (Pfad/Anweisung/…)

]

Die Unit-Tests werden mit JUnit umgesetzt und können automatisch ausgeführt werden.

**Beispiel Unit-Test zur Bestimmung von Zeitformen aus POS-Tags**

Die Klasse TenseDeterminer verfügt über eine Methode, die Part-Of-Speech Tags als Eingabe entgegennimmt, anhand der Tags die Zeitform bestimmt und diese zurückgibt.

Zunächst wird für die Methode durch mehrere Testfälle geprüft, ob für eine Eingabe aus Tags, die eine bestimmte Zeitform repräsentieren, das Resultat dieser Zeitform entspricht. Für die Zeitform „Present Simple“ werden die Testfälle im Folgenden aufgeführt:

@Test

**public** **void** testDetermineTenseSimplePresentBase() {

String[] posTags = {"VB"};

String actualTense = TenseDeterminer.*determineTense*(posTags);

*assertEquals*("present simple", actualTense);

}

@Test

**public** **void** testDetermineTenseSimplePresentNon3rdPerson() {

String[] posTags = {"VBP"};

String actualTense = TenseDeterminer.*determineTense*(posTags);

*assertEquals*("present simple", actualTense);

}

**public** **void** testDetermineTenseSimplePresent3rdPerson() {

String[] posTags = {"VBZ"};

String actualTense = TenseDeterminer.*determineTense*(posTags);

*assertEquals*("present simple", actualTense);

}

Außerdem wird getestet, ob bei einer Eingabe aus Tags, die keine Zeitform repräsentieren, eine unbekannte Zeitform das Resultat ist:

@Test

**public** **void** testDetermineTenseWithUnknownTense() {

String[] posTags = {"VB", "VBN", "VBZ"};

String actualTense = TenseDeterminer.*determineTense*(posTags);

*assertEquals*("unknown tense", actualTense);

}

@Test

**public** **void** testDetermineTenseWithUnknownTags() {

String[] posTags = {"VBA"};

String actualTense = TenseDeterminer.*determineTense*(posTags);

*assertEquals*("unknown tense", actualTense);

}

Zuletzt wird geprüft, ob die Methode auch bei der Eingabe eines leeren Part-of-Speech Tag Arrays eine unbekannte Zeitform als Ergebnis liefert:

@Test

**public** **void** testDetermineTenseWithEmptyTags() {

String[] posTags = **new** String[0];

String actualTense = TenseDeterminer.*determineTense*(posTags);

*assertEquals*("unknown tense", actualTense);

}

## Integrationstests

Mit Integrationstest soll das Zusammenwirken der bereits durch Modultests geprüften Einheiten überprüft werden. Ohne Integrationstests kann nicht sichergestellt werden, dass die korrekt funktionierenden Einzelteile auch als zusammenwirkende Einheit korrekt funktionieren. Die Integrationstests werden demnach zeitlich gesehen nach den Modultests durchgeführt, da hier nur das Zusammenspiel der korrekt funktionierenden Einzelteile getestet werden soll.

Auch die Integrationstests werden in diesem Projekt mit JUnit-Tests umgesetzt. Im Gegensatz zu den Unit-Tests werden hier jedoch nicht einzelne Klassen isoliert voneinander getestet sondern größere funktional zusammengehörende Einheiten.

**Beispiel Integrationstests zur Generierung von einfachen Sätzen**

Für die in diesem Projekt hinzugefügte in sich geschlossene Funktionalität des Generierens von Sätzen wurde ein Integrationstest erstellt. Die Funktionalität wird als ganzes mit verschiedenen Eingaben getestet.

Zunächst wird geprüft ob die Funktion korrekte Sätze in den verfügbaren Zeitformen definieren kann. Als Eingabe dienen die Wörter des Satzes, bestehend aus Subjekt, Verb und Objekt sowie eine korrekte Zeitform. Im folgenden Beispiel wird ein Satz in der Zeitform „present simple“ mit den Wörtern „woman“, „eat“ und „orange“ generiert.

@Test

**public** **void** testGenerateSimpleSentencePresent() **throws** EvaluatorException, OpenMathException {

List<Object> args = **new** ArrayList<Object>();

args.add(OMCreator.*createOMSTR*("woman"));

args.add(OMCreator.*createOMSTR*("eat"));

args.add(OMCreator.*createOMSTR*("orange"));

args.add(OMCreator.*createOMSTR*("present simple"));

Object result = *func*.evaluate(args);

*assertEquals*(OMCreator.*createOMSTR*("The woman eats oranges."), result);

}

Neben der korrekten Verwendung der Funktionalität sollte auch getestet werden, wie die Funktion mit irregulären Eingaben umgeht. Dazu dienen beispielsweise die folgenden zwei Testfälle. Der Erste der beiden Testfälle prüft, ob eine Ausnahme geworfen wird, wenn eine unbekannte Zeitform als Eingabe an die Funktion übergeben wird. Der Zweite Testfall prüft, ob eine Ausnahme auftritt, wenn der Funktion zu viele Parameter übergeben werden.

@Test(expected = EvaluatorException.**class**)

**public** **void** testGenerateUnknownTense() **throws** EvaluatorException, OpenMathException {

List<Object> args = **new** ArrayList<Object>();

args.add(OMCreator.*createOMSTR*("woman"));

args.add(OMCreator.*createOMSTR*("eat"));

args.add(OMCreator.*createOMSTR*("orange"));

args.add(OMCreator.*createOMSTR*("present abc"));

Object result = *func*.evaluate(args);

*fail*();

}

@Test(expected = FunctionInvalidNumberOfArgumentsException.**class**)

**public** **void** testGenerateWithTooManyParams() **throws** EvaluatorException, OpenMathException {

OMOBJ omobj = ExpressionParser.*parse*("genSimpleSentence('Test','Test','Test','Test','Test')", **null**, **null**);

OMToResultVisitor.*getInstance*().visit(omobj);

*fail*();

}

## Szenario-Tests

Mit Szenario-Tests sollen einzelne konkrete und komplexe Anwendungsfälle des Systems überprüft werden. Für dieses Projektes bedeutet das, dass beispielsweise die Erstellung einer Aufgabe zum Satzbau durch einen Nutzer in der Rolle des Dozenten oder das Bearbeiten einer Aufgabe durch einen Nutzer in der Rolle eines Lernenden als Szenarien getestet werden sollen. Die Szenario-Tests werden im Rahmen dieses Projektes manuell durch Interaktion eines Testers mit der Benutzeroberfläche der Webanwendung überprüft, da es sich um eine geringe Anzahl von Szenarien handelt. Für die Szenario-Tests dieses Projekts werden die auszuführenden Aktionen bzw. notwendigen Eingaben des Nutzers, erwartete Ausgaben und Zustände des Systems, das Resultat des Szenarios sowie mögliche Zwischenresultate nach der Ausführung einzelner Aktionen definiert. Bei der Durchführung eines Szenario-Tests werden diese Aktionen mit den definierten Eingaben durch einen Tester ausgeführt und es wird geprüft, ob die Ausgaben und Zustände des Systems mit den im Testfall definierten Ausgaben bzw. Zuständen übereinstimmen und ob das Resultat der Ausführung korrekt ist. Die Szenario-Tests erfolgen in diesem Projekt als letzte Testinstanz, da sie komplex und aufwendig in der Durchführung sind und von der Korrektheit der einzelnen Komponenten abhängen.

Konkret werden für dieses Projekt Szenarien zum Erstellen von Aufgaben und Szenarien für das Beantworten von Aufgaben mit richtigen und falschen Eingaben getestet. Das Beantworten von Aufgaben mit falschen Eingaben hat in diesem Projekt eine besondere Relevanz, da in diesen Fällen ein von der Eingabe abhängiges Feedback erzeugt und angezeigt wird. Für jeden der drei neu erstellten Aufgabentypen werden demnach mindestens drei Szenarien (Erstellen einer Aufgabe, Beantworten einer Aufgabe mit richtiger Eingabe, Beantworten einer Aufgabe mit falscher Eingabe) getestet.

**Beispiel Szenario Test Beantwortung einer Aufgabe**

Im Folgenden ist ein Szenario-Test zur Beantwortung einer Aufgabe aus dem Bereich Zeittransformation als Beispiel aufgeführt. In diesem Szenario soll getestet werden, ob das System die Aufgabe zur Transformation eines Satzes darstellt, die Eingabe des Nutzers korrekt überprüft wird und ob dem User bei einer falschen Eingabe ein entsprechendes Feedback angezeigt wird.

Testfall Definition – Korrektes Bearbeiten einer Aufgabe

|  |  |  |
| --- | --- | --- |
| Schritt | Eingabedaten | Erwartetes Resultat/erwarteter Zustand |
| Einloggen in das System in der Rolle des Studierenden | Gültiger Benutzername und Passwort eines Studierendenkontos. | Der Benutzer ist als Studierender eingeloggt. Die zur Verfügung stehenden Aufgaben werden angezeigt. |
| Auswahl der Aufgabe „SimplePresent2SimplePast Feedback“ | - | Die Aufgabe „SimplePresent2SimplePast Feedback“ wird geöffnet und die Aufgabenstellung wird angezeigt. |
| Eingabe einer falschen Lösung und Einreichen der Lösung über den dazu vorgesehenen Button. | Falsche Lösung für den dargestellten Satz. (Keine Eingabe eines Determiners für das Subjekt, Angabe des Subjekts in Plural statt Singular, Angabe des Satzes in Zeitform Present Progressive statt Simple Past, Angabe des Objekts in Singular statt Plural) | Die Lösung wird vom System als falsch bewertet und dem User wird als Feedback mitgeteilt, dass der Determiner für das Subjekt fehlt, dass das Subjekt in Plural anstelle von Singular ist, dass seine gewählte Zeitform present progressive anstatt simple past entspricht und dass das Objekt seines Satzes in Singular anstatt Plural ist. |

Testfalldurchführung:

Schritt 1:

Ein Nutzer in der Rolle des Studierenden wird am System Angemeldet und die dem Nutzer zugeordneten Aufgaben werden angezeigt.

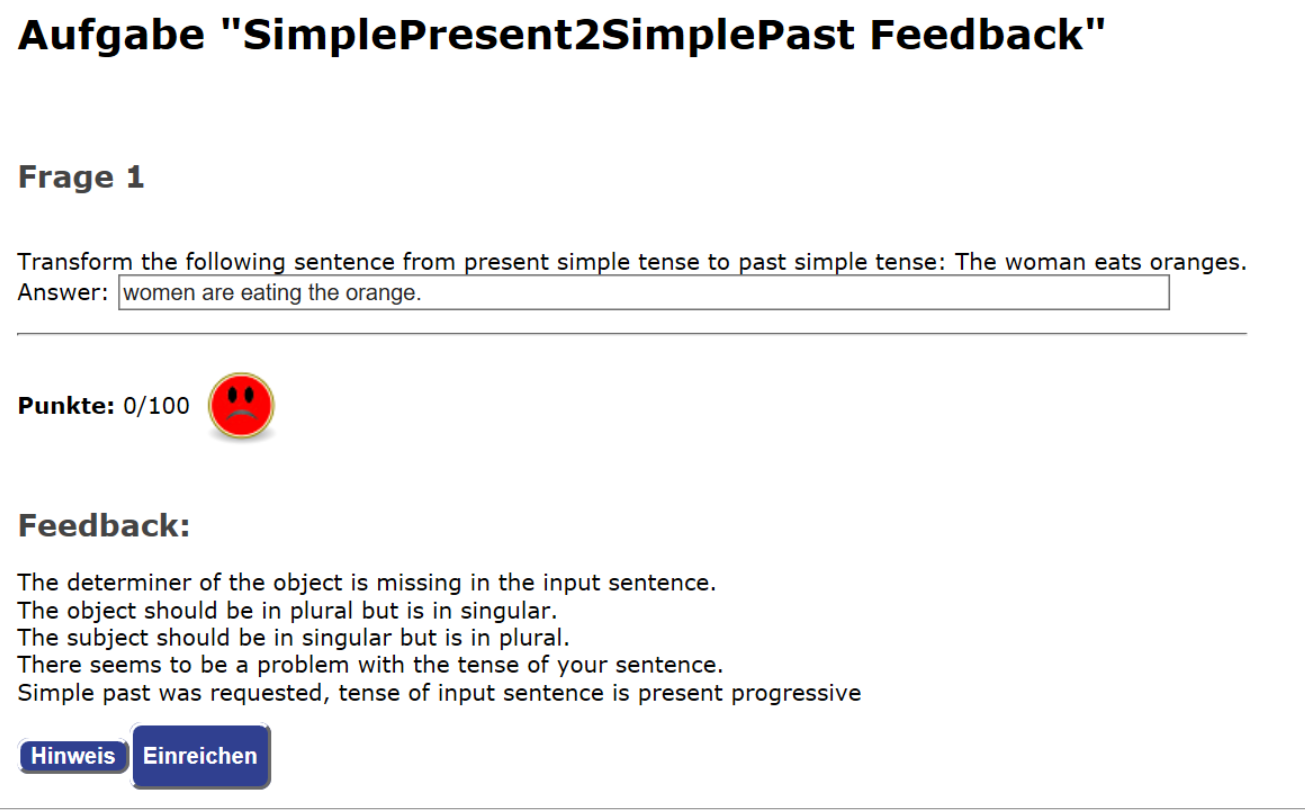
Schritt 2:

Der Benutzer wählt die Aufgabe „SimplePresent2SimplePast Feedback“, die Aufgabe wird geöffnet und wie im Folgenden dargestellt:



Schritt 3:

Ein den Eingabedaten entsprechender Satz wird in das dafür vorgesehene Feld eingegeben. Das System meldet, dass die Antwort falsch ist und gibt ein Feedback zurück, welches dem definierten Feedback entspricht.



# Konkretes Beispiel

(Screenshoot) für Lehrender und Lernender

# Bewertung der Umsetzung und Fazit

(1) -> Einschränkungen