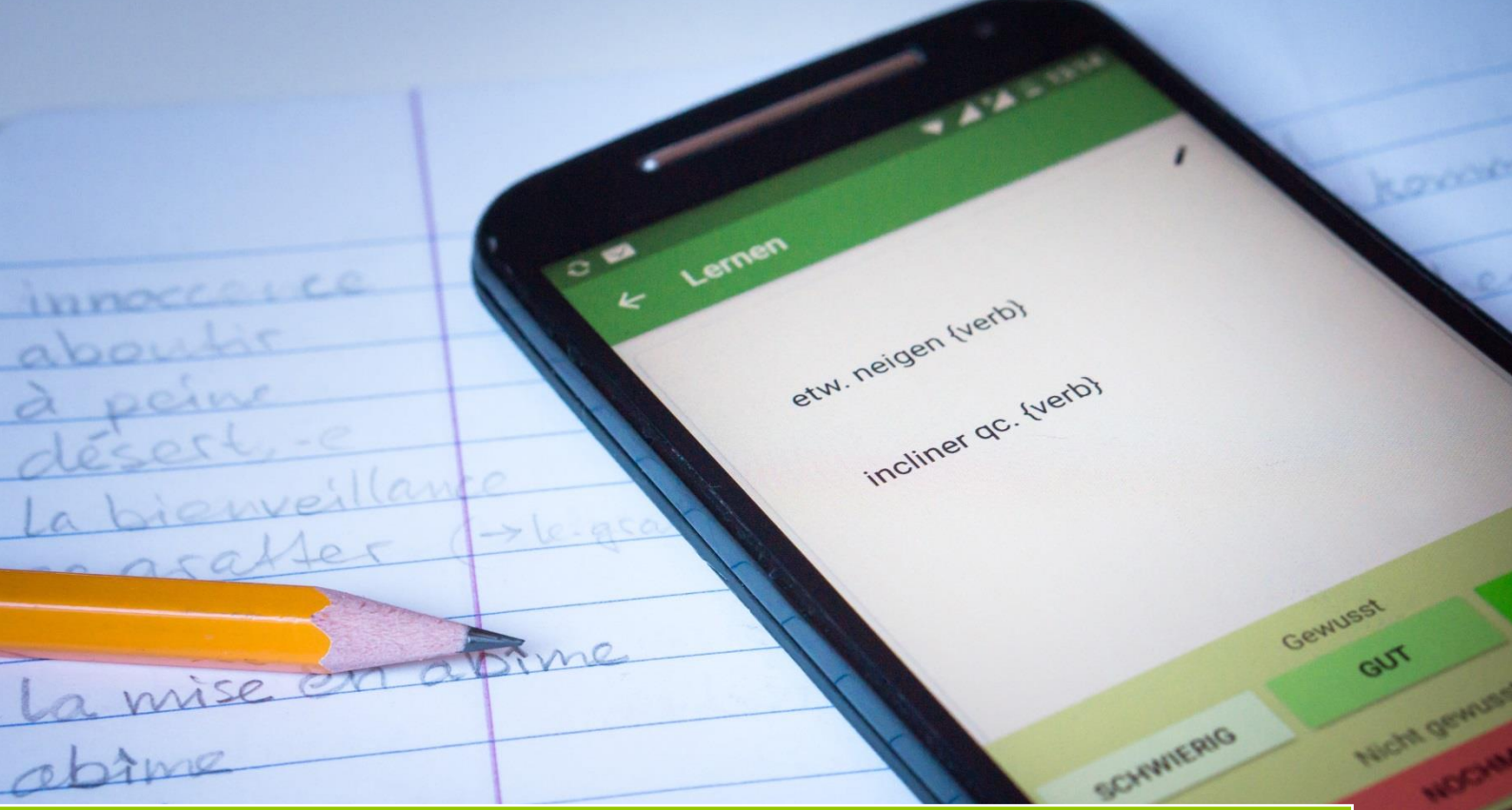
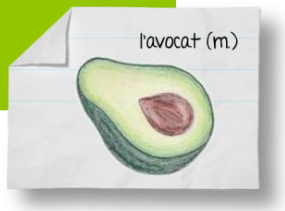


Entwicklung eines digitalen Lernbegleiters

Von Lernmethodik bis zur App



Vocado

Maturaarbeit von Jonas Wolter
im Fach Informatik

Betreuer: Dr. Rainer Steiger
Korreferent: Raphael Riederer
Kantonsschule Schaffhausen
06.12.2016

Inhaltsverzeichnis

1 Einleitung 3
1.1 Motivation und Ziel 3
1.2 Vorgehensweise 4
2 Lernforschung und Lernmethoden 5
2.1 Persönliche Lernmethoden 5
2.2 Wissenschaftliche Theorien 9
2.3 Wie unser Gehirn funktioniert 10
2.4 Gedächtnisstufen 11
2.5 Assoziationshilfen 14
2.6 Lerntypen 14
2.7 Begleitinformationen 16
2.8 Emotionen 17
2.9 Wiederholung 18
2.10 Pausen 21
3 Appkonzept 23
3.1 Ideen aus dem Lernforschungsteil 23
3.2 Probleme herkömmlicher Vokabel-Lernapps und Lösungen 27
3.2.1 Hinzufügen neuer Wörter 27
3.2.2 Vernetzung mit analogen Lernmethoden 28
4 Programmierung 30
4.1 Einleitung 30
4.2 Modulplanung 31
4.3 Wahl des Betriebssystems 33
4.4 Daten 33
4.5 Grundlagen 34
4.5.1 Android Studio 34
4.5.2 Objektorientierte Programmierung 35
4.5.3 Aufbau einer Android-App 39
4.6 Übersicht 43
4.7 MainActivity 45
4.8 Datenverwaltung 50
4.8.1 SQLite-Datenbank 50
4.8.2 DataSource 54

4.8.3 Daten ändern ... 58
4.9 DictionaryFragment ... 59
4.10 UnitsOverviewFragment ... 63
4.11 UnitViewerActivity ... 64
4.12 TestActivity ... 65
4.13 Algorithmus zur Planung der Wiederholungszeitpunkte ... 66
4.14 EditVocableActivity ... 72
4.15 Ausblick ... 73
5 Schlusswort ... 74
6 Dank ... 75
7 Bibliographie ... 76
7.1 Literaturverzeichnis ... 76
7.2 Hinweis zu Internetquellen ... 77
7.3 Abbildungen ... 77
8 Anhang ... 77
8.1 Quellcode und App Download ... 77

1 Einleitung

1.1 Motivation und Ziel

Als Kantischüler sehe ich mich tagtäglich mit der Aufgabe konfrontiert, möglichst viel Wissen in möglichst kurzer Zeit zu lernen.

Jede einzelne Schulstunde ist vollgepackt mit neuen Informationen. Und auch zuhause ist der Lernalltag noch nicht vorbei, Hausaufgaben sollten erledigt werden, am nächsten Tag steht schon wieder die nächste Französisch-Prüfung an – der Stoffumfang mal wieder gigantisch. Wenn ich dann spät am Abend ins Bett gehe, kommt oft der Gedanke „Hätte ich doch früher angefangen zu lernen". Für normale Hausaufgaben ist mal wieder keine Zeit geblieben.

Was ich mit diesem Beispiel illustrieren möchte, ist, dass bei dieser Art des Lernens die Sorgfältigkeit oft auf der Strecke bleibt. Eine Woche später ist die Hälfte der Vokabeln schon wieder vergessen, und spätestens ein Jahr nach der Matur sind die komplizierten Grammatikregeln längst aus dem Gedächtnis verschwunden.

Mir persönlich wäre nachhaltiges Lernen oft lieber. Unter dem hohen Zeitdruck ist es jedoch leichter, die Prüfungen mit kurzfristigem Lernen zu bestehen. Es erfordert äusserst viel Disziplin und Engagement, über den Rahmen der nächsten Prüfung hinauszudenken und eine rare freie Stunde dafür zu opfern, etwas Altes nochmals zu wiederholen oder schon früh im Voraus zu lernen. Daher auch die unter Schülern weit verbreitete „Lernbulimie".

Leider wird jedoch genau dieses Thema, welches für den Lernerfolg eigentlich essentiell wichtig wäre, an unserer Schule kaum thematisiert.

Mein persönliches Anliegen ist es, genau hier mit meiner Maturaarbeit anzusetzen und mich zuerst ausgiebiger mit Lernforschung zu befassen. Dieses Interesse für wirksames Lernen möchte ich mit meiner Faszination für Computer und fürs Programmieren verbinden.

Ich möchte die Ergebnisse in einer Smartphone-App umsetzen, welche die Schüler praktisch beim Lernen begleitet. Das Ziel ist es, nachhaltigeres Lernen deutlich zu vereinfachen und dazu beizutragen, Lernbulimie in gewissem Masse einzudämmen.

Ich habe mir während meiner Zeit an der Kantonsschule ein solches Werkzeug gewünscht, konnte aber unter den zahlreichen Apps nichts Passendes finden.

Entsprechend meinem Anfangsbeispiel konzentriere ich mich bei der App auf das Erlernen von Vokabeln. Dieser Bereich ist einerseits mit einem grossen Lernaufwand verbunden, andererseits besteht meiner Meinung nach grosses Potenzial, dieses Lernen durch digitale Unterstützung zu vereinfachen.

1.2 Vorgehensweise

Da für mich vor allem der praktische Nutzen der Applikation im Vordergrund steht, habe ich mich für eine interdisziplinäre Vorgehensweise entschieden. Ich beschäftige mich daher nicht nur mit der technischen Seite der App-Entwicklung, sondern setze mich zuerst ausführlich mit Pädagogik und Lernforschung auseinander. Diese Erkenntnisse fliessen dann in das Konzept der Applikation ein. Im letzten Schritt erfolgt dann die Umsetzung, die Programmierung.

In diesem Sinne habe ich den Hauptteil in drei grosse Teile unterteilt:

1. Lernforschung und Lernmethoden
2. Appkonzept
3. Programmierung

Der dritte Teil der Arbeit ist modular aufgebaut. Das Ziel besteht darin, die wichtigsten Ideen aus der Lernmethodik in einer App umzusetzen. Es war zu Beginn schon klar, dass ich niemals alle Ideen würde umsetzen können – dies hätte den zeitlichen Rahmen gesprengt. Daher habe ich mich entschlossen, mich vorerst auf die Grundfunktionen der App zu beschränken und soweit möglich weitere Funktionen hinzuzufügen. Die Idee dahinter ist die, dass ich die App sehr wahrscheinlich auch nach der Maturaarbeit noch weiterentwickeln werde.

2 Lernforschung und Lernmethoden

2.1 Persönliche Lernmethoden

Bevor ich mich theoretisch mit dem Thema Lernmethoden auseinandersetzte, wollte ich mir zuerst einen Überblick verschaffen, mit welchen Methoden meine Mitschülerinnen und Mitschüler Vokabeln lernen. Zu diesem Zweck habe ich drei Schülerinnen und zwei Schüler anhand eines konkreten Beispiels über ihre persönlichen Lernmethoden befragt. Folgende Auflistung gibt einen Einblick.

Ausgangssituation:

- Vokabelliste Deutsch-Fremdsprache mit 100 neuen Vokabeln
- Prüfung in 4 Wochen

Schülerin A beginnt etwa 3 – 4 Tage vor der Prüfung, die Vokabeln zu lernen. Zuerst schreibt sie alle Vokabeln (Deutsch und Fremdsprache) handschriftlich einmal ab. Die nächsten Tage lernt sie mit dieser handschriftlichen Liste durch Abdecken und lautem Aussprechen der Wörter. Schwierige Wörter schreibt sie eventuell nochmals auf.

Gesamt investierte Zeit: ca. 1h 15min (davon 30-45min fürs Abschreiben)
Durchschnittliche Note: 5.25

Den Lernerfolg der Methode schätzt sie als gut ein, wenn auch eher kurzfristig. Bei Prüfungen macht sie meist nur kleine Fehler (bei der Rechtschreibung). Nach ein paar Wochen sind jedoch viele der Wörter wieder vergessen.

Schülerin A hat noch nie mit einem digitalen Vokabeltrainer gelernt, könnte sich dies jedoch vorstellen. Vor allem für unterwegs sei es praktisch. Allerdings findet sie es mühsam, die Wörter zuerst eintippen zu müssen. Ausserdem schätzt sie ihren Lernerfolg bei handschriftlichem Schreiben höher ein.

Schülerin B beginnt 1-2 Tage vorher, die Vokabeln zu lernen. Sie lernt durch Abdecken und mündlichem Aufsagen der fremdsprachigen Wörter. Falls sie mehr Zeit hat, schreibt sie die Wörter ebenfalls auf. Sie wiederholt die Vokabeln höchstens einmal.

Gesamt investierte Zeit: ca. 5 – 30min
Durchschnittliche Note: 3

Den kurzfristigen Erfolg schätzt sie mässig ein, für längerfristiges Behalten sei die Methode nicht geeignet. Auch bei Prüfungen schneidet sie meist schlecht ab. Sie ist nicht wirklich zufrieden mit der Lernmethode, jedoch sei diese zeitsparend. Die Schülerin sieht Verbesserungspotential, wenn sie mehr Zeit

investieren würde. Sie gibt an, dass sie lieber früher mit dem Lernen beginnen würde, ihr dazu jedoch oft die Motivation fehlt.

Schülerin B hat noch nie digitale Vokabeltrainer verwendet, hat jedoch vor, dies auszuprobieren. Ein Hemmnis sei das langwierige Eintippen. Auch sie kann sich Wörter beim handschriftlichen Schreiben besser einprägen.

Schüler C beginnt 3 – 4 Tage vor der Prüfung zu lernen. Zuerst liest er alle Wörter und deren Übersetzungen durch. Anschliessend lernt er durch Abdecken der fremdsprachigen Wörter in täglicher Wiederholung. Er lernt lieber in einer ruhigen Lernumgebung, meistens zuhause.

Gesamt investierte Zeit: 3 – 4h
Durchschnittliche Note: 5.25

Er schätzt den kurzfristigen sowie längerfristigen Erfolg der Methode als gut ein. Bei Prüfungen macht er nur wenige Fehler (kleine Fehler wie Artikel, accent aigu, etc.). Der Grossteil der Wörter bleibt auch nach der Prüfung im Gedächtnis. Schüler C ist mit der Lernmethode zufrieden, sie sei zwar wahrscheinlich nicht die zeiteffizienteste, habe sich jedoch bewährt.

Schüler C ist beim Lernen noch nie auf die Idee gekommen, digitale Vokabeltrainer zu verwenden. Wichtig wäre es ihm, dass eine solche Anwendung flexibel ist und zu einem Wort auch mehrere Übersetzungen erlaubt. Er ist der Meinung, dass man während des Eintippens der Wörter nicht viel lernt, er bevorzugt deshalb vorgefertigte Wörterlisten. Wichtig ist für ihn, dass die Übersetzungen stimmen.

Schülerin D beginnt 1-2 Tage vor der Prüfung mit Lernen. Sie liest die Vokabeln zuerst alle einmal durch. Sie unterteilt die Liste in Portionen von ca. 15 Wörtern. Dann deckt sie die fremdsprachigen Wörter ab und schreibt alle Wörter der Portion in schnellem Tempo auf (nicht besonders schön, es geht mehr um die Bewegung). Anschliessend korrigiert sie diese und markiert mit rotem Stift die Fehler. Für jede Portion wiederholt sie diese Prozedur drei Mal, bevor sie zur nächsten Portion übergeht. Abschliessend nimmt sie sich nochmals die schwierigen Wörter vor und schreibt diese mehrmals untereinander. Schülerin D lernt nicht so gerne an öffentlichen Orten, ihre bevorzugte Lernumgebung ist ihr Schreibtisch zuhause.

Gesamt investierte Zeit: 1h – 1h 30min
Durchschnittliche Note: 5.5 - 6

Sie ist sehr zufrieden mit ihrer Lernmethode, bei Prüfungen erzielt sie immer sehr gute Ergebnisse. Sie schätzt, dass ungefähr die Hälfte der Vokabeln auch nach ein paar Wochen noch im Gedächtnis vorhanden ist, das sei für sie in Ordnung.

Trotzdem wünscht sie sich im Nachhinein oft, früher mit dem Lernen begonnen zu haben.

Schülerin D hat digitale Vokabeltrainer schon einmal benutzt, findet es jedoch sehr mühsam, die Wörter vorher eintippen zu müssen. Praktisch seien diese jedoch, wenn man auf bereits erstellte Listen anderer Benutzer zugreifen kann. Ein Nachteil sei, dass man die Vokabeln beim digitalen Lernen nur visuell lerne. Schülerin D legt grossen Wert auf handschriftliches Schreiben.

Schüler E hat früher mit digitalen Vokabeltrainern gelernt, inzwischen lernt er jedoch wieder mit analogen Lernmethoden.

Beim analogen Lernen beginnt Schüler E in etwa 5 Tage vor der Prüfung mit dem Lernen. Am ersten Tag schreibt er die komplette Liste (Deutsch und Fremdsprache) zuerst einmal ganz ab. An den darauffolgenden Tagen wiederholt er das Gelernte täglich durch Abdecken der fremdsprachigen Wörter. Er fertigt eine Liste mit Wörtern an, mit denen er oft Mühe hat und schreibt diese nochmals.

Gesamt investierte Zeit: 2 – 3h (davon 1h 30min zum Abschreiben)
Durchschnittliche Note: 4.5 – 5.5

Beim digitalen Lernen beginnt er etwa 1 ½ Wochen vor der Prüfung zu lernen. Er lernt mit Memrise, einem spielerisch aufgebauten Online-Vokabeltrainer. Die einzutippenden Wörter teilt er sich mit Kollegen auf. Anschliessend lernt er täglich entweder mit dem Computer oder unterwegs mit dem Handy. Die Vokabel wird dabei in verschiedenen Formen (von klassischer Eingabe bis Hangman und Wörter verbinden) abgefragt.

Gesamt investierte Zeit: 6h
Durchschnittliche Note: 5 – 6

Ein Problem beim digitalen Lernen war für Schüler E vor allem der hohe Zeitaufwand, weshalb er wieder auf eine analoge Lernmethode umgestiegen ist. Das digitale, spielerische Lernen macht ihm zwar mehr Spass, jedoch sei es weniger effektiv. Die Ablenkungsmöglichkeiten beim Lernen mit dem Handy oder Computer seien grösser. Beim Lernen mit einem Buch kann er sich besser konzentrieren. Ausserdem prägt er sich Wörter durch handschriftliches Schreiben besser ein, als beim Schreiben mit Tastatur, da die Buchstaben dort nicht einzeln, sondern verbunden sind.

Zusammenfassung / Quintessenz

Die Befragungen sind aufgrund der geringen Anzahl nicht repräsentativ, jedoch geben sie einen, wie ich glaube, recht typischen Einblick in das Lernverhalten von Kantischülern. Die Schülerbefragungen lieferten für mich folgende Erkenntnisse:

- Niemand beginnt wirklich frühzeitig (mehr als 2 Wochen vor der Prüfung) mit dem Lernen.
- Zeitaufwand und Ergebnisse variieren signifikant, die Effizienz scheint stark von der Methode und selbstverständlich auch von der Person abzuhängen.
- Ein Grossteil der Schüler lernt nach wie vor mit analogen Lernmethoden. Am häufigsten wird dabei die Methode des Abdeckens und mündlichem Aufsagens von Wörtern verwendet, auch wenn diese nicht am effizientesten scheint.
- Effizient scheinen handschriftliches Schreiben und das Einteilen in Portionen sowie das gezielte Wiederholen von schwierigen Vokabeln.
- Im Nachhinein wünschen sich Viele, früher mit Lernen begonnen zu haben, dafür fehlt jedoch die Motivation.

Meine Erwartungen wurden in den Befragungen weitestgehend bestätigt. Ich war jedoch überrascht, wie klar die Ergebnisse bezüglich digitalem Schreiben versus handschriftlichem Schreiben ausfielen: Alle Befragten gaben an dass sie sich Begriffe deutlich besser einprägen, wenn sie diese von Hand schreiben.

Interessant waren für mich auch die angegebenen Gründe, warum keine digitalen Vokabeltrainer verwendet werden. Genannt wurde:

- Grösserer Zeitaufwand
- Mühsames Eintippen neuer Wörter
- Probleme mit falsch eingetippten Wörtern
- Mehr Ablenkungsmöglichkeiten
- Bevorzugung von handschriftlichem Lernen
- Nicht auf Idee gekommen

2.2 Wissenschaftliche Theorien

Nach diesem praxisbezogenen Einstieg möchte ich nun zur wissenschaftlichen Sichtweise übergehen und Lernen in einem grösseren Rahmen betrachten. Es sei im Vorhinein zu erwähnen, dass im Bereich Lernforschung keine einheitliche wissenschaftliche Meinung besteht. Bei der Recherche bin ich auf viele Publikationen unterschiedlicher Autoren gestossen, die sich auch teilweise widersprechen[1]. Im Grundsatz ähneln sie sich jedoch meistens. Der Einfachheit halber und um den Umfang nicht zu weit auszudehnen, werde ich aus diesem Grund im Folgenden darauf verzichten, zu jedem Thema die wissenschaftliche Diskussion zu schildern, sondern mich auf einige wichtige Erkenntnisse konzentrieren. Letztlich geht es darum, Ideen für die Konzipierung einer Lern-App zu gewinnen, weniger um die endgültige wissenschaftliche Verifikation. Es sei an dieser Stelle jedoch erwähnt, dass diese Diskussion existiert und auch weiter geführt wird. Ich verzichte hier jedoch auf deren detaillierte Darstellung, da sie im Hinblick auf die Umsetzung in einer App nicht relevant ist.

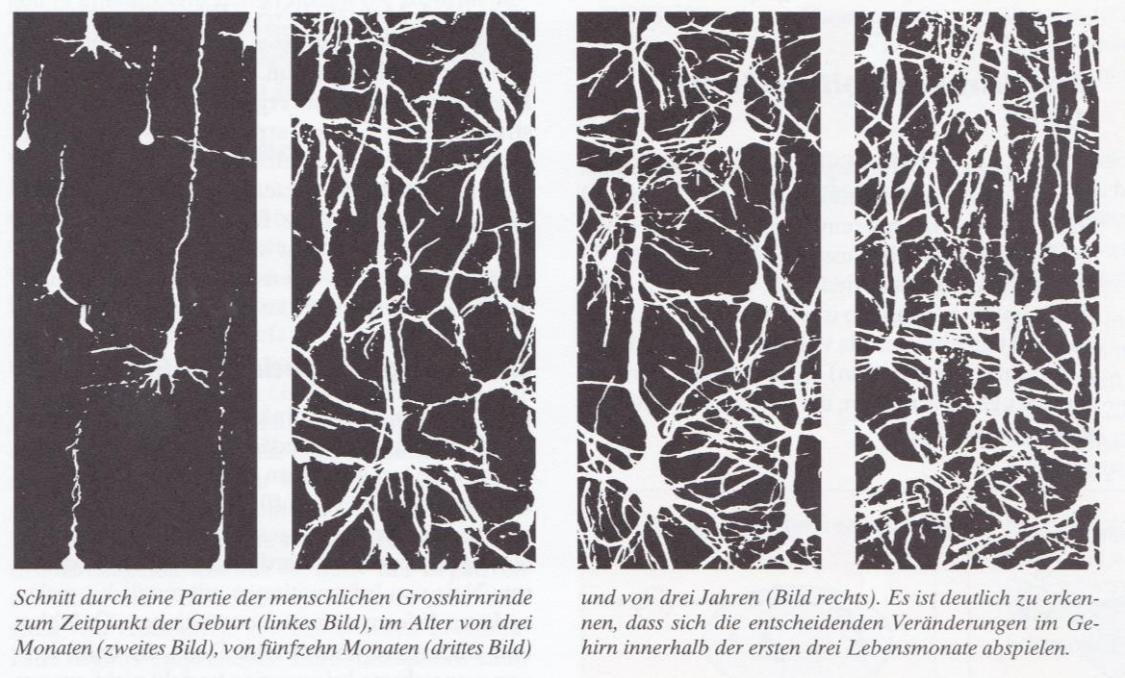
Der Grund, dass ich mich in den folgenden Abschnitten hauptsächlich auf die Theorien Frederic Vesters[2] und Tony Buzans[3] stützen werde, besteht darin, dass diese viele nützliche Erkenntnisse und praktische Ansätze liefern, die sich auch in einer Lern-App umsetzen lassen.

[1] Beispiel für den wissenschaftlichen Diskurs:
Looß, Maike: *Lerntypen?, Ein pädagogisches Konstrukt auf dem Prüfstand*
http://www.ifdn.tu-bs.de/didaktikbio/mitarbeiter/looss/looss_Lerntypen.pdf
Sehr kritische Auseinandersetzung mit der Lerntypen-Theorie Frederic Vesters

[2] Vester, Frederic: *Denken, Lernen, Vergessen*. 2001.
[3] Buzan, Tony: *Kopftraining, Eine Anleitung zum kreativen Denken*. 1993.

2.3 Wie unser Gehirn funktioniert

Jeder Gedanke basiert auf biologischen Vorgängen im Gehirn. Das Gedächtnis als solches ist nicht lokalisierbar, sondern über das ganze Gehirn verteilt. Es im Grunde genommen ein riesiges Informationsnetz, bestehend aus Milliarden neuronaler Verknüpfungen. Das Grundmuster dieser Verknüpfungen entsteht hauptsächlich während der ersten drei Lebensmonate – sozusagen die „Hardware“ des Gehirns. Später hinzukommendes Wissen wird in diese Struktur eingearbeitet, das Grundmuster verändert sich jedoch kaum mehr. Abbildung 1 zeigt dies eindrücklich anhand eines Schnittes durch die Grosshirnrinde:



Schnitt durch eine Partie der menschlichen Grosshirnrinde zum Zeitpunkt der Geburt (linkes Bild), im Alter von drei Monaten (zweites Bild), von fünfzehn Monaten (drittes Bild) und von drei Jahren (Bild rechts). Es ist deutlich zu erkennen, dass sich die entscheidenden Veränderungen im Gehirn innerhalb der ersten drei Lebensmonate abspielen.

Abb. 1
Quelle: Vester, Frederic: Denken, Lernen, Vergessen
(Bild entnommen aus René Frick, Werner Mosimann: Lernen ist lernbar)

Ebenso wie das Gedächtnis aus Verknüpfungen besteht, ist auch ein Gedanke keineswegs eine einzelne Information, sondern vielmehr eine Verknüpfung mit anderen Informationen wie Erinnerungen, Emotionen, etc.

Lernen

Bei gleichzeitiger Aktivität zweier Synapsen werden neue Verknüpfungen aufgebaut oder vorhandene verstärkt. So lässt sich auch erklären, warum wir Wissen wieder vergessen. Verknüpfungen werden schwächer, wenn diese nicht mehr genutzt werden. Die Gehirnzellen werden dann für andere Verknüpfungen verwendet. Für Lernvorgänge ist es daher wichtig, dass das Wissen genutzt wird, ansonsten gehen die Verknüpfungen wieder verloren. Im Normalfall passiert dies auf natürlichem Weg im Alltag - die meisten

Lernvorgänge laufen unbewusst ab. Denn nicht nur das Auswendiglernen französischer Grammatikregeln, sondern auch das Einprägen einer Erinnerung (z.B. an die letzte Reise) ist ein Lernvorgang. Egal wo wir sind, was wir machen – wir lernen im Grunde genommen den ganzen Tag.

Doch nicht jeder Eindruck gelangt überhaupt in unser Gedächtnis. Entscheidend dafür, ob eine Information aufgenommen oder verworfen wird, sind Assoziationen. Kann eine neue Information mit einer bereits vorhandenen Information assoziiert, also verknüpft werden, wird diese aufgenommen.

2.4 Gedächtnisstufen

Bevor Informationen jedoch dauerhaft gespeichert werden, durchlaufen sie verschiedene Gedächtnisstufen[4]. Diese unterscheiden sich vor allem durch die zeitliche Dauer der Informationsspeicherung.

Gedächtnisstufe	**Zeitdauer[5]**
Ultrakurzzeitgedächtnis	10 – 20 Sekunden
Kurzzeitgedächtnis	ca. 20 Minuten
Langzeitgedächtnis	theoretisch unbegrenzt

Neue, über die Sinneswahrnehmung aufgenommene Reize bzw. Informationen gelangen zuerst ins Ultrakurzeitgedächtnis. In Form elektrischer Ströme zirkulieren diese im Gehirn, bevor sie nach ca. 10 bis 20 Sekunden wieder abklingen.

Eine Information gelangt nur weiter auf die nächst höhere Gedächtnisstufe, wenn Resonanz mit einer vorhandenen Erinnerung entsteht (also Assoziationen gebildet werden) oder wenn die Information innerhalb dieser Zeit bewusst abgerufen wird.

[4] Modellvorstellung, es existieren verschiedene Modelle
[5] In der Forschungsliteratur findet man verschiedene Zeitangaben. Hier angegeben sind die Werte aus Vester, Frederic: *Denken, Lernen, Vergessen*

Ein kurzes Beispiel[6] soll diesen Vorgang illustrieren:

American Football

Quelle: wikimedia.org

Beim rauen American Football sind Fouls besonders häufig. Bei den Spielern machten Forscher eine interessante Entdeckung: Spieler, die ein paar Minuten nach dem Foul zu dessen Hergang befragt wurden, waren nicht mehr in der Lage zu beschreiben, wie genau es zum Foul kam. Wurden die Spieler jedoch sofort nach dem Foul befragt, konnten sie den Vorfall ohne Mühe genauestens wiedergeben. Mehr noch: Spieler, die direkt nach dem Foul befragt wurden, waren auch später noch in der Lage, den Hergang zu schildern.

Während des Spiels sind die Spieler auf Sofortreaktionen eingestellt. Ankommende Impulse müssen schnell verarbeitet, anschliessend wieder vergessen werden. Normalerweise blockiert die Schockreaktion nach dem Foul den Übergang genauer Informationen vom Ultrakurzzeitgedächtnis ins Kurzzeitgedächtnis. Wird der Spieler allerdings innerhalb der ersten 20 Sekunden nach dem Foul befragt, müssen die Informationen bewusst abgerufen werden. Durch diese Wiederholung gelangen sie weiter ins Kurzzeitgedächtnis.

Für das Lernen ist dieser mehrstufige Speichervorgang zwar hinderlich, im Alltag hat er jedoch auch Vorteile. Er dient als Filter und sorgt dafür, dass nur wirklich wichtige Informationen ins Langzeitgedächtnis gelangen. Somit ist er für das Gehirn ein Schutz vor Reizüberflutung. Ausserdem ermöglicht er uns wesentliche Zusammenhänge zu erkennen.

Der Vorteil besteht darin, dass unterschiedlich genaue Informationen in verschiedenen Zeiträumen verfügbar sind. Dies wird beim Autofahren beispielsweise sehr gut deutlich:

---

[6] Beispiel entnommen aus Vester, Frederic: Denken, Lernen, Vergessen, S. 61
[7] Im Vergleich zum Buch werden die Bilder hier in leicht abgeändertem Zusammenhang verwendet

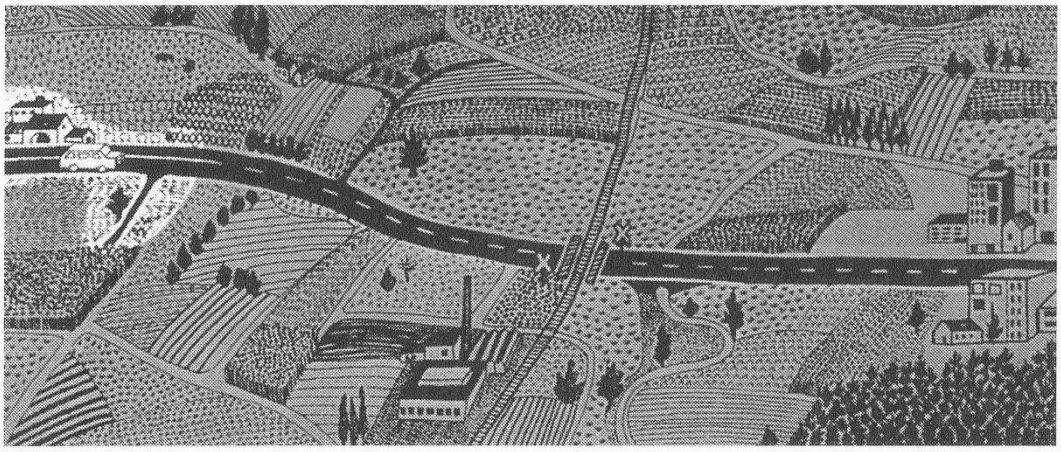


Abb. 2
Quelle: Vester, Frederic: Denken, Lernen, Vergessen[7]

Die helle Fläche in Abb. 2 markiert den Abschnitt, der durch die Sinneswahrnehmung ins Ultrakurzzeitgedächtnis aufgenommen wurde.

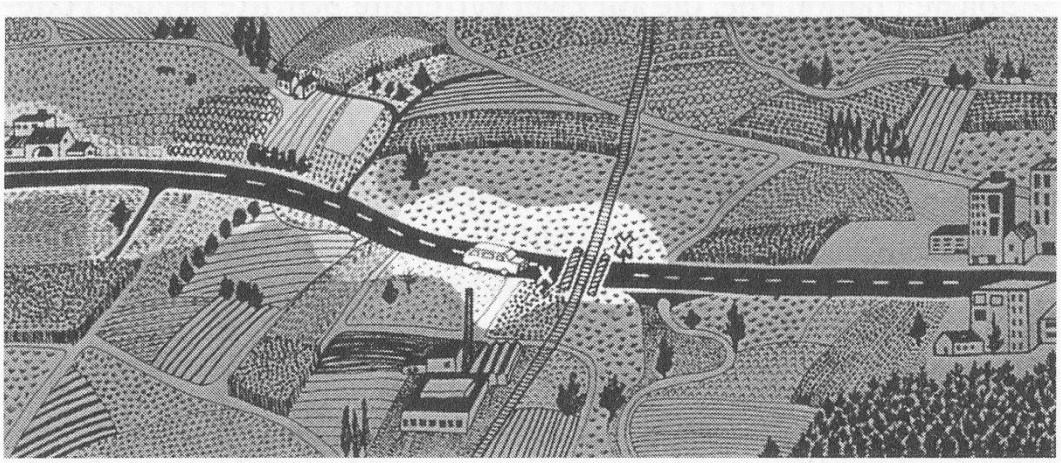


Abb. 3
Quelle: Vester, Frederic: Denken, Lernen, Vergessen[7]

Ein Teil der Informationen geht ins Kurzzeitgedächtnis über (repräsentiert durch mittelgraue Fläche in Abb. 3). Im Moment des Fahrens ist es für den Fahrer wichtig, Strasse, Verkehrssymbole, und Umgebung genau im Kopf zu haben, um richtig reagieren zu können. Zu einem späteren Zeitpunkt befinden sich die Informationen über diesen passierten Abschnitt im Kurzzeitgedächtnis, allerdings längst nicht mehr so genau. Der Fahrer muss allerdings auch nicht mehr wissen, wo genau der Baum stand, wie der Fussgänger am Strassenrand aussah, etc. Ein paar Tage später werden noch mehr Informationen verblasst sein. Das Gehirn sortiert für die Zukunft unwichtige Informationen aus und verhindert so Überforderung.

2.5 Assoziationshilfen

Wie im vorherigen Abschnitt geschildert, spielen Assoziationen bei der Informationsspeicherung und somit beim Lernen eine grosse Rolle. Ob eine neue Information mit bereits vorhandenen Informationen assoziiert werden kann, entscheidet schlussendlich darüber, ob diese die nächst höhere Gedächtnisstufe erreicht und längerfristig gespeichert werden kann. Dieser Vorgang verläuft nicht etwa willkürlich, sondern kann beeinflusst werden. Jedoch ist er von Erfahrungen sowie individuellem Denkmuster abhängig und somit von Person zu Person unterschiedlich. Je besser das Muster, in welchem eine Information angeboten wird, mit dem individuellen Assoziationsmuster im Einklang steht, desto mehr Assoziationen (und somit neuronale Verknüpfungen) können gebildet werden und desto besser wird die Information behalten. Ob jemand ein Thema versteht oder nicht versteht, hängt nach Vester nicht von der absoluten Intelligenz der Person, sondern vielmehr von der relativen Übereinstimmung des Informationsmusters mit dem Denkmuster ab.

Das Denkmuster kann kaum verändert werden, sehr wohl jedoch das Muster der Information. Ziel ist es daher, die Information auf unterschiedliche Arten zu verpacken, sodass dem Gehirn möglichst viele Assoziationshilfen geboten werden. Je mehr Arten der Erklärung angeboten werden, je mehr Wahrnehmungskanäle benutzt werden, desto fester und vielfältiger kann Wissen gespeichert werden (Vester 2001, S. 51).

2.6 Lerntypen

Aufbauend auf dem Assoziationsvorgang postuliert Vester die Existenz verschiedener Lerntypen. Da jeder Mensch eine andere „Verdrahtung“ der Neuronen, ein anderes Grundmuster besitzt, sind auch die Verknüpfungen zu unterschiedlichen Eingangskanälen verschieden stark ausgeprägt. So gibt es den auditiven Lerntyp, also Personen die Informationen bevorzugt über den Sinneskanal *Hören* aufnehmen. Oder den visuellen Lerntyp, Personen die Informationen am besten über den Sinneskanal *Sehen* aufnehmen. Nach diesem Muster lassen sich weitere Lerntypen nach verschiedenen Eingangskanälen unterscheiden. In der Realität kommen nach Vester jedoch nie reine bspw. visuelle Lerntypen vor, sondern immer Mischformen. In einer Schulklasse gibt es meist ebenso viele verschiedene Kombinationen von Lerntypen wie Schüler. Trotzdem sei das heutige Unterrichtssystem (mit dem häufigen Frontalunterricht) vor allem auf einen sehr spezifischen verbal-abstrakten Lerntyp zugeschnitten. Andere Lerntypen seien oft benachteiligt.

Um sein eigenes Lernverhalten besser zu verstehen, lässt sich ein einfacher Selbsttest[8] durchführen. Man versucht sich über verschiedene Methoden zehn Begriffe einzuprägen. Anschliessend wird überprüft, wie viele davon behalten wurden. Die Anzahl der behaltenen Begriffe wird auf verschiedenen Achsen eines Lernkreuzes (Abb. 4) eingetragen. Verbunden ergibt sich schliesslich eine Fläche, die den Lerntyp repräsentiert.

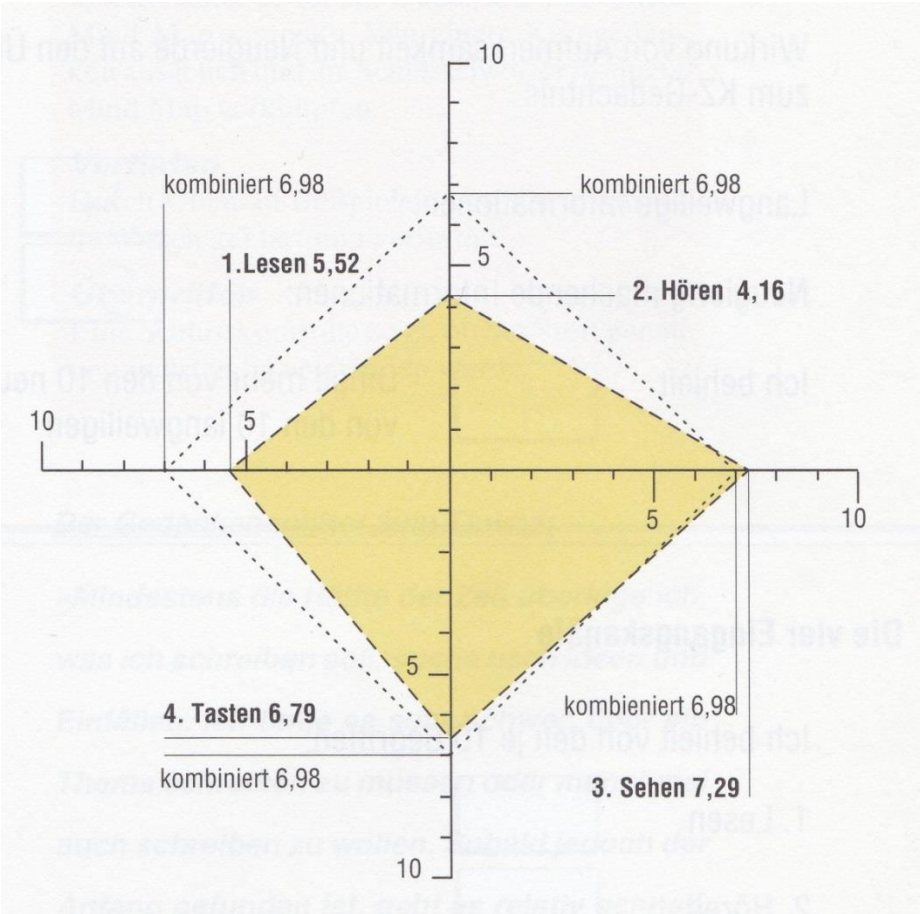


Abb. 4: Durchschnittlicher Lerntyp
Datenquelle: Vester, Frederic: Denken, Lernen, Vergessen
(Grafik entnommen aus René Frick, Werner Mosimann: Lernen ist lernbar)

Die Abbildung 4 zeigt die durchschnittlichen Werte der Testauswertung bei 500 Personen (Standardabweichung ~ ±1.6). Auffällig sind die deutlich höheren Ergebnisse bei den Eingangskanälen *Sehen* und *Tasten* im Vergleich zu den Eingangskanälen *Hören* und *Lesen*. Auch die Methode *Kombiniert* schneidet sehr gut ab.

[8] Beschreibung des Vorgehens: Vester, Frederic: *Denken, Lernen, Vergessen*, S. 201ff.

2.7 Begleitinformationen

Beim Lernen wird nicht nur der zu lernende Stoff eingeprägt, sondern immer auch damit verbundene Begleitinformationen. Also Eindrücke über den Ort, Gerüche, Gedanken, Emotionen, etc. zum Zeitpunkt der Informationsaufnahme. Im Gehirn werden diese Sekundärassoziationen nicht etwa getrennt gespeichert, sondern sind ebenso mit der gelernten Information verknüpft.

Oft erinnert man sich beim Lernen von Vokabeln nicht nur an die Wörter, sondern auch an welcher Stelle der Seite diese geschrieben standen. Umgekehrt kann es sein, dass ein bestimmter Geruch oder ein bestimmter Ort (etwa eine alte Wohnsiedlung) auf einmal wieder Erinnerungen wach ruft. Dieses Prinzip macht sich die Loki-Methode zunutze.

Loki-Methode

Die Loki-Methode ist eine Mnemo-Technik, die oft von Gedächtniskünstlern zum Auswendiglernen langer Folgen verwendet wird. Sie eignet sich auch zum Lernen von Wörtern oder Vokabeln, besonders dann, wenn auch die Reihenfolge entscheidend ist.

Vorgehen

Ein bekannter Weg (z.B. Arbeitsweg) wird im Kopf abgelaufen und die verschiedenen Orte bildhaft mit den zu lernenden Wörtern verbunden. Wäre das erste Wort der Liste zum Beispiel „Löffel“, könnte man sich einen riesigen goldenen Löffel vorstellen, der aus der Haustüre herausragt. Der nächste Ort wäre zum Beispiel die Strasse. Dort könnte man sich ein grosses oranges Auto in Form eines Goldfischs vorstellen, dass die Strasse entlangfährt – ein Beispiel für den Begriff „Goldfisch“. Und so weiter… Abstraktere Begriffe erfordern etwas mehr Vorstellungskraft, der Phantasie sind jedoch keine Grenzen gesetzt.

Auch wenn diese Methode zuerst etwas komisch erscheint, so ist sie doch wirksam. Umso skurriler die Verknüpfungen, desto besser kann man sich die Wörter merken. Um die gelernten Begriffe wieder abzurufen, geht man nun die Orte wieder der Reihenfolge nach durch.

2.8 Emotionen

Auch Emotionen und Gedanken gehören zu den zuvor geschilderten Begleitinformationen und nehmen im Lernprozess eine wichtige Rolle ein. Nach Vester werden Informationen, die mit Freude, Spass oder Spiel, Neugierde, Erfolgserlebnissen verbunden sind, weit besser verankert als Informationen, die mit negativen Emotionen verbunden sind. Bei angenehmen Begleitinformationen wird eine positive Hormonreaktion ausgelöst und Informationen gelangen leichter ins Gedächtnis. Bei intensiv erlebten Ereignissen, die mit starken Emotionen verbunden sind, reicht das einmalige Erleben aus, um für immer in Erinnerung zu bleiben. Anders sieht es leider oft beim Lernen von Schulstoff aus, wo Informationen nicht erlebt, sondern gehört oder gelesen werden.

Dahingegen beeinträchtigen negative Emotionen Speichervorgänge. Stress kann diese gar blockieren. Stress hilft in Gefahrensituationen körperlich schnell reagieren zu können und hindert dazu Denkvorgänge – ein Relikt aus Zeiten, wo sich der Mensch noch gegen wilde Tiere verteidigen musste.

Versuch

Stressfaktor	Antwort gewusst
Keiner	91 %
Angst machen	50 %
Fremd	41 %
Abstrakt	33 %

Abb. 5
Quelle: Vester, Frederic: Denken, Lernen, Vergessen (S.149)

Abb. 5 zeigt die Ergebnisse eines Versuchs, in dem Schüler über gelerntes Wissen befragt wurden. Variiert wurde jeweils die Abfragesituation (Verhalten und Formulierungen der Lehrperson), nicht jedoch der Inhalt der Fragen. Die statistischen Ergebnisse zeigen eine deutliche Verschlechterung der Leistungsfähigkeit unter Einwirkung von Stressfaktoren.

Beim Wiederholen der Informationen, wie das beim Lernen üblich ist, werden die als Begleitinformation gespeicherten positiven oder negativen Emotionen mitgelernt. Im positiven Fall kann dies Freude hervorrufen und die Motivation fördern, im negativen Frustration und Lernblockaden auslösen.

Umso wichtiger ist es daher, in der Schule wie zuhause eine angenehme Lernumgebung zu schaffen, die Lernprozesse begünstigt und nicht beeinträchtigt.

Die Motivation kann zusätzlich dadurch gesteigert werden, dass Lerninhalte in Portionen aufgeteilt werden, sodass der Stoff überschaubar wird und kein Gefühl von Überforderung entsteht.

2.9 Wiederholung

Die vorhergehenden Abschnitte haben sich vor allem mit der Aufnahme der Informationen beschäftigt. In diesem Abschnitt geht es um einen beim Lernen ebenso wichtigen Faktor – die Wiederholung. Ohne Wiederholung geht ein Grossteil der Informationen bereits nach kurzer Zeit wieder verloren (Abb. 6).

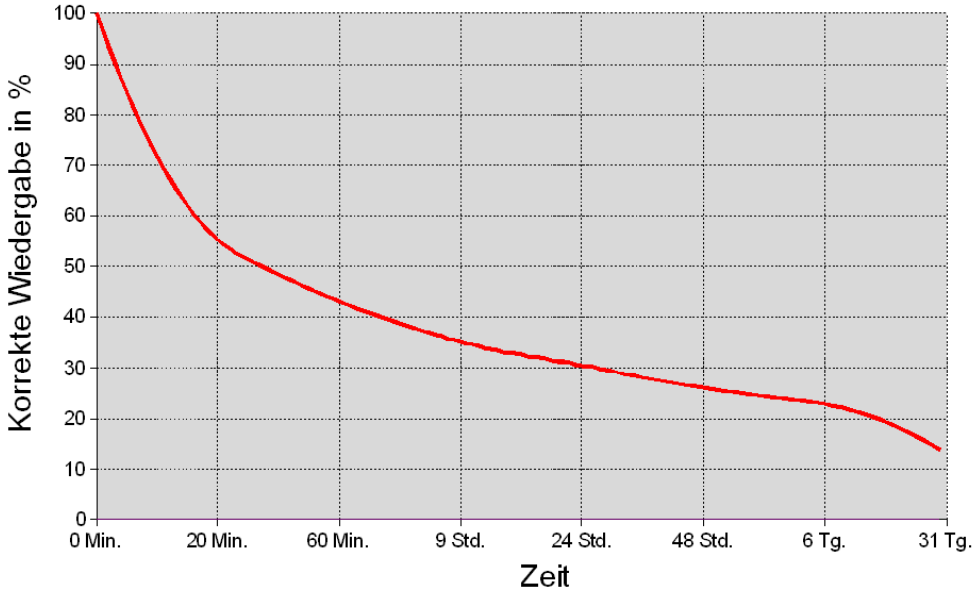


Abb. 6: Vergessenskurve nach Ebbinghaus
Datenquelle: Ebbinghaus, Hermann: Über das Gedächtnis
(Grafik: wikimedia.org)

Ebbinghaus entdeckte bereits 1885 durch Selbstversuche mit Silbenfolgen, dass die Erinnerung unverknüpfter Informationen nahezu logarithmisch abfällt. Ohne Wiederholungen ist nach 24 Stunden bereits 70% des Gelernten wieder vergessen.

Durch regelmässige Wiederholung lässt sich dieses schnelle Abfallen verhindern.

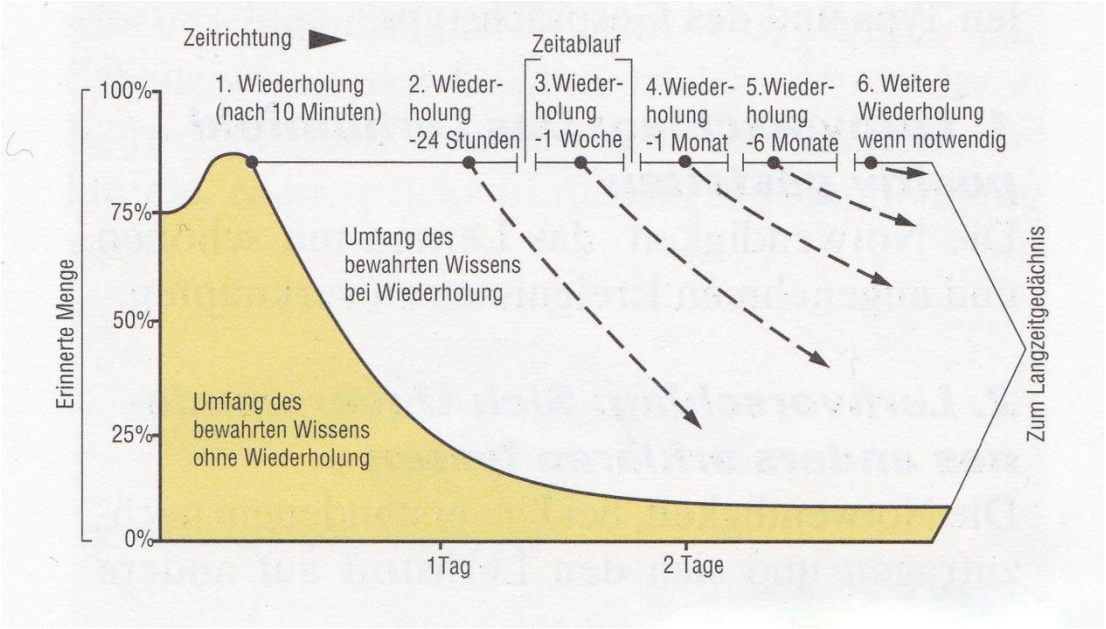


Abb. 7: Veränderungen der Vergessenskurve nach Wiederholungen
Quelle: Buzan, Tony: Kopftraining
(Grafik entnommen aus René Frick, Werner Mosimann: Lernen ist lernbar)

Bei jeder Wiederholung wird die Erinnerung wieder auf das Anfangsniveau gebracht. Abbildung 7 zeigt, dass die Vergessenskurve durch das Festigen der Informationen jedoch mit jedem Mal flacher wird.

Die Kurve beginnt mit dem Zeitpunkt einer abgeschlossenen Lernphase. Auffällig ist, dass die Erinnerung zuerst ansteigt, bevor sie abzufallen beginnt. Nach Tony Buzan sowie Frederic Vester benötigt das Gehirn für die Verarbeitung Zeit. Die Erinnerungsleistung erreicht erst einige Minuten nach der Lernphase ihren Höhepunkt, da erst dann alle wechselseitigen Verbindungen hergestellt worden sind und der Stoff „einsinken" konnte.[9]

Spaced Repetition

Da die Vergessenskurve mit jeder Wiederholung flacher wird, macht es auch Sinn, die Zeitintervalle zwischen Wiederholungen mit der Zeit zu erhöhen. Dieser Ansatz wird in der Lernforschung als Spaced Repetition bezeichnet. Verglichen mit der herkömmlichen Methode (gleichbleibende Wiederholungsintervalle) ist Spaced Repetition deutlich effektiver, wenn es darum geht, Informationen über einen langen Zeitraum speichern zu können.

Dabei stellt sich die Frage, wann genau diese Wiederholungszeitpunkte angesetzt werden sollen. Durch eine geschickte Planung lässt sich viel Zeit sparen. Wird zu häufig wiederholt braucht man unnötig viel Zeit für die

[9] Buzan, Tony: *Kopftraining, Eine Anleitung zum kreativen Denken* (S.73)

Wiederholungen, wird zu selten wiederholt, sind viele Informationen schon wieder vergessen und müssen wieder neu gelernt werden – was ebenfalls Zeit kostet.

Leitner Lernkartei

Ein einfacher aber relativ grober Ansatz ist die von Sebastian Leitner entwickelte und sehr populär gewordene Lernkartei.[10] Informationen werden nach dem Schema Frage – Antwort umformuliert und auf Vorder- und Rückseite von Karteikarten geschrieben und in die Lernkartei einsortiert. Durch unterschiedlich gestaltete Fächergrössen werden neue sowie nicht gewusste Karteikarten häufiger wiederholt als bereits gewusste Karten.

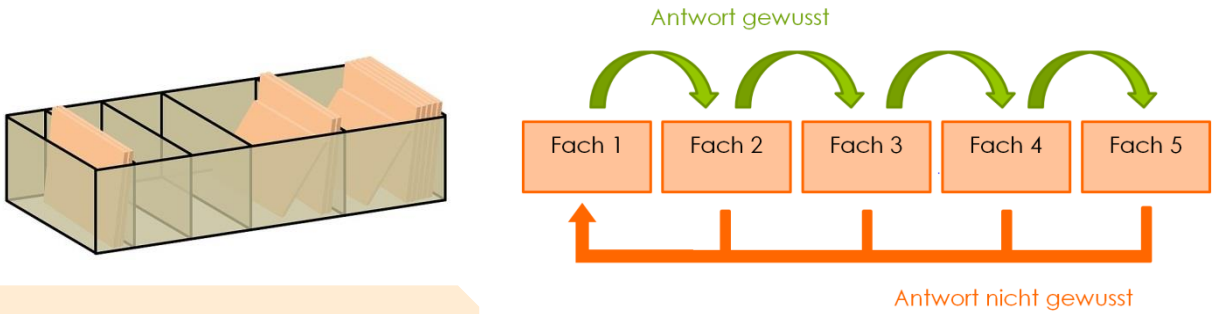


Abb. 8: Lernkartei nach Leitner
Quelle: wikimedia.org

Die Zeitintervalle sind nicht absolut, sondern relativ zueinander definiert. Sie hängen von der Einteilung des Lernenden ab.

Der Vorteil dieser Methode liegt in ihrer Einfachheit. Sie kann analog und ohne Berechnungen durchgeführt werden. Allerdings ist sie nicht unbedingt die effizienteste.

SuperMemo Algorithmus

Bei computerunterstütztem Lernen lassen sich auch komplexere Algorithmen verwenden, die sich noch besser der individuellen Gedächtnisleistung anpassen. Ein effizienter Algorithmus ist der von Piotr Woźniak im Jahr 1985 entwickelte SuperMemo-Algorithmus.

Der wesentliche Unterschied zum Leitner Algorithmus besteht darin, dass der Lernstand einer Karte nicht nur anhand von Gewusst/Nicht gewusst beurteilt wird, sondern der Lernende jeweils nach einer Einschätzung der Schwierigkeit gefragt wird. Dadurch lassen sich die Wiederholungszeitpunkte noch genauer planen. Anhand der Einschätzungen berechnet der Algorithmus für jede Karte individuell einen sogenannten Easyness-Factor, der sich mit der Zeit verändert. Anhand dieses Faktors wird dann der nächste Wiederholungszeitpunkt der Karte

[10] Leitner, Sebastian: *So lernt man lernen, Der Weg zum Erfolg*

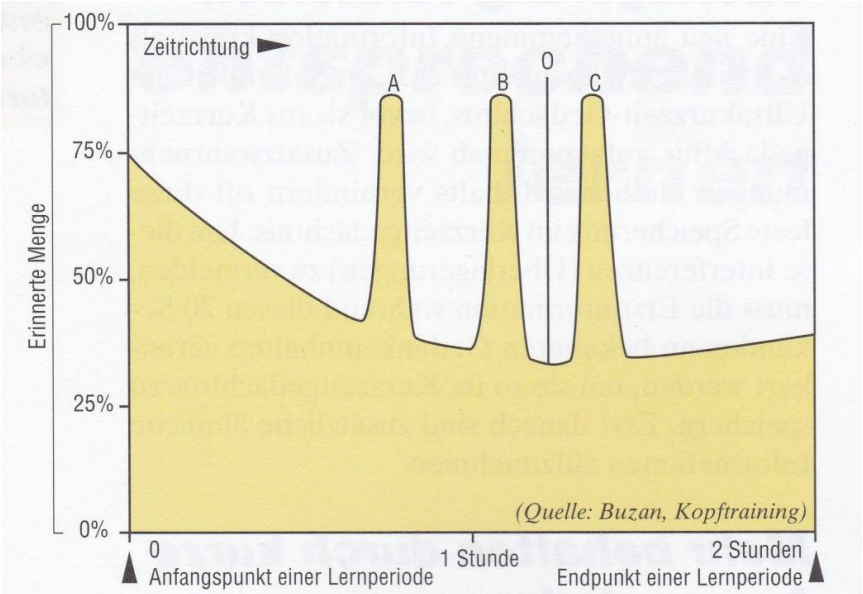
absolut berechnet. Im Durchschnitt sind die Intervalle in etwa ähnlich den von Tony Buzan vorgeschlagenen Intervallen.
Genaueres zum Algorithmus wird in Kapitel 4.13 ausgeführt.

2.10 Pausen

Zu guter Letzt ist zu beachten: Sehr langes Lernen bringt wenig.

Verständnisleistung und Erinnerungsleistung verhalten sich beim Lernen nicht gleich. Wie man bei der Erinnerungskurve sehen konnte, erreicht die Erinnerung erst einige Minuten nach Beenden der Lernperiode ihr Maximum. Das Ultrakurzzeitgedächtnis und das Kurzzeitgedächtnis brauchen ungestörte Zeit, um die kreisenden Impulse zu verarbeiten und an das Langzeitgedächtnis weiterzugeben. Wird dem Gehirn nicht genügend Verarbeitungszeit gegeben, sinkt die Erinnerungsleistung kontinuierlich ab (unter der Annahme einer konstanten Verständnisleistung).

Versuche haben gezeigt, dass wir uns bei normalen Bedingungen mehr an die Dinge erinnern, die am Anfang oder am Ende einer Lernperiode gelernt wurden, sowie an einzelne spezielle Themen (siehe Abb. 9).



Erinnern während des Lernens. Das Diagramm zeigt, dass wir mehr aus den Anfangs- und Endphasen einer Lernperiode und von Dingen behalten, die assoziiert verknüpft sind (A, B und C) und die aus dem Rahmen fallen oder einmalig sind (O).

Abb. 9
Quelle: Buzan, Tony: Kopftraining
(Grafik entnommen aus René Frick, Werner Mosimann: Lernen ist lernbar)

Wird dem Gehirn zu wenig Verarbeitungszeit gegeben, sinkt die Erinnerungsleistung drastisch ab, wie die folgende Grafik eindrücklich zeigt.

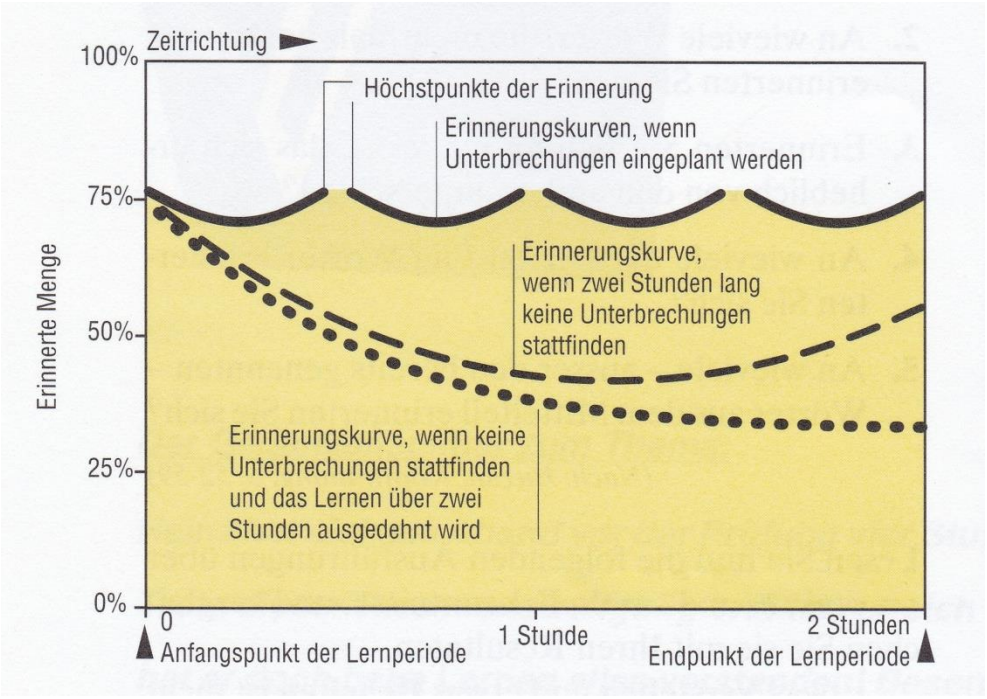


Abb. 10
Quelle: Buzan, Tony: Kopftraining
(Grafik entnommen aus René Frick, Werner Mosimann: Lernen ist lernbar)

Abb. 10 zeigt die Erinnerung während des Lernens mit und ohne Unterbrechungen. Eine Lernperiode zwischen 20 und 50 Minuten erbringt das beste Verhältnis zwischen Verständnis und Erinnerung.

Pausen steigern ausserdem die Konzentrationsfähigkeit, da die geistige und muskuläre Anspannung gelockert wird.

Beim längeren Lernen zu ähnlicher Informationen tritt immer mehr Interferenz auf. Informationen überlagern sich und verdrängen sich gegenseitig. Um die Erinnerungsleistung zu steigern, ist es daher auch sinnvoll, regelmässig das Thema zu wechseln.

3 Appkonzept

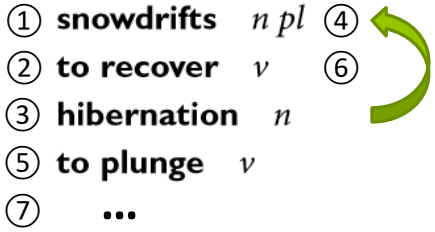
3.1 Ideen aus dem Lernforschungsteil

Welche Schlüsse lassen sich aus den geschilderten Erkenntnissen der Lernforschung ziehen, welche Ideen für die Erstellung einer Lern-App gewinnen? Die folgende Liste zeigt die Ideen, die ich aus dem Lernforschungsteil gewonnen habe und wenn möglich in die App einfliessen lassen möchte.

A. Schnelllernmethode

Wiederholen neuer Wörter innerhalb von 10 - 20 Sekunden

Statt eine Wörterliste linear von Anfang bis Ende durchzuarbeiten, wird nach jedem zweiten oder dritten Wort wieder auf das vorherige Wort zurückgesprungen.



Auf diese Weise könnten die Mechanismen der Informationsaufnahme besser ausgenutzt werden. Ein Wort wird so noch innerhalb der Zeitdauer der ersten Gedächtnisstufe, dem Ultrakurzzeitgedächtnis, wiederholt. Äquivalent zum Beispiel mit dem Footballspieler gelangt das Wort durch das bewusste Abrufen auf die nächst höhere Gedächtnisstufe, das Kurzzeitgedächtnis.

Beginnt man erst nach vollständigem Durcharbeiten der Liste zu wiederholen, sind diverse neue Wörter wieder aus dem Gedächtnis verschwunden und müssen erneut eingeprägt werden.

B. Eingangskanäle

Einbezug diverser Eingangs- und Sinneskanäle beim Lernvorgang:

Sehen: Arbeiten mit Bildern und Farben
Zur Visualisierung und besseren Verknüpfung könnten bei der Abfrage passende Bilder zu den Vokabeln angezeigt werden. Für den Nutzer soll auch die Möglichkeit bestehen, eigene Bilder mit persönlichem Bezug hinzuzufügen. Farben könnten als visuelle Gedankenstütze verwendet werden. So würden beispielsweise weibliche Substantive immer in roter Schrift gezeigt, männliche Substantive immer in blauer Schrift, etc.

Hören: Möglichkeit zum Vorsprechen lassen der Wörter
Beim Anzeigen der Lösung wird gleichzeitig eine Audio-Datei abgespielt. Auf diese Weise lernt der Nutzer gleichzeitig die korrekte Aussprache des Wortes.

Lesen
Im normalen Abfragemodus lernt der Nutzer durch Lesen: Zuerst wird nur eine Sprache, anschliessend die Lösung angezeigt.

Schreiben: Digital und handschriftlich
Nebst der normalen mündlichen Abfrage soll der Nutzer auch einen schriftlichen Abfragemodus auswählen können, wo die Wörter über die Tastatur eingetippt werden.
Ergänzend zur digitalen Eingabe könnte eine Funktion zum Ausdrucken von Wörterlisten mit Lösungsleiste zum Wegfalten hinzugefügt werden, sodass auch handschriftlich Lernende profitieren (mehr dazu in Kap. 3.2.2 Vernetzung mit analogen Lernmethoden).

Durch den Einbezug mehrerer Sinneskanäle wird das Wissen auf vielseitigere Art und Weise eingeprägt. Dem Gehirn werden so mehr Assoziationshilfen angeboten, wodurch mehr Verknüpfungen gebildet werden können. Mittels visueller Hilfsmittel kann man sich das Prinzip der Begleitinformationen zunutze machen: Man erinnert sich vielleicht nicht mehr an den Genus eines Wortes, sehr wohl jedoch noch an die Farbe. Über diese Begleitinformation lässt sich dann wieder ein Rückschluss auf den Artikel ziehen.

Die Eingangskanäle sind bei jeder Person unterschiedlich ausgeprägt. Durch verschiedene Abfragemuster wird es jedoch möglich, unterschiedlichen Lerntypen gerecht zu werden. Ausserdem macht es auch mehr Spass, wenn Lernmethoden abgewechselt werden können. Durch die Kombination mehrerer Methoden wird die Wirkung gesteigert.

C. Lernzeitpunkte

Algorithmus
Eigener Algorithmus, aufbauend auf dem SuperMemo-Algorithmus von Piotr Woźniak

Den Algorithmus zur Planung der Wiederholungszeitpunkte möchte ich statt wie ursprünglich vorgesehen nicht auf dem Leitner-Algorithmus, sondern auf dem Supermemo-Algorithmus von Piotr Woźniak aufbauen. Nach eingehender Prüfung bin ich zum Schluss gekommen, dass dieser für meine App besser geeignet ist. Das Bewerten der Schwierigkeit der Antworten bedeutet beim Lernen zwar einen kleinen Mehraufwand, allerdings können die Lernzeitpunkte dadurch besser geplant und unnötige Wiederholungen vermieden werden,

wodurch im Endeffekt Zeit gespart wird. Unter Einbezug der Gedächtnisstufen soll ein eigener Algorithmus entwickelt werden, der die Bewertungen berücksichtigt.

Benachrichtigungen
Denkbar wäre es auch, eine Benachrichtigungsfunktion hinzuzufügen, sodass der Nutzer zum richtigen Zeitpunkt eine Erinnerung als Push-Notification erhält. Dies könnte auch durch eine Funktion zum Eingeben eines Prüfungstermins erweitert werden. Aufgrund des Datums kann die App dann einen Lernplan erstellen und immer benachrichtigen, wann wieder gelernt werden sollte.

Lockscreen
Eine weitere Idee zur Abfrage bestünde darin, einen eigenen Lockscreen zu programmieren. Jedes Mal wenn der Nutzer das Handy entsperrt, müsste dieser dann zum Entriegeln die Übersetzung eines Wortes eingeben.

D. Lerntipps

Gewonnenes Wissen dem Nutzer durch kurzweilige Lerntipps zur Verfügung stellen

Durch den Lernforschungsteil und die Literaturrecherche habe ich viel neues nützliches Wissen zum Thema Lernen hinzugewonnen – auch Wissen welches nicht im vorherigen Teil erwähnt wurde. Vieles davon hat mir beim Lernen auf Prüfungen sehr geholfen. Dieses Wissen möchte ich den Nutzern der App weitergeben, jedoch nicht durch trockene Theorielektüre, sondern in Form von kurzweiligen Lerntipps.

Einerseits könnten Tipps während des Lernens angezeigt werden. Zum Beispiel könnte der Nutzer nach langem ununterbrochenem Lernen über einen Hinweis zu einer Pause aufgefordert werden (Abb. 11).

Andererseits soll es in der App einen eigenen Bereich *Lerntipps* geben, wo aktuelle Inputs zum Thema Lernen angezeigt werden. Mögliche Themenbereiche könnten sein:

- Loki-Methode sowie weitere Mnemo-Techniken
- Tipps zum Schaffen einer angenehmen Lernumgebung
- Pausen
- Mindmapping-Techniken

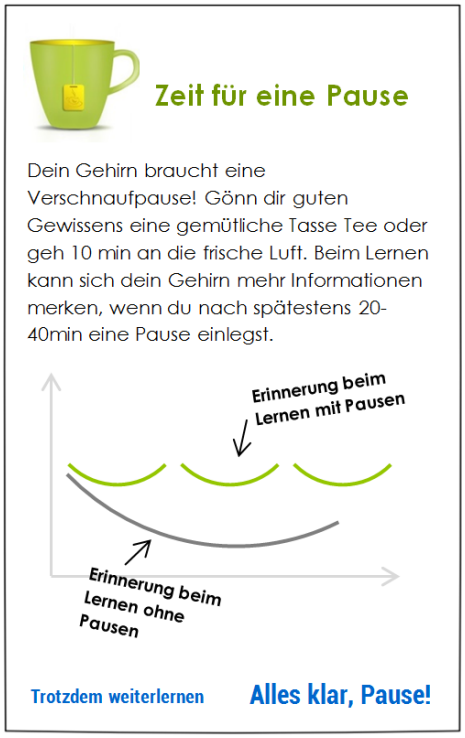


Abb. 11: Beispiel für einen Lerntipp

Es werden Inputs angezeigt, welche das Lernen allgemein betreffen und nicht nur das Lernen von Vokabeln.

Durch die Lerntipps wird persönlicher mit dem Nutzer interagiert, die Lernapp wird zum Lernbegleiter. Durch die nützlichen Hinweise wird der Nutzer während seines persönlichen Lernprozesses unterstützt.

E. Motivation

Durch Belohnungen, Spielerisches Lernen, sozialer Interaktion mit anderen Nutzern und Humor die Motivation beim Lernen steigern

Emotionen spielen für den Lernerfolg und für die Motivation beim Lernen eine grosse Rolle. Durch verschiedene Massnahmen können die Emotionen positiv beeinflusst werden und der Erfolg gesteigert werden.

Belohnungen
Durch positives Feedback soll der Nutzer beim Lernen motiviert werden. So könnte der Nutzer durch eine Meldung gelobt werden, wenn er sein selbst gestecktes Tagesziel erreicht hat. Ausserdem könnte man ein System mit Achievements einführen wie es auch in Games verwendet wird. Zum Beispiel könnte der Nutzer eine Auszeichnung erhalten, wenn er es schafft eine gewisse Anzahl Tage am Stück täglich zu lernen.

Interaktion mit anderen Nutzern
Dem Nutzer soll es möglich sein Lerngruppen beizutreten. Nutzer könnten sich dadurch gegenseitig beim Lernen unterstützen und motivieren.
Es soll dem Nutzer freigestellt werden seine persönlichen Leistungen mit denen anderer zu vergleichen. Achievements und erreichte Punkte können in Bestenlisten widergespiegelt werden. Ein zu überbietender Level eines anderen Nutzers kann Ansporn sein regelmässig zu lernen. Ausserdem könnten auch Achievements eingeführt werden, die nur im Teamwork erreicht werden können. So kann Gruppendynamik positiv genutzt werden.

Spielerisches Lernen
Durch Spiele kann das Lernen abwechslungsreicher gestaltet werden und damit der Spass und die Konzentrationsfähigkeit gesteigert werden. Zum Beispiel könnten Vokabeln mit Spielen wie Hangman, Kreuzworträtsel oder Wörter verbinden abgefragt werden.

Humor
Humor wirkt sich ebenfalls positiv auf den Lernerfolg aus. Er trägt zum Spannungsabbau bei und steigert das Erinnerungsvermögen[11]. Als Aufmunterung könnte nach langem Lernen ein lustiges Bild eingeblendet werden. Zum Beispiel könnten Witze zum Thema Sprachen oder Lehrerwitze angezeigt werden.

3.2 Probleme herkömmlicher Vokabel-Lernapps und Lösungen

Inzwischen gibt es auf dem Markt für Lernsoftware eine grosse Anzahl verschiedener Applikationen, welche mitunter sehr viele nützliche Funktionen anbieten und auch Teile der geschilderten Ideen schon implementieren. Allerdings gibt es auch Probleme, welche praktisch allen[12] Applikationen gemein sind. Wie die Interviews gezeigt haben, bestehen insbesondere zwei grosse Hemmnisse zur Verwendung digitaler Lernsoftware, auf welche ich hier eingehen und Lösungen vorschlagen möchte.

3.2.1 Hinzufügen neuer Wörter

Bei den meisten Vokabeltrainern müssen die Vokabeln zuerst mühselig hinzugefügt werden. Einige Vokabeltrainer bieten inzwischen zwar die Möglichkeit zur kollaborativen Eingabe, das Hauptproblem aber bleibt: Bevor mit dem eigentlichen Lernen begonnen werden kann, müssen neue Vokabeln erstmals eingetippt werden, was meist sehr zeitintensiv ist. Aus persönlicher Erfahrung bringt dieses blosse Abtippen fürs Einprägen noch sehr wenig. Wird nicht nochmals einige Zeit fürs Kontrollieren eingesetzt, arbeitet man mit der Unsicherheit, ob die Wörter, die man lernt auch korrekt erfasst wurden. Etwaige Tippfehler werden mitgelernt. Manche Vokabeltrainer bieten die Möglichkeit, von anderen Nutzern erstellte Listen zu importieren. Auch hier besteht jedoch wieder das Problem der ungewissen Vollständigkeit und Rechtschreibfehlern.

Um das Problem zu umgehen, habe ich mir zwei verschiedene Lösungen überlegt:

---

[11] Nielson, Kristy: *Positive and negative sources of emotional arousal enhance long-term word-list retention when induced as long as 30 min after learning* https://pdfs.semanticscholar.org/b663/9c8e9a8207d44482afa82ba31c1970b6ac53.pdf

[12] Ich bin bis zum Zeitpunkt des Schreibens dieser Maturaarbeit noch auf keine Applikation gestossen, welche benutzerfreundliche Lösungen für die beiden angesprochenen Probleme anbietet.
(Getestet wurden u. a. card2brain, Quizlet, E-Vocabulary, Anki, AnkiDroid, Memrise, duoLingo, Babbel, busuu, Semper, PrismaCards, CoboCards, dict.cc / Pons / Langenscheidt Vokabeltrainer, …)

Vernetzung mit Wörterbuch

Um eine schnelle Eingabe zu ermöglichen und die Richtigkeit der Wörter sicherzustellen, möchte ich die Lernapp eng mit einem Wörterbuch verflechten. Neue unbekannte Wörter können auf diese Weise beim erstmaligen Nachschlagen direkt zum Lernvokabular hinzugefügt werden. Diese Methode eignet sich besonders bei selbständiger Aneignung einer Fremdsprache. Im Vergleich zum klassischen Vokabelheft, wo neue Wörter handschriftlich aus dem Wörterbuch kopiert werden, kann einige Zeit gespart werden: Mit nur einem Klick kann neues Vokabular hinzugefügt werden. In vielen Fällen hat man zudem das Vokabelheft nicht zur Hand – das Smartphone jedoch schon.

Auch wenn das Vokabular, wie in der Schule üblich, in Form von Listen vorliegt, bietet die Verknüpfung mit einem Wörterbuch Vorteile. Durch Auto-Vervollständigung kann das Abtippen deutlich beschleunigt werden. Während des Eintippens kann die App im Hintergrund das Wörterbuch durchsuchen und Vorschläge für das aktuelle Wort sowie Übersetzungen liefern, die mit einem Tipp ausgewählt werden können. Falls Redewendungen oder ganze Sätze eingegeben werden, die in dieser Form nicht im Wörterbuch gespeichert sind, können die einzelnen Wörter auf Rechtschreibfehler überprüft werden.

Texterkennung

In vielen Apps wird heutzutage die Kamera zur einfacheren Texterfassung verwendet. Für eine App zum Lernen von Vokabeln liesse sich dieses Konzept ebenfalls zunutze machen. Statt abtippen liessen sich Vokabeln so durch einfaches Fotografieren hinzufügen. Eine der besten Texterkennungssoftwares (Tesseract) ist frei im Internet verfügbar (Open-Source) und kann auch in mobile Applikationen eingebunden werden. Kombiniert mit einer automatischen Fehlerkorrektur mithilfe der hinterlegten Wörterbuchdaten liesse sich so eine sehr zuverlässige und schnelle Methode zum Hinzufügen ganzer Vokabellisten schaffen.

3.2.2 Vernetzung mit analogen Lernmethoden

In den meisten Vokabel-Lernapps wird das digitale Lernen nicht oder nur rudimentär mit klassischen Lernmethoden verknüpft. Insbesondere scheint eine klare Trennung von digitalem zu klassischem Lernen zu bestehen: Entweder lernt man analog und handschriftlich oder man lernt digital mit Eintippen. Ich habe noch niemanden kennen gelernt, der zur Vorbereitung einer Prüfung beide Methoden mischt. Eine solche Trennung ist jedoch eigentlich nicht nötig. Es wäre mit der heutigen Technik sehr wohl möglich analoges und digitales Lernen zu verknüpfen. Die Vorteile der einen Methode könnten mit den Vorteilen der anderen verbunden werden. Im Folgenden zeige ich einige Ideen wie diese Vernetzung funktionieren könnte:

Immer mehr Drucker sind heutzutage WLAN- oder Bluetooth-fähig, sodass Dokumente vom Handy aus direkt über Funk an den Drucker geschickt werden können. In der App könnte man daher eine Funktion zum Ausdrucken von Wörtertests hinzufügen. Auf diese Weise können Wörter, die digital in der App gespeichert sind, auch handschriftlich gelernt werden. Jedoch bringt es wenig, wenn diese Wörterlisten nur eine Sprache enthalten. Der Nutzer kann die Wörter dann zwar aufschreiben, muss zum Kontrollieren jedoch wieder jedes Wort einzeln zeitaufwändig nachschauen. Praktisch wäre es deshalb, eine Lösungsleiste anzufügen, welche man nach hinten wegfalten kann (Abb. 12). So kann der Nutzer zuerst aus dem Gedächtnis alle Übersetzungen aufschreiben, anschliessend die Lösungsleiste zurückfalten und mit einem roten Stift seine Lösungen korrigieren.

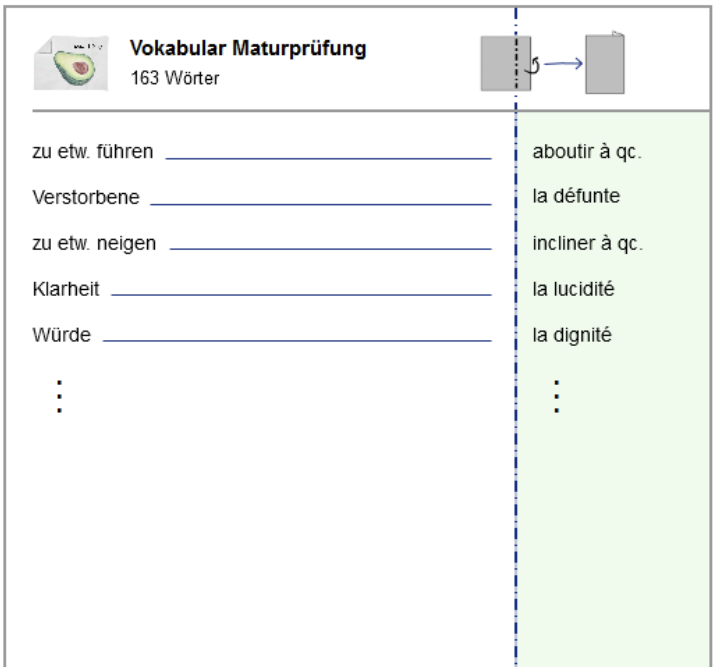


Abb. 12: Wörterliste mit Lösungsleiste zum Wegfalten

Gegenüber dem handschriftlichen Anfertigen solcher Listen hat das Ausdrucken den Vorteil, dass beliebig viele Exemplare gedruckt werden können. Eine unübersichtliche Vokabularseite aus einem Lehrbuch könnte mittels der Texterkennungsfunktion einfach importiert werden und anschliessend praktische Wörterlisten ausgedruckt werden.

Auch spielerische Übungen wie Kreuzworträtsel könnten gedruckt werden.

Ausserdem könnte eine Funktion hinzugefügt werden, mit der gezielt Listen mit Wörtern zusammengestellt werden können, mit denen der Nutzer noch Mühe hat. Die App kann dazu die Informationen über die Schwierigkeit der Vokabeln nutzen. Wenn der Nutzer Wörter nicht mit der App gelernt hat und daher noch keine Informationen vorliegen, könnte die App zum Auswählen der schwierigen Vokabeln statistische Daten anderer Nutzer heranziehen.

Während man zuhause mit der handschriftlichen Methode und mit ausgedruckten Wörterlisten lernen könnte, könnte man unterwegs die Vokabeln mobil auf dem Handy lernen. Je nach Arbeitsort können beide Methoden angewandt werden.

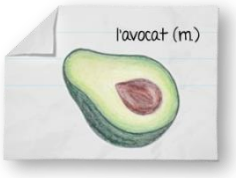
4 Programmierung

4.1 Einleitung

Bei der Programmierung musste ich mich für eine Auswahl von Funktionen entscheiden. Die Implementierung aller Ideen hätte den zeitlichen Rahmen der Maturaarbeit gesprengt.

Nach knapp 100 Stunden Programmieraufwand und mehr als 3000 Zeilen[13] geschriebenem Quellcode unterstützt die App nun folgende Features:

- Schnelles Nachschlagen von Wörtern mit Auto-Vervollständigung im Französisch-Deutsch-Wörterbuch (dict.cc, 68'000 Einträge) (offline)
- Hinzufügen von gefundenen Wörtern zum Lernvokabular durch nur einen Tipp
- Lernmodus mit Möglichkeit zur Bewertung der Schwierigkeit der Vokabel
- Individuelle Planung des optimalen Wiederholungszeitpunkts der Vokabel mit weiterentwickeltem SuperMemo-2-Algorithmus
- Verwalten der Vokabeln
  - Eigene Ordner erstellen und Vokabeln einsortieren
  - Ordner umbenennen und löschen
  - Lernintervall der Vokabel zurücksetzen
  - Vokabeln löschen



Vocado

Abb. 13: Logo und Name der App

Viele der im letzten Teil gewonnenen Ideen gehören schon zu einer erweiterten Funktionalität. Damit die App praxistauglich ist, müssen jedoch zuerst einmal die Grundfunktionen funktionieren. Nebst diesen habe ich mich für die aus meiner Sicht wichtigsten Funktionen, die Verknüpfung mit dem Wörterbuch sowie die Implementierung des SuperMemo-2-Algorithmus, entschieden. In Zukunft möchte ich jedoch noch mehr der skizzierten Ideen umsetzen.

[13] Bei der Zahl 3000 sind nur die Java-Klassen einberechnet (ohne Kommentar- und Leerzeilen gerechnet). Berechnet man die XML-Files für das Layout mit ein ergibt sich eine höhere Zahl.

4.2 Modulplanung

Da ich zu Beginn des Programmierens noch nicht wusste, welche Teile wie viel Zeit in Anspruch nehmen würden und wo ich eventuell auf Probleme stosse, bin ich modulartig vorgegangen. Ziel war es, die Grundfunktionen zu implementieren und – sofern Zeit vorhanden – aufbauend auf diesen weitere Funktionen hinzuzufügen. Folgende Tabelle zeigt die programmierten Module und deren Reihenfolge.

1. Basic Benutzerober-fläche		In diesem Modul wird ein neues App-Projekt angelegt, die Arbeitsumgebung eingerichtet, grundsätzliche Entscheide (wie unterstützte Hardware, Android-Version) getroffen, sowie grundlegende Funktionen der Benutzeroberfläche programmiert.
1.1 NavigationDrawer		Programmieren des Grundlayouts mit seitlicher Navigationsleiste.
1.2 SearchView		Programmieren eines Suchfelds für die Wörterbuchsuche innerhalb der Kopfleiste der App.
2. Grundgerüst		In diesem Modul wird der für den Benutzer nicht sichtbare Teil der Anwendung programmiert, (Hintergrundprozesse, Grundstrukturen zur Datenverarbeitung und Speicherung).
2.1 Klassenstrukturen		Entwurf und Programmierung eines objektorientierten Klassendesigns zur Verarbeitung von Vokabeldaten.
2.2 Datenbank aufsetzen		Aufsetzen einer Datenbank mithilfe von SQLite, Klassen für asynchronen Zugriff programmieren.
2.3 Einlesen der Wörterbuchdateien		Einlesen einer .txt Datei mit allen Wörterbuchdaten in Datenbank, String Parsing (vorerst nur Unterstützung für deutsch-französisches Wörterbuch).
3. Benutzerober-fläche		Über das Grundgerüst wird ein User Interface gelegt – eine Benutzeroberfläche, mit welcher der User mit der App interagieren und auf die Daten zugreifen kann.
3.1 Dictionary		Programmieren der Wörterbuchsuche.

Zielsetzung Grund-funktionen →

3.2 UnitsOverview, UnitViewer		Programmieren der Ordnerverwaltung und der Ansicht.
3.3 Test		Programmieren des Testmodus, in dem Nutzer abgefragt wird. Vorerst noch ohne Scheduling-Algorithmus für Vokabeln.
3.4 BatchEditVocables		Nutzeroberfläche zum gleichzeitiger Bearbeiten/Verschieben mehrerer angewählter Vokabeln.
3.5 EditVocable		Nutzeroberfläche für Anzeige und Bearbeitung einzelner Vokabeln.
4. Erweiterte Funktionalität		In diesem Modul werden weitere Funktionen hinzugefügt und skizzierte Ideen implementiert.
4.1 Scheduling-Algorithmus		Planung der optimalen Wiederholungszeitpunkte mit dem SM-2-Algorithmus.

Stand Code-Freezing →

Die oben gelisteten Funktionen wurden bis zum Zeitpunkt des Code-Freezings (Ende Oktober 2016) programmiert. Mein gesetztes Ziel, die Implementierung aller Grundfunktionen, habe ich damit erreicht. Es konnten sogar noch ein paar weitere Funktionen hinzugefügt werden.

Die nächsten geplanten Schritte wären:

4.2 Print		Funktion zum Ausdrucken von Vokabellisten mit Lösung zum Wegfalten.
4.3 Export und Import		Funktion zum Exportieren und Importieren von Listen → .Möglichkeit zum Austausch mit anderen Nutzern.
4.4 verbesserte Sprachverarbeitung		Erkennung von Wortmerkmalen in verschiedenen Formen (z.B. la = une = (f.) = (f) = f. = f), Algorithmus zur Erkennung von ähnlichen Wörtern (für Wörterbuchsuche und als Vorbereitung auf Fehlerkorrektur bei Texterkennung).
4.5 manuelles Hinzufügen		Funktion zum manuellen Hinzufügen eigener Wörter oder Sätze, unterstützt durch Fehlerkorrektur und automatische Vervollständigung.
4.6 Bilderkennung		Unterstützung von Texterkennung zum Hinzufügen von Wörtern in Zusammenarbeit mit der Drittsoftware *Textfee* (Open Source).
4.7 Englisch-Deutsch Wörterbuch Unterstützung		Einlesen des Englisch-Deutsch-Wörterbuchs in die Datenbank, Unterstützung für englische Vokabeln programmieren.

4.3 Wahl des Betriebssystems

Bei der Entwicklung einer Applikation für Mobilgeräte stellt sich am Anfang zuerst die Frage, welches Betriebssystem unterstützt werden soll. Die Wahl der Plattform determiniert einerseits die Zielgruppe, andererseits die Entwicklungswerkzeuge. Ich habe mich in meinem Fall für Android entschieden, dies aus folgenden Gründen:

Wie die Abbildung 14 zeigt, dominieren im Bereich der mobilen Betriebssysteme Android und iOS. In der Schweiz hat iOS zwar auch einen vergleichsweise hohen Marktanteil, dieser ist jedoch trotzdem tiefer als der von Android.[14] Da ich mit meiner App später möglichst viele Personen erreichen möchte, fällt die Wahl auf Android.

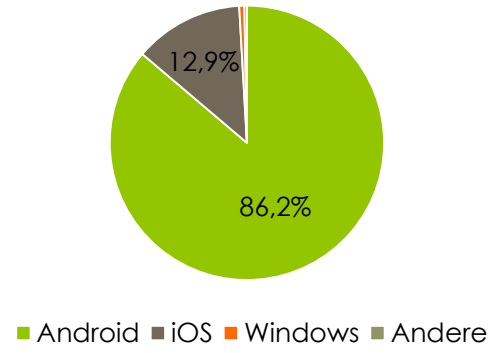


Abb. 14: Weltweiter Markanteil von mobilen Betriebssystemen
Daten: Gartner (August 2016)
http://www.gartner.com/newsroom/id/

Für mich persönlich das ausschlaggebendere Kriterium waren jedoch die Entwicklungswerkzeuge. Während sich Applikationen für iOS ausschliesslich auf Apple-Rechnern entwickeln lassen, ist Android nicht auf ein Betriebssystem fixiert, wodurch auch Windows-Rechner unterstützt werden. Da ich zuhause nur einen Windows-Computer zur Verfügung habe und mir keinen Apple-Computer zulegen möchte, fällt die Option iOS für mich aus Kostengründen weg.

4.4 Daten

Die App soll stark mit einem Wörterbuch verflochten werden. Damit dies funktioniert, braucht es jedoch zuerst eine Datengrundlage. Die meisten Wörterbücher sind proprietär und die Daten sind nicht frei verwendbar. Nach einiger Recherche habe ich mich für dict.cc entschieden, da es einen der grössten Wortschätze bietet und dieser frei verfügbar ist. Dict.cc ist ein offenes Wörterbuch, welches durch eine Community von Freiwilligen ständig erweitert wird. Auf der Website kann man eine durch Tabstopps separierte Liste (Abb. 15) herunterladen, die alle auf dict.cc verfügbaren Einträge umfasst.

[14] NZZ Infografik: http://images.nzz.ch/eos/v2/image/view/1240/-/text/inset/99f5b2ef/1.18563630/1434477009/wi.png

```
65548  œuvres·{f.pl}·complètes»Gesamtwerk·{n}·[gesammelte·Werke]»  noun¶
65549  œuvres·{f.pl}·[actes]»  Taten·{pl}» noun¶
65550  œuvre·{f}»  Arbeit·{f}» noun¶
65551  œuvre·{f}·d'art»Kunstwerk·{n}»  noun¶
65552  œuvre·{f}·de·bienfaisance·[ONG]»Hilfswerk·{n}·[NGO]»noun¶
65553  œuvre·{f}·rédemptrice»  Erlösungswerk·{n}»  noun¶
65554  œuvre·{f}»  Schöpfung·{f}·[Werk]»  noun¶
65555  œuvre·{f}»  Werk·{n}»  noun¶
65556  œuvre·{m}»  Gesamtwerk·{n}» noun¶
65557  œuvre·{m}»  Schaffen·{n}·[Werk]»noun¶
```

Abb. 15: Auszug der Liste von dict.cc

4.5 Grundlagen

4.5.1 Android Studio

Um dem Programmierer das Leben einfacher zu machen, verwendet man in der Regel Entwicklungsumgebungen (engl. *integrated development environment,* kurz *IDE*), die einem beim Programmieren viel mühsame Arbeit ersparen. *Android Studio* ist seit einigen Jahren die offizielle, von Google empfohlene IDE zur Entwicklung von Android-Applikationen. Zuvor kam der universalen Entwicklungsumgebung *Eclipse* diese Rolle zu. Android Studio ist eine sehr komfortable Entwicklungsumgebung, welche nebst Syntaxkorrektur und Auto-Vervollständigung beim Schreiben auch einen grafischen Editor für Benutzeroberflächen sowie viele nützliche Funktionen fürs Debugging (Fehlersuche) bietet. Eine Entwicklungsumgebung ist jedoch nicht zwingend notwendig, theoretisch liesse sich eine Android-App auch mithilfe eines ganz normalen Text-Editors schreiben - jedoch mit deutlich mehr Aufwand.

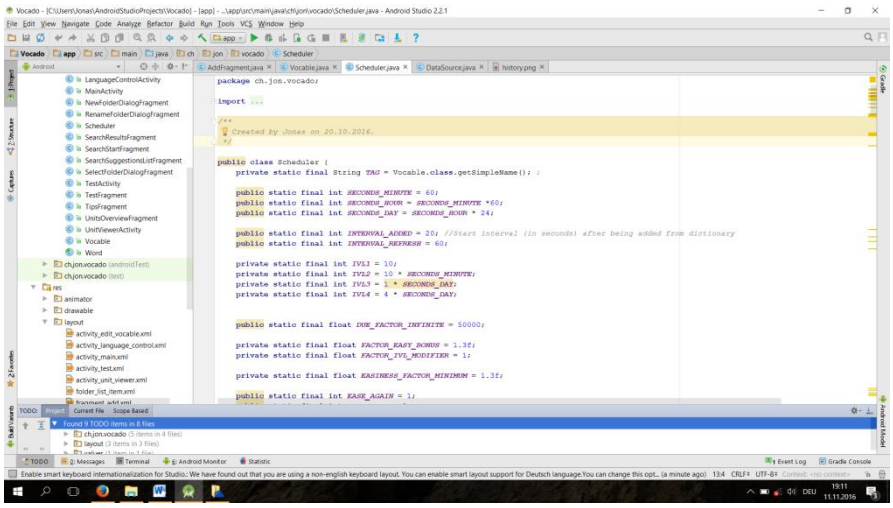


Abb. 16: Screenshot der Entwicklungsumgebung Android Studio

4.5.2 Objektorientierte Programmierung

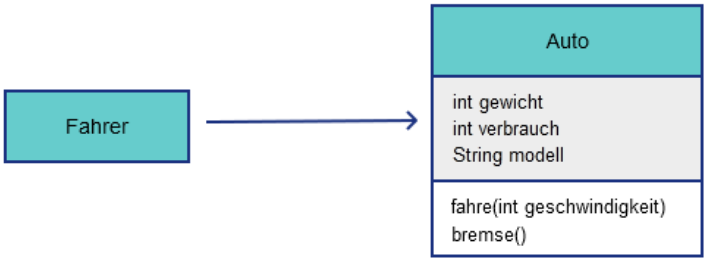
Android-Apps werden in Java programmiert – einer objektorientierten Programmiersprache. **O**bjekt**o**rientierte **P**rogrammierung (kurz OOP) ist ein Konzept, das in der Software-Entwicklung verwendet wird, um Programmiercode zu strukturieren. Auch bei grossen Programmen mit mehreren zehntausend Zeilen Code verliert man dank OOP den Überblick nicht. OOP ist der realen Welt nachempfunden, welche ebenfalls aus vielen Objekten besteht. Statt in einer einzigen Datei wird der Programmiercode auf Objekte verteilt. Jedes Objekt ist für einen bestimmten Teilbereich zuständig und kommuniziert über vordefinierte Methoden mit den anderen Objekten. In Java werden Objekte nicht direkt definiert, sondern über Klassen – einer Art Bauplan für Objekte. Eine Klasse kann entweder aus primitiven Datentypen bestehen oder aus anderen Objekten, welche schlussendlich jedoch ebenfalls aus primitiven Datentypen aufgebaut sind.

Abbildung 17 zeigt die gebräuchlichsten primitiven Datentypen.

Datentyp	Inhalt
`int` (kurz für Integer)	Ganzzahl speichert positive oder negative ganzzahlige Werte bis 2^{31}
`long`	Ganzzahl speichert positive oder negative ganzzahlige Werte bis 2^{63}
`float`	Fliesskommazahl speichert Kommazahlen mit normaler Genauigkeit
`double`	Fliesskommazahl speichert Kommazahlen mit hoher Genauigkeit
`boolean`	Logischer Wert kann entweder den Wert `true` (wahr) oder den Wert `false` (unwahr) annehmen
`String`	Zeichenfolge speichert z.B. Wörter und Sätze

Abb. 17: Auswahl primitiver Datentypen

Ein Beispiel zur Verdeutlichung:



Ein Objekt *Auto* wird (in einer vereinfachten Version) durch die Datenfelder *Gewicht*, *Verbrauch* und *Modell* charakterisiert. Ein Objekt *Fahrer* kann durch die vordefinierten und öffentlichen Methoden `fahre(geschwindigkeit)` und `bremse()` mit dem Auto-Objekt kommunizieren. Der Vorteil dieser objektorientierten Vorgehensweise liegt darin, dass der Fahrer das Auto bedienen kann, ohne die dahinterliegende Funktionsweise verstehen zu müssen. Im Hintergrund wird das Auto verschiedene private Methoden aufrufen, um z.B. die Einspritzung zu verändern und das Auto auf die gewünschte Geschwindigkeit zu beschleunigen. Oder es wird beim Bremsen automatisch die private Methode `bremslichtEinschalten()` aufgerufen. Der Fahrer braucht von allen diesen Vorgängen jedoch nichts zu wissen, er ruft einfach die Methode `fahre` bzw. `bremse` auf.

In Java definiert, sieht die oben beschriebene Klasse *Auto* wie folgt aus:

```
1.  public class Auto {
2.      private int gewicht;
3.      private float verbrauch;
4.      private String modell;
5.
6.
7.      // öffentlicher Konstruktor
8.      public Auto (int kg, float liter, String modell){
9.          this.gewicht = kg;
10.         this.verbrauch = liter;
11.         this.modell = modell;
12.     }
13.
14.     // öffentliche Methode
15.     public void fahre(int geschwindigkeit){
16.         ...
17.     }
18.
19.     // öffentliche Methode
20.     public void bremse(){
21.         bremslichtEinschalten();
22.         ...
23.     }
24.
25.     // private Methode
26.     private void bremslichtEinschalten(){
27.         ...
28.     }
29. }
```

Direkt vor dem Methodennamen steht der Rückgabetyp (z.B. int, boolean, etc). Liefert die Methode keinen Wert zurück, steht void.

Durch das Schlüsselwort *private* können Methoden und Datenfelder als privat gekennzeichnet werden. Sie können dann nur innerhalb der eigenen Klasse aufgerufen werden und sind für andere Klassen nicht ersichtlich.
Mit dem Schlüsselwort *public* gekennzeichnete Methoden sind öffentlich und können so auch durch andere Klassen aufgerufen werden.

Listing 18: Beispiel einer Java-Klasse

Durch das Aufrufen von

```
Auto meinAuto = new Auto(980, 4.3f, "Fiat 500");
```

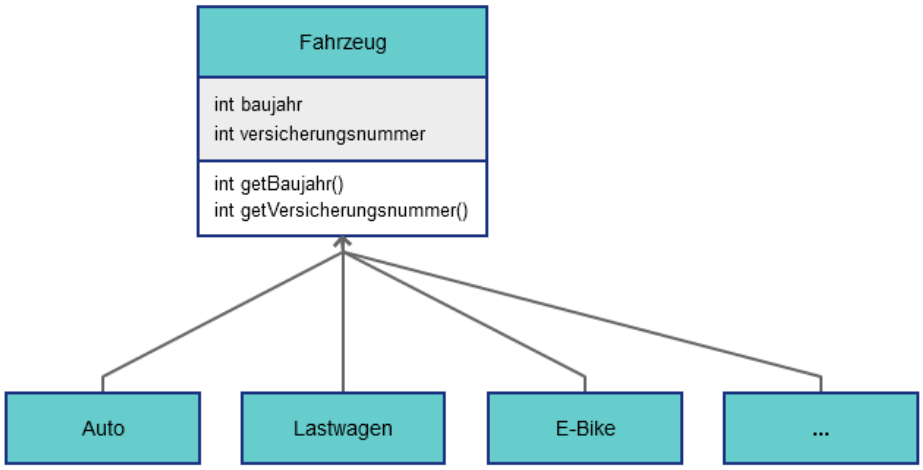
kann der Fahrer ein neues Auto-Objekt erstellen. Dadurch wird der sogenannte Konstruktor der Klasse *Auto* aufgerufen, der ein neues Auto-Objekt erzeugt und den Datenfeldern die übermittelten Werte zuweist. In diesem Fall 980kg, 4.3l/100km und die Modellbezeichnung „Fiat 500".
Durch den ersten Teil der Anweisung `Auto meinAuto` wird zuerst eine neue Variable erstellt, die Objekte des Typus `Auto` aufnehmen kann. Im zweiten Teil der Anweisung wird ein neues Auto-Objekt erstellt und anschliessend aufgrund des Gleichheitszeichens der Variable *meinAuto* zugewiesen.

Ein mögliches Fahrszenario könnte so aussehen:

```
meinAuto.fahre(50);
meinAuto.fahre(80);
meinAuto.bremse();
```

Zuerst fährt das Auto mit 50km/h, anschliessend beschleunigt es der Fahrer durch Aufruf der Methode `fahre(80)` auf 80km/h. Aufgrund eines Hindernisses bremst der Fahrer.

Ein weiterer Vorteil der objektorientierten Programmierung besteht darin, dass Klassen von anderen Klassen erben können. So müssen Klassen nicht jedes Mal von Grund auf neu programmiert werden.



So könnte eine Klasse *Fahrzeug* definiert werden, welche grundsätzliche Informationen wie Baujahr und Versicherungsnummer enthält. Die Klassen *Auto*, *Lastwagen*, *E-Bike*, … leiten sich von der Superklasse Fahrzeug ab, was durch das Schlüsselwort *extends* angegeben wird:

```
public class Auto extends Fahrzeug{
      ...
}
```

Die in *Fahrzeug* definierten Datenfelder und Methoden sind nun auch in den abgeleiteten Klassen verfügbar. Die Klasse *Auto* erweitert die Klasse *Fahrzeug*. So kann der Fahrer vom Auto-Objekt aus auch die Methode getBaujahr aufrufen:

```
int jahr = meinAuto.getBaujahr();
```

Statt Methoden der Superklasse einfach zu übernehmen, besteht auch die Möglichkeit, diese zu überschreiben und eine eigene Implementation anzubieten.

Nebst Klassen können auch sogenannte Interfaces definiert werden. Interfaces dienen dazu, gemeinsame Schnittstellen zu definieren. Sie enthalten selbst keinen ausführbaren Code, sondern definieren nur die Methodennamen. Interfaces können dann von Klassen implementiert werden, welche die definierten Methoden überschreiben.

Als Beispiel könnte man sich z.B. ein Interface *Unfallwarnung* denken, dass die Methode `achtungUnfall(double breitengrad, double längengrad)` definiert:

```
1. public interface Unfallwarnung {
2.     void achtungUnfall(double breitengrad, double längengrad);
3. }
```

Listing 19: Beispiel für ein Interface

Als Beispiel könnte die Klasse *Auto* das Interface implementieren. Dazu überschreibt es die Methode `achtungUnfall`.

```
1. public class Auto implements Unfallwarnung{
2.
3.     ...
4.
5.     @Override
6.     public void achtungUnfall(double breitengrad, double längengrad) {
7.        // Code der ausgeführt werden soll
8.         ...
9.     }
10. }
```

Listing 20: Klasse, welche das Interface *Unfallwarnung* implementiert

Nebst *Auto* können auch beliebige andere Klassen (auch nicht von *Fahrzeug* abgeleitete Klassen) das Interface *Unfallwarnung* implementieren. So könnte z.B. eine Klasse *Regionalzeitung* ebenfalls das Interface implementieren. Im Falle eines Unfalls können Klassen dann über die einheitliche Methode `achtungUnfall` benachrichtigt werden.

Der Vorteil von Interfaces besteht darin, dass Interfaces unabhängig von Vererbungshierarchien der Klassen eingesetzt werden können. Eine Klasse kann

immer nur von einer Superklasse erben, jedoch beliebig viele Interfaces implementieren.

4.5.3 Aufbau einer Android-App

Der sichtbare Teil der App besteht aus Activities und Fragments, welche hier kurz erklärt werden.

Activities

Eine Activity entspricht vereinfacht gesagt einer Bildschirmansicht. Jede Activity hat eine bestimmte Aufgabe, wie z.B. das Versenden einer Mail, Bearbeiten eines Kontaktes, Aufnehmen eines Fotos, etc.

Eine App besteht aus einer oder mehreren Activities, die über Schaltflächen miteinander verknüpft sind. So ruft ein Tippen auf einen Kontakt in der Kontaktliste eine neue Activity auf, die den Kontakt anzeigt. Klickt man in dieser Ansicht z.B. auf das Bearbeitungssymbol, so wird eine neue Activity aufgerufen, die es ermöglicht, den Kontakt zu bearbeiten (Abb. 21).

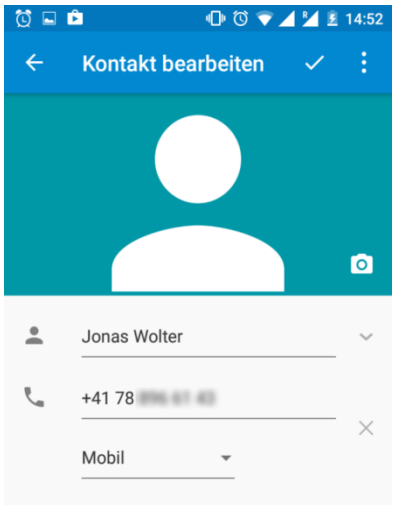


Abb. 21: Beispiel einer Activity

Wird der Zurück-Button des Handys betätigt, so wird einfach die vorhergehende Activity aufgerufen.

Bei Betätigung des Back-Buttons

Um eine eigene Activity zu programmieren, legt man eine neue Java-Klasse an, welche sich von der Superklasse *Activity* ableitet. Während des Lebenszyklus einer Activity werden zu bestimmten Zeitpunkten Methoden (Abb. 22) aufgerufen. Durch Überschreiben dieser Methoden lässt sich die Activity nach eigenen Wünschen modifizieren.

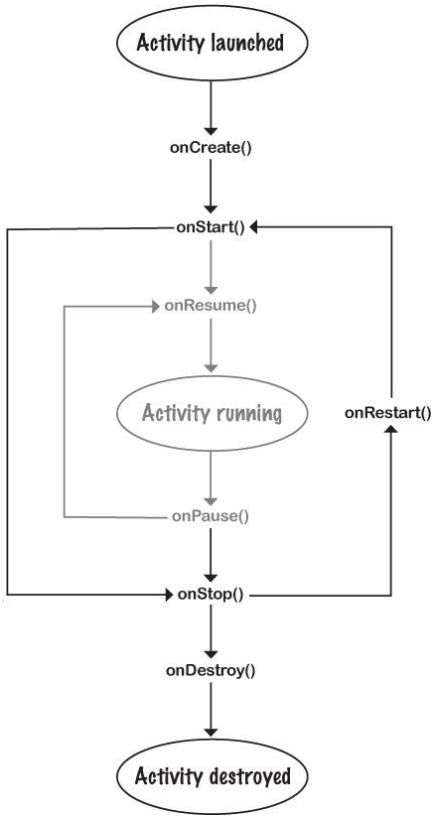


Abb. 22: Lifecycle einer Activity
Quelle: Griffiths, Dawn & David: *Head First Android Development*

```
1.  public class MyActivity extends Activity {
2.
3.      @Override
4.      protected void onCreate(Bundle savedInstanceState) {
5.          super.onCreate(savedInstanceState);
6.
7.          // eigener Code
8.          ...
9.      }
10. }
```

Listing 23: Eigene Activity

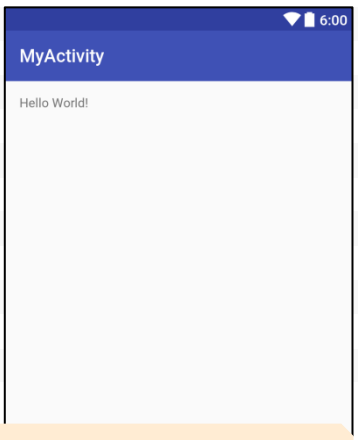
Listing 23 zeigt die überschriebene Methode `onCreate`. Diese wird häufig verwendet, um das Layout der Activity festzulegen sowie Daten wiederherzustellen.

Das Layout einer Activity wird durch einen Objektbaum verschiedener View-Objekte definiert. Dieser kann entweder direkt in Java, oder aber komfortabler in einem externen XML-Layout-File definiert werden.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_sample"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <TextView android:id="@+id/textView"
        android:text="Hello World!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"/>

</RelativeLayout>
```



Listing 24: Externes XML-Layout-File und dazugehörige Bildschirmansicht

Listing 24 zeigt den XML-Code für ein einfaches Layout, welches mittels einer TextView „Hello World!“auf dem Bildschirm anzeigt. Zur Laufzeit des Programms wird das XML-File mittels eines `inflaters` zu einem Objektbaum „aufgeblasen“:

```
inflater.inflate(R.layout.activity_sample, container, false);
```

Fragments

Fragments sind sozusagen Mini-Activities. Im Gegensatz zu Activities, welche immer den gesamten Bildschirm bedecken, können Fragments auch für Teilbereiche eingesetzt werden. Fragments haben den Vorteil, dass sie flexibler eingesetzt werden können. Abb. 25 zeigt ein Beispiel für ein flexibles Layout, wo Fragments je nach Bildschirmgrösse unterschiedlich eingesetzt werden.

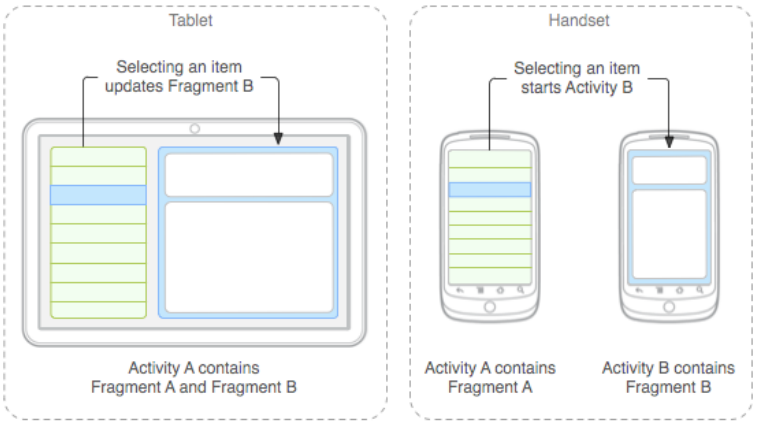


Abb. 25: Anwendungsmöglich-keiten von Fragments
Quelle: github.com

Programmiertechnisch ähneln Fragments Activities jedoch sehr. Sie durchlaufen einen ähnlichen Lebenszyklus, wie der von Activities. Im Normalfall gehört zu einem Fragment ebenfalls ein XML-File.

Ordnerstruktur

Die verschiedenen Komponenten einer Android-App sind über verschiedene Ordner verteilt. Abb. 26 zeigt, wo sich die wichtigsten Komponenten befinden.

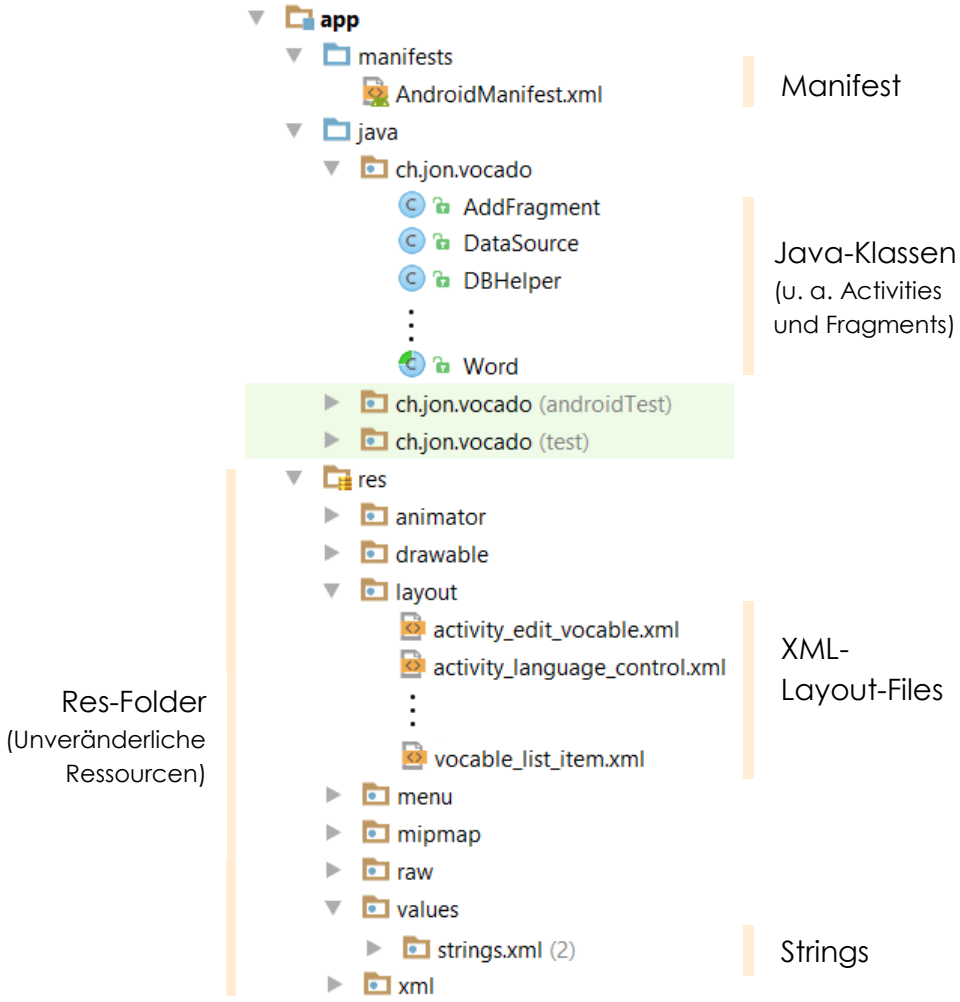


Abb. 26: (Vereinfachte) Ordnerstruktur einer Android-App

Jede App besitzt eine AndroidManifest-Datei. In dieser Datei müssen alle Activities der Applikation registriert werden. Ausserdem müssen erforderliche Berechtigungen (z.B. Internetzugriff, Ortungsdienste, etc.) hier angegeben werden.

Java-Klassen befinden sich im Package *ch.jon.vocado*, XML-Files für das Aussehen der Benutzeroberfläche im Res-Folder unter *layout*.

Statt Strings, welche auf der Benutzeroberfläche angezeigt werden sollen, direkt im Quellcode zu spezifizieren, besteht die Möglichkeit, diese in eine externe Datei namens *strings.xml* auszulagern.
Dies hat den Vorteil, dass die App so flexibel für Sprachanpassungen ist, da mehrere *strings.xml* Dateien für unterschiedliche Sprachen angelegt werden können. Android wählt dann automatisch je nach Nutzersprache die passende Datei aus.

Die Standardsprache meiner App ist Englisch. Die deutsche Benutzeroberfläche ist lediglich eine Übersetzung der englischen *strings.xml* Datei.

4.6 Übersicht

Bei der Programmierung meiner App habe ich auf ein striktes objektorientiertes Vorgehen geachtet, um eine einfache Erweiterbarkeit zu gewährleisten und die App für Änderungen flexibel zu machen. Aus demselben Grund sind Benutzeroberfläche und Datenverarbeitung strikt voneinander getrennt.

Auf der nächsten Seite befindet sich eine Übersicht aller selbst programmierten Java-Klassen (Abb. 27). Momentan mag diese noch etwas verwirren, im Folgenden werden die einzelnen Komponenten jedoch Schritt für Schritt erklärt, um ein Verständnis für die Funktionsweise zu erhalten. Die Übersichtsgrafik soll während des Lesens als Orientierungshilfe dienen.

Aufgrund des grossen Umfangs ist es jedoch nicht möglich, alle Komponenten gleich detailliert zu behandeln. Auf einige wichtige Komponenten gehe ich sehr genau ein, bei anderen beschränke ich mich auf die Benutzeroberfläche.

Auch wenn der abgedruckte Quellcode nicht verstanden wird, sollte es trotzdem möglich sein, einen Einblick in die Vorgänge zu erhalten, die sich im Hintergrund einer solchen Android-App abspielen.

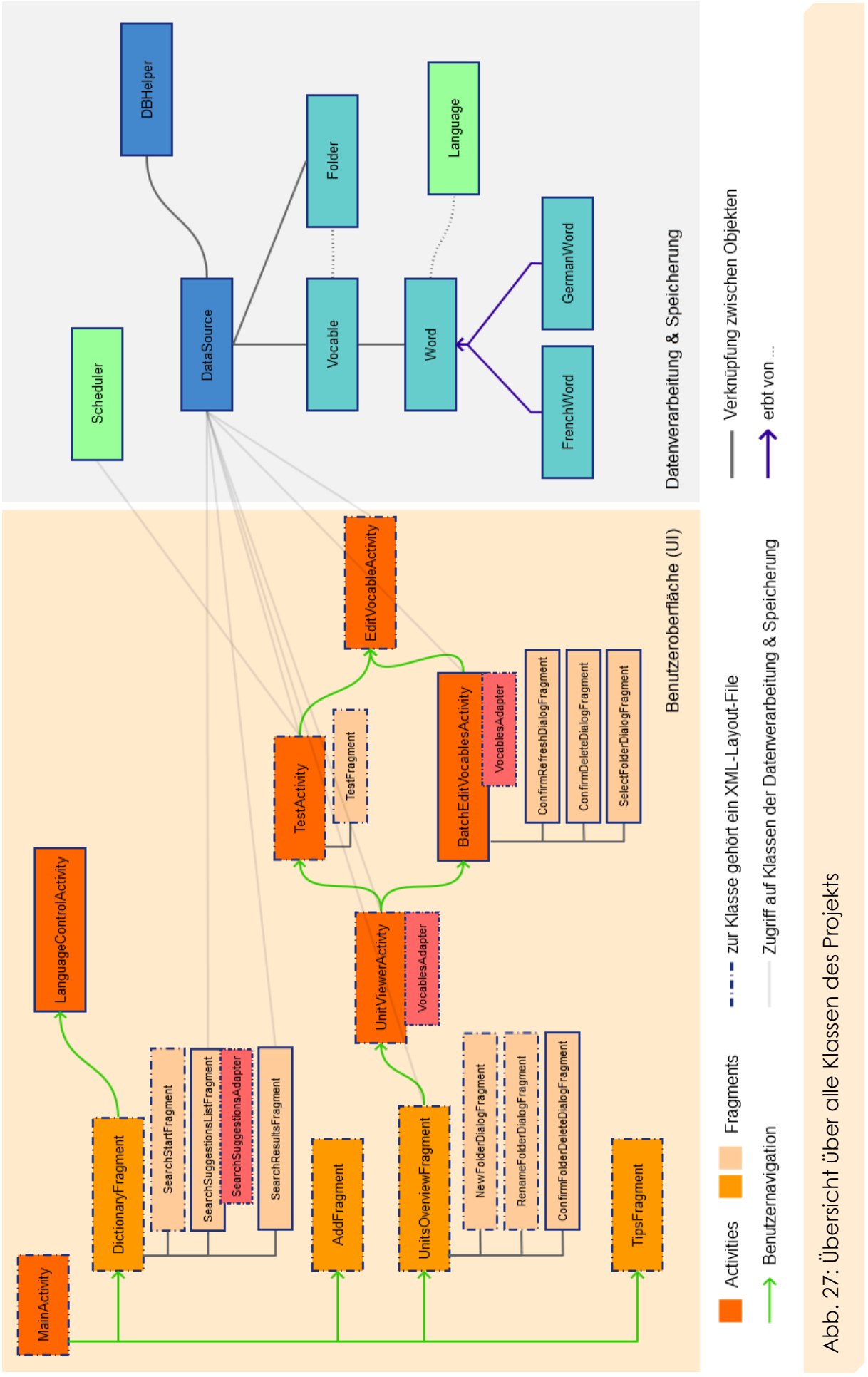


Abb. 27: Übersicht über alle Klassen des Projekts

4.7 MainActivity

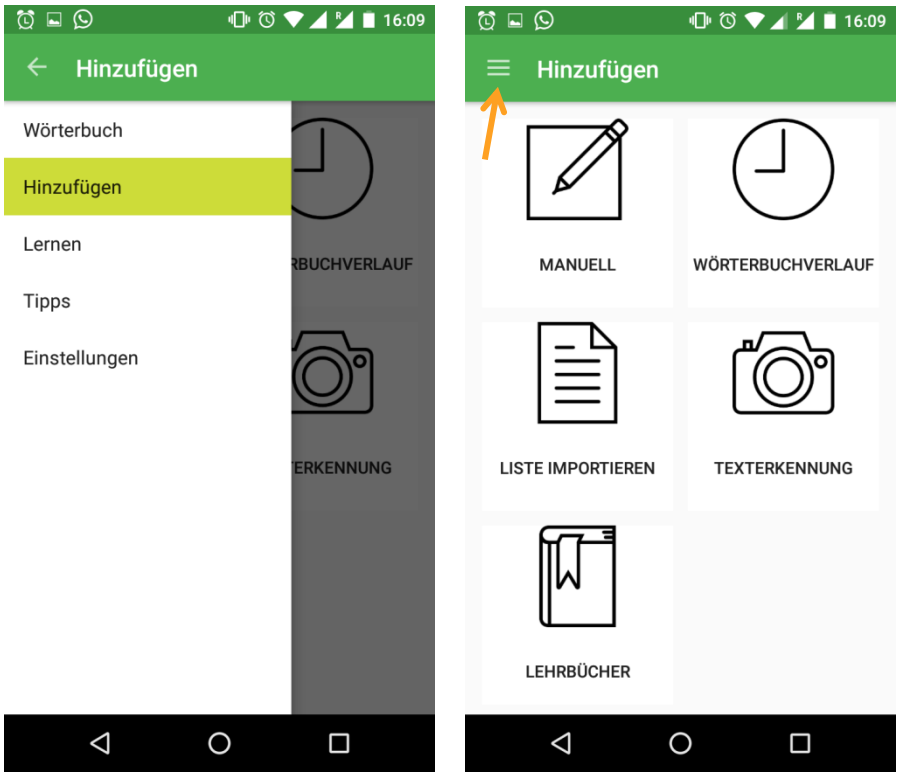


Abb. 28: geöffneter oder geschlossener NavigationDrawer

Die MainActivity wird beim Starten der App aufgerufen. Sie ist für die Navigation innerhalb der App zuständig. Von der Menuansicht gelangt man in die verschiedenen Bereiche der App:

- Unter dem Reiter ***Wörterbuch*** können Begriffe im Wörterbuch gesucht und das Suchergebnis durch einen Tipp hinzugefügt werden.
- Der Bereich ***Hinzufügen*** (auf Abb. 28 rechts zu sehen) bietet weitere Möglichkeiten zum Hinzufügen von Wörtern zum Lernvokabular an.
- Unter ***Lernen*** können die Wörter in Ordner organisiert und gelernt werden.
- Im Bereich ***Tipps*** werden dem Benutzer nützliche Informationen zum Thema Lernen angezeigt, in Form von kurzen Tipps
- Unter ***Einstellungen*** kann der Nutzer verschiedene Funktionen der App anpassen und Lernintervalle beeinflussen.

Die seitliche Navigationsbar öffnet sich durch Tippen auf den DrawerToggle (markiert durch orangen Pfeil) oder durch von links nach rechts streichen.

Dieses Verhalten wird über ein Widget namens *DrawerLayout* in Zusammenarbeit mit einem *ActionBarDrawerToggle* erreicht. Listing 29 zeigt das Layout File.

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </FrameLayout>

    <ListView
        android:id="@+id/drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#ffffff">
    </ListView>

</android.support.v4.widget.DrawerLayout>
```

Listing 29: activity_main.xml

Das Layout der MainActivity ist recht einfach gehalten: Das *DrawerLayout* enthält lediglich eine *ListView* und ein *FrameLayout*. Die *ListView* wird zum Anzeigen der Menueinträge verwendet, das *FrameLayout* dient als Platzhalter: Je nach Auswahl wird es durch verschiedene Fragments ausgefüllt. In Abb. 28 enthält es beispielsweise das *AddFragment*.

Listing 30 zeigt den Java-Code, der für das Funktionieren der MainActivity nötig ist:

```java
public class MainActivity extends Activity {

    private String[] drawerTitles;
    private ListView drawerList;
    private DrawerLayout drawerLayout;
    private ActionBarDrawerToggle drawerToggle;

    private int currentPosition = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //Layout der Activity festlegen
        setContentView(R.layout.activity_main);

        // Variabeln initialisieren um später darauf zugreifen zu können
        drawerTitles = getResources().getStringArray(R.array.drawer_titles);
        drawerList = (ListView) findViewById(R.id.drawer);
        drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);

        //ListView mit Einträgen füllen
        drawerList.setAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_activated_1, drawerTitles));

```

```
26.         //Drawer Click Listener erzeugen
27.         drawerList.setOnItemClickListener(new ListView.OnItemClickListener() {
28.             @Override
29.             public void onItemClick(AdapterView<?> parent, View view, int posit
    ion, long id) {
30.                 selectItem(position);
31.             }
32.         });
33.
34.         //Restore position state
35.         if(savedInstanceState != null){
36.             currentPosition = savedInstanceState.getInt("position");
37.         }
38.         selectItem(currentPosition);
39.
40.
41.         //Create ActionBarDrawerToggle
42.         drawerToggle = new ActionBarDrawerToggle(this, drawerLayout,
43.                 R.string.open_drawer, R.string.close_drawer){
44.             @Override
45.             public void onDrawerClosed(View view){
46.                 super.onDrawerClosed(view);
47.             }
48.             @Override
49.             public void onDrawerOpened(View drawerView){
50.                 super.onDrawerOpened(drawerView);
51.             }
52.         };
53.         drawerLayout.addDrawerListener(drawerToggle);
54.
55.         //Pfeil in ActionBar anzeigen
56.         getActionBar().setDisplayHomeAsUpEnabled(true);
57.         getActionBar().setHomeButtonEnabled(true);
58.
59.         // OnBackStackChangedListener
60.         getFragmentManager().addOnBackStackChangedListener(new FragmentManager.
    OnBackStackChangedListener(){
61.             @Override
62.             public void onBackStackChanged() {
63.                 Fragment visibleFragment = getFragmentManager().findFragmentByT
    ag("visible_fragment");
64.
65.                 if(visibleFragment instanceof DictionaryFragment){
66.                     currentPosition = 0;
67.                 }else if (visibleFragment instanceof AddFragment){
68.                     currentPosition = 1;
69.                 }else if (visibleFragment instanceof UnitsOverviewFragment){
70.                     currentPosition = 2;
71.                 }else if (visibleFragment instanceof TipsFragment){
72.                     currentPosition = 3;
73.                 }
74.                 // ...
75.                 drawerList.setItemChecked(currentPosition, true);
76.             }
77.         });
78.     }
79.
80.     // DrawerToggle benachrichtigen
81.     @Override
82.     protected void onPostCreate(Bundle savedInstanceState){
83.         super.onPostCreate(savedInstanceState);
84.         drawerToggle.syncState();
85.     }
86.
87.     // DrawerToggle benachrichtigen
88.     @Override
89.     public void onConfigurationChanged(Configuration newConfig){
```

```
90.         super.onConfigurationChanged(newConfig);
91.         drawerToggle.onConfigurationChanged(newConfig);
92.     }
93.
94.     // Position sichern
95.     @Override
96.     public void onSaveInstanceState(Bundle outState){
97.         super.onSaveInstanceState(outState);
98.         outState.putInt("position", currentPosition);
99.     }
100.
101.    // Aktualisiert den Hauptinhalt durch Austauschen von Fragments
102.    private void selectItem(int position){
103.        Fragment fragment;
104.        switch (position){
105.            case 0:
106.                fragment = new DictionaryFragment();
107.                break;
108.            case 1:
109.                fragment = new AddFragment();
110.                break;
111.            case 2:
112.                fragment = new UnitsOverviewFragment();
113.                break;
114.            case 3:
115.                fragment = new TipsFragment();
116.                break;
117.            default:
118.                fragment = new DictionaryFragment();
119.        }
120.
121.        //Fragment austauschen
122.        FragmentManager fm = getFragmentManager();
123.        fm.beginTransaction()
124.                .replace(R.id.content_frame, fragment, "visible_fragment")
125.                .addToBackStack(null)
126.                .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
127.                .commit();
128.        currentPosition = position;
129.
130.        //Drawer schliessen
131.        drawerLayout.closeDrawer(drawerList);
132.    }
133.
134.    @Override
135.    public boolean onOptionsItemSelected(MenuItem item){
136.        if (drawerToggle.onOptionsItemSelected(item)){
137.
138.            // Beim Öffnen des Drawers Keyboard schliessen
139.            View view = getCurrentFocus();
140.            if (view != null) {
141.                InputMethodManager imm = (InputMethodManager) getSystemService
    (Context.INPUT_METHOD_SERVICE);
142.                imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
143.            }
144.            return true;
145.        }
146.        return super.onOptionsItemSelected(item);
147.    }
148.}
```

Listing 30: MainActivity

Die Klasse MainActivity überschreibt die Lifecycle-Methode `onCreate`. In dieser wird das Layout eingerichtet sowie verschiedene Listener registriert. Die Listener-Interfaces werden direkt durch innere anonyme Klassen implementiert, weshalb das Schlüsselwort implements entfällt.

Listener

Die Verwendung von Listenern ist ein Konzept, welches in der objektorientierten Programmierung angewandt wird. In Klasse A tritt ein bestimmtes Ereignis (z.B. Click Event) auf, für dass sich Klasse X interessiert. Um Klasse X zu benachrichtigen könnte Klasse A direkt eine Methode der Klasse X aufrufen. Dies würde allerdings dem objektorientierten Konzept widersprechen, wo Klassen möglichst wenig übereinander wissen sollen. Um dies zu verhindern verwendet Klasse A ein Interface um mögliche Zuhörer (Listener) zu benachrichtigen. Klassen die sich für das Ereignis interessieren können das Interface implementieren und diese Implementierung anschliessend bei Klasse A über die Methode `set[…]Listener` registrieren. Im Falle des Auftretens des Ereignisses informiert Klasse A dann alle registrierten Listener.

Bei der *drawerList* registriert sie einen *OnItemClickListener* (27ff.). So wird die MainActivity benachrichtigt, falls ein anderer Menueintrag ausgewählt wurde. Tritt dieses Ereignis auf, wird die private Methode `selectItem(int position)` (102ff.) aufgerufen. In dieser spielt sich der wichtigste Teil ab: Anhand des übermittelten Parameters *position* (Position des angewählten Menu-Eintrags) wählt die Methode das passende Fragment aus. Über den *FragmentManager* führt es eine *FragmentTransaction* durch, wobei das im Platzhalter befindliche Fragment durch das neue Fragment ersetzt wird. Anschliessend wird die Navigation durch den Aufruf von `drawerLayout.closeDrawer(drawerList)` geschlossen.

Damit der *drawerToggle* richtig funktioniert muss dieser zu verschiedenen Zeitpunkten benachrichtig werden, was in Zeilen 80 – 92 passiert.

Der restliche Code dient vor allem dazu, dass sich die Activity in verschiedenen Situationen richtig verhält:

- Beim Drücken des Zurück-Buttons des Handy wird zwar das letzte Fragment angezeigt, allerdings wechselt der Menu-Eintrag nicht – es ist immer noch der vorherige markiert. Aus diesem Grund registriert die MainActivity einen *OnBackStackChangedListener* (60ff.) um auf das Drücken des Zurück-Buttons reagieren zu können. Durch die Abfrage `visibleFragment instanceof [...]` wird ermittelt, welches Fragment gerade angezeigt wird. Anschliessend wird der richtige Menueintrag manuell über den Aufruf von `drawerList.setItemChecked(currentPosition, true)` markiert.
- Beim Drehen des Handys ändert sich die Bildschirmauflösung (anderes Seitenverhältnis), weshalb die Activity neu initialisiert werden muss. Die aktuelle MainActivity wird zerstört und eine neue MainActivity erzeugt. Damit sich der Nutzer nach Drehen des Bildschirms immer noch an

gleicher Position in der App befindet, wird in `onSaveInstanceState(Bundle outState)` (96ff.) die aktuelle Position in einem Bundle gesichert. Die Lifecycle-Methode `onSaveInstanceState` wird kurz vor dem zerstören aufgerufen. Die neue MainActivity stellt in *onCreate* die Position aus dem Bundle wieder her. (35ff.)

- Beim Öffnen des Drawers soll eine möglicherweise offene Tastatur geschlossen werden, da diese ansonsten die Navigationsleiste überlappt. Dies wird durch den Code ab Zeile 139ff. erreicht.

4.8 Datenverwaltung

Die nachfolgenden Activities und Fragments müssen auf verschiedene Hintergrunddaten zugreifen. Deswegen möchte ich an dieser Stelle genauer auf die interne Datenverwaltung der App eingehen.

4.8.1 SQLite-Datenbank

Zur Speicherung der Appdaten arbeitet im Hintergrund eine lokale SQLite-Datenbank. Das Open-Source-Datenbanksystem SQLite ist bereits in Android integriert. SQLite funktioniert ähnlich wie serverbasierte Datenbanksysteme. Befehle werden ebenfalls in der Datenbanksprache SQL formuliert. Es hat gegenüber diesen jedoch den Vorteil, dass es sehr platzsparend arbeitet und schlank daher kommt, da jegliche Server-Software entfällt.

Android stellt verschiedene Klassen zum einfacheren Zugriff auf SQLite-Datenbanken bereit. Um eine eigene Datenbank zu erstellen, leitet man eine Klasse von SQLiteOpenHelper ab. In meiner App habe ich sie *DBHelper* genannt. Listing 31 zeigt den vollständigen Code dieser Klasse:

```
1.  public class DBHelper extends SQLiteOpenHelper{
2.
3.      private String TAG = DBHelper.class.getSimpleName();
4.
5.      public static final String DB_NAME = "vocado.db";
6.      public static final int DB_VERSION = 1;
7.
8.
9.      // Table properties
10.     // Table name
11.     public static final String TABLE_FRENCH_GERMAN = "french_german";
12.     //Columns
13.     public static final String _ID = "_id";
14.     public static final String FRENCH_ENTRY = "french";
15.     public static final String GERMAN_ENTRY = "german";
16.     public static final String FRENCH_STANDARD_FORMAT = "french_standard";
17.     public static final String GERMAN_STANDARD_FORMAT = "german_standard";
18.
19.     public static final String ADDED = "added";
20.     public static final String FOLDER_ID = "folder_id";
```

```
21.     public static final String SCHEDULED = "scheduled";
22.     public static final String STUDIED = "studied";
23.     public static final String EDITED = "edited";
24.     public static final String IMAGE_RESOURCE_ID =  "image_resource_id";
25.     public static final String E_FACTOR = "e_factor";
26.
27.
28.     //Folder Table
29.     public static final String TABLE_FOLDERS = "folders";
30.     public static final String FOLDER_NAME = "name";
31.
32.
33.
34.     DBHelper(Context context){
35.         super(context, DB_NAME, null, DB_VERSION);
36.     }
37.
38.     @Override
39.     public void onCreate(SQLiteDatabase db) {
40.         updateMyDatabase(db, 0, DB_VERSION);
41.     }
42.
43.     @Override
44.     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
45.         updateMyDatabase(db, oldVersion, newVersion);
46.     }
47.
48.
49.     private void updateMyDatabase(SQLiteDatabase db, int oldVersion,
50.                                   int newVersion){
51.
52.         // Create Database if it does not exist
53.         if (oldVersion < 1){
54.             // French-German Tabelle erstellen
55.             db.execSQL("CREATE TABLE " + TABLE_FRENCH_GERMAN + " ("
56.                     + _ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
57.                     + FRENCH_ENTRY + " TEXT, "
58.                     + GERMAN_ENTRY + " TEXT, "
59.                     + FRENCH_STANDARD_FORMAT + " TEXT, "
60.                     + GERMAN_STANDARD_FORMAT + " TEXT, "
61.                     + ADDED + " TEXT, "
62.                     + FOLDER_ID + " INTEGER, "
63.                     + SCHEDULED + " TEXT, "
64.                     + STUDIED + " TEXT, "
65.                     + EDITED + " TEXT, "
66.                     + IMAGE_RESOURCE_ID + " INTEGER, "
67.                     + E_FACTOR + " REAL);"
68.             );
69.
70.             // Import dictionary from file
71.             InputStream in = null;
72.             InputStreamReader isr = null;
73.             BufferedReader br = null;
74.
75.             try {
76.                 in = App.getContext().getResources()
77.                         .openRawResource(R.raw.french_german);
78.                 isr = new InputStreamReader(in);
79.                 br = new BufferedReader(isr);
80.
81.                 // String parsen und French-German Tabelle füllen
82.                 String line;
83.                 String delims = "[\t]+";
84.                 while ((line = br.readLine()) != null) {
85.                     line.trim();
86.                     String[] tokens = line.split(delims);
87.
```

```
88.                     FrenchWord frenchWord;
89.                     GermanWord germanWord;
90.
91.                     if (tokens.length == 3) {
92.                         frenchWord = new FrenchWord(
93.                                 tokens[0] + " {" + tokens[2] + "}");
94.                         germanWord = new GermanWord(
95.                                 tokens[1] + " {" + tokens[2] + "}");
96.                     }else {
97.                         frenchWord = new FrenchWord(tokens[0]);
98.                         germanWord = new GermanWord(tokens[1]);
99.                     }
100.
101.                    addNewLine(db, frenchWord.getOriginalString(),
102.                            germanWord.getOriginalString(),
103.                            frenchWord.getString(Word.FORMAT_STANDARD),
104.                            germanWord.getString(Word.FORMAT_STANDARD));
105.                }
106.            }catch (IOException t){
107.                Log.e(TAG, "updateMyDatabase: ", t);
108.            }finally{
109.                if (br != null){
110.                    try {
111.                        br.close();
112.                    }catch (IOException e){
113.                        Log.e(TAG, "updateMyDatabase: ",e);
114.                    }
115.                }
116.                if (isr != null){
117.                    try {
118.                        isr.close();
119.                    }catch (IOException e){
120.                        Log.e(TAG, "updateMyDatabase: ",e);
121.                    }
122.                }
123.                if (in != null){
124.                    try {
125.                        in.close();
126.                    }catch (IOException e){
127.                        Log.e(TAG, "updateMyDatabase: ",e);
128.                    }
129.                }
130.            }
131.
132.            // Folders Tabelle erstellen
133.            db.execSQL("CREATE TABLE " + TABLE_FOLDERS + " ("     //Name
134.                    + _ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
135.                    + FOLDER_NAME + " TEXT);"
136.            );
137.
138.            // Standard-Ordner hinzufügen
139.            ContentValues cV = new ContentValues();
140.            cV.put(FOLDER_NAME, App.getContext().getResources()
141.                    .getString(R.string.added_from_dictionary));
142.            db.insert(TABLE_FOLDERS, null, cV);
143.        }
144.
145.        // Update Database to a newer version
146.        if (oldVersion < 2){
147.
148.        }
149.    }
150.
151.    // fügt Zeile in French-German Tabelle ein
152.    private void addNewLine(SQLiteDatabase db, String frenchEntry,
153.                            String germanEntry, String frenchStandardFormat,
154.                            String germanStandardFormat){
```

```
155.
156.        ContentValues cV = new ContentValues();
157.        cV.put(FRENCH_ENTRY, frenchEntry);
158.        cV.put(GERMAN_ENTRY, germanEntry);
159.        cV.put(FRENCH_STANDARD_FORMAT, frenchStandardFormat);
160.        cV.put(GERMAN_STANDARD_FORMAT, germanStandardFormat);
161.
162.        cV.put(ADDED, "");
163.        cV.put(FOLDER_ID, 0);
164.        cV.put(SCHEDULED,"");
165.        cV.put(STUDIED, "");
166.        cV.put(EDITED, "");
167.        cV.put(IMAGE_RESOURCE_ID, 0);
168.        cV.put(E_FACTOR, 2.5);
169.
170.        db.insert(TABLE_FRENCH_GERMAN, null, cV);
171.    }
172.}
```

Listing 31: DBHelper

Die Klasse *DBHelper* überschreibt die beiden Methoden `onCreate` und `onUpgrade`. Beim erstmaligen Zugriff auf den DBHelper wird durch die Superklasse *SQLiteOpenHelper* die überschriebene `onCreate` Methode aufgerufen, welche zum Erstellen einer neuen Datenbank dient. Die Methode `onUpgrade` wird aufgerufen, wenn die Datenbank in einer älteren Version vorliegt und auf eine neue Version aktualisiert werden muss.

In meiner Implementierung rufen beide die private Methode `updateMyDatabase` auf, welche die Arbeit übernimmt. In dieser werden zwei Tabellen erstellt: Die Tabelle *french_german* (Z.55-68) für Wörterbuchdaten und die Tabelle *folders* (Z.133-136) für Vokabelordner.

Anschliessend wird über einen InputStream (Z. 76/77) die Datei *french_german.txt* eingelesen, welche sich im Res-Folder unter Raw befindet und die heruntergeladenen Wörterbuchdateien enthält. Die Textdatei (ein Bild ist in Kap. 4.4 Daten zu finden) wird Zeile für Zeile eingelesen. Durch den Befehl `line.split(delims)` wird die Zeile anhand der Tabstopps in mehrere Tokens zerlegt. Die restliche Formatierung wird mithilfe von FrenchWord und GermanWord-Objekten vorgenommen. Anschliessend werden die Daten in die *french_german* Tabelle geschrieben.

Da die Textdatei mehr als 68'000 Zeilen enthält, dauert es aufgrund dieses Vorgangs beim erstmaligen Start der App ca. 1-2 Minuten, bis die Datenbank verfügbar ist. Dies ist jedoch eine einmalige Angelegenheit, hinterher funktioniert der Zugriff schnell.

4.8.2 DataSource

Die Klasse *DataSource* ist der Dreh- und Angelpunkt für die appinterne Datenverwaltung.

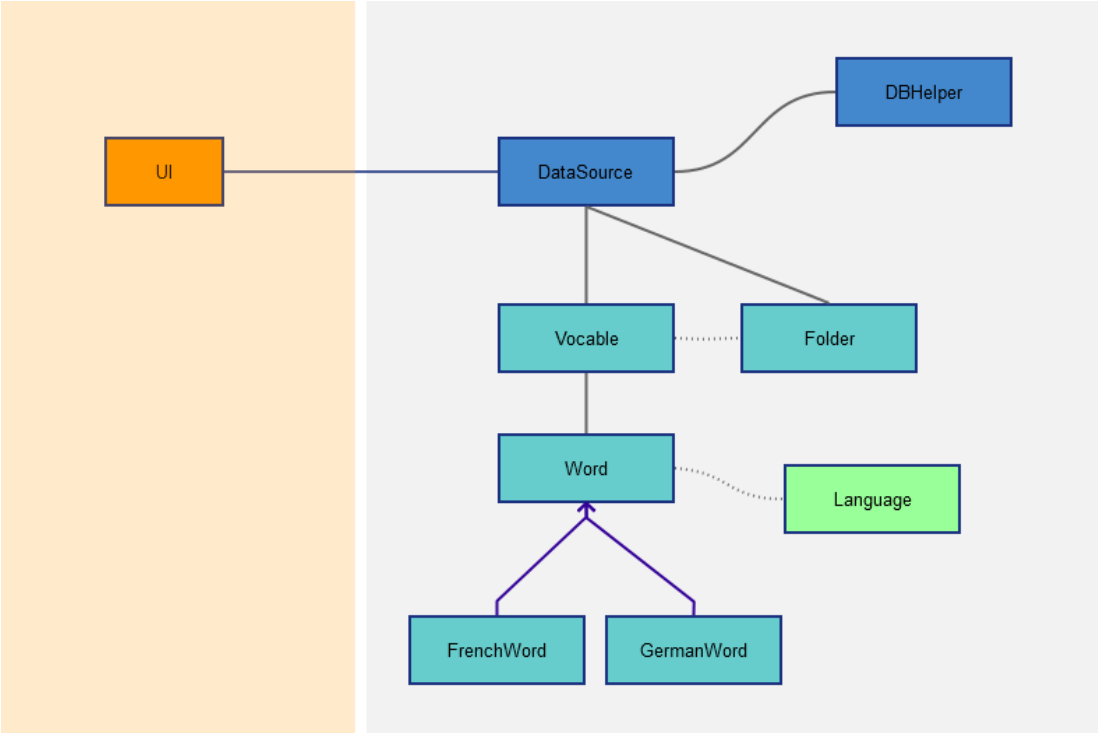


Abb. 32: Übersicht Datenverwaltung

Würden Klassen, welche für das User Interface zuständig sind (Activities oder Fragments) direkt Daten über DBHelper abfragen, gäbe es zwei Probleme:

- Ergebnisse würden in Form eines Tabelleneintrags zurückgeliefert, was für die Weiterverarbeitung sehr unpraktisch wäre.
- Bei grösseren Datenbankabfragen, kann es sein, dass diese ein paar Sekunden dauert. Während dieser Zeit friert die Benutzeroberfläche ein, was im schlimmsten Fall zu einem Absturz der App führen kann.

Thread

Threads werden zur parallelen Ausführung von Prozessen eingesetzt. Standardmässig läuft eine Android-App in einem Thread, dem UI-Thread. Es können jedoch weitere Threads hinzugefügt werden. Die Aufteilung von Programmcode auf mehrere parallel laufende Threads wird *Multithreading* genannt.

Wünschenswert wäre es, die Ergebnisse in Form von Objekten zu erhalten sowie die ganze Datenbankabfrage asynchron in einen Background-Thread auszulagern, sodass die Benutzeroberfläche reaktionsfähig bleibt. Aus diesem Grund habe ich die Klasse *DataSource* programmiert, welche beide Aufgaben übernimmt.

DataSource stellt verschiedene öffentliche Methoden bereit welche in Abbildung 33 gezeigt werden:

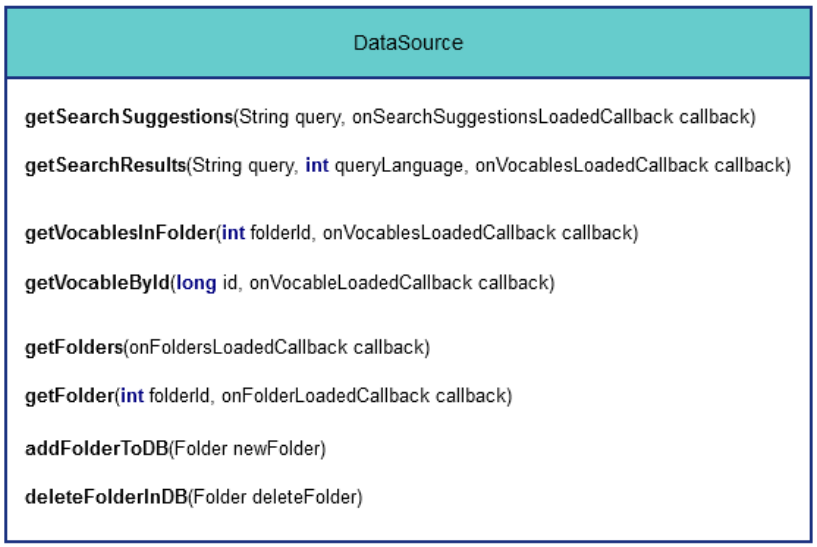


Abb. 33: öffentliche Methoden von DataSource

Alle diese Methoden haben den Rückgabetyp *void*, da die Daten nicht direkt sondern über ein Callback-Interface zurückgeliefert werden. Dies hat den Vorteil, dass die aufrufende Klasse nicht zu warten braucht, bis die Daten abgerufen wurden, sondern zu anderen Aufgaben übergehen kann. Sobald die Daten verfügbar sind, wird sie von DataSource über das Callback-Interface benachrichtigt.

DataSource selbst lagert die Datenbankabfrage mittels eines AsyncTask in einen seperaten Thread aus, davon braucht die Activity oder das Fragment jedoch nichts zu wissen.

DataSource erhält die Daten zuerst in Form von Tabelleneinträgen, bildet diese jedoch auf Vocable bzw. Folder Objekte ab.

Abb. 34 zeigt zwei Beispiele wie solche Tabellendaten aussehen.

_ID	FRENCH_ ENTRY	GERMAN_ ENTRY	FRENCH_ STANDARD_ FORMAT	GERMAN_ STANDARD_ FORMAT
13'043	comparution {f} {noun}	Erscheinen {n} {noun}	comparution	Erscheinen
30'216	incliner qc. {verb}	etw. neigen {verb}	incliner qc.	etw. neigen

_ID	ADDED	FOLDER_ID	SCHEDULED	STUDIED	EDITED	IMAGE_ RESOURCE_ ID	E_FACTOR
13'043	2016-11-02 12:42:17	7	2016-11-08 12:05:25; 2016-11-09 13:00:29	2016-11-08 11:55:25; 2016-11-08 13:00:29	-	0	2.8
30'216	2016-11-05 17:40:41	7	2016-11-05 17:39:46; 2016-11-05 18:27:29;	2016-11-05 18:27:19; 2016-11-05 18:27:41;	-	0	2.65

Abb. 34: Tabelleneinträge in French-German Tabelle

In den Kolonnen *FRENCH_ENTRY* und *GERMAN_ENTRY* befinden sich die vollständigen Einträge, wie sie im Wörterbuch gespeichert sind. *FRENCH_STANDARD_FORMAT* und *GERMAN_STANDARD_FORMAT* enthalten die gleichen Wörter, jedoch ohne die ganzen Zusatzinformationen, die in geschweiften, eckigen oder normalen Klammern stehen.

ADDED bezeichnet das Datum, an dem eine Vokabel zum Lernvokabular hinzugefügt wurde, *FOLDER_ID* die ID des Ordners, in dem sie sich befindet. *SCHEDULED* gibt an, für welchen Lernzeitpunkt die Vokabel geplant wurde, *STUDIED* wann sie effektiv gelernt wurde. Unter *EDITED* sind allfällige Bearbeitungen eingetragen.

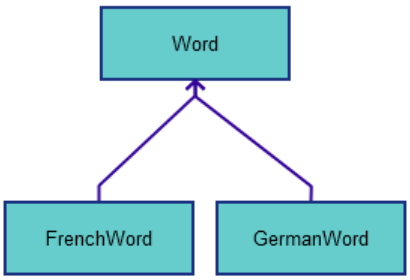
Gehört zu der Vokabel ein Bild, so wird dessen ID unter *IMAGE_RESOURCE_ID* eingetragen.

E_FACTOR gibt den berechneten Schwierigkeitsfaktor für die Vokabel an. Dieser wird zur Berechnung der Lernintervalle verwendet (dazu später mehr in Kapitel 4.13).

DataSource nimmt die Daten einer solchen Zeile und bildet sie auf ein Vocable-Objekt ab.

Ein Vocable-Objekt ist im Wesentlichen einfach eine Verknüpfung von einem Muttersprachenwort *firstLanguageWord* mit einem Fremdsprachenwort *foreignLanguageWord*. Nebst dieser Information enthält es Metadaten (oben beschrieben) über die Vokabel wie bspw. die Lerndaten.

Die Datenfelder *firstLanguageWord* sowie *foreignLanguageWord* enthalten je ein Objekt des Typus *Word*. Die Klasse Word ist jedoch abstrakt definiert, d.h. es können nicht direkt Objekte von der Klasse erzeugt werden. Ähnlich wie beim Einstiegsbeispiel zur objektorientierten Programmierung die Klasse *Fahrzeug*, gibt *Word* die Kategorie an. Word ist die Superklasse, von der dann die Klassen *FrenchWord* und *GermanWord* ableiten und die in Word definierten Methoden überschreiben.



Dieser Ansatz hat den Vorteil, dass ein Vocable-Objekt völlig unabhängig von der beinhalteten Sprache funktioniert. Wenn in Zukunft beispielsweise Englisch hinzugefügt werden soll, kann einfach eine weitere Klasse *EnglishWord* von Word abgeleitet werden.

Vocable ist es egal, in welcher Sprache die Wörter vorliegen, es kommuniziert einfach über die standardisierten, in Word festgelegten Methoden mit ihnen.

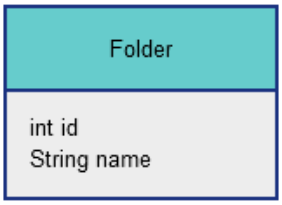
Die Verwaltung von Ordnern funktioniert ähnlich, sie sind aber weniger komplex als Vokabeln. Der Vollständigkeit halber erwähne ich sie hier auch kurz.

Alle Ordner sind in der Datenbank in der Folders-Tabelle abgelegt, die jedoch nur die zwei Kolonnen *_ID* und *FOLDER_NAME* enthält. Abb. 35 zeigt Beispieleinträge.

_ID	FOLDER_NAME
1	Aus Wörterbuch hinzugefügt
6	Unité 3
7	Mon vocabulaire

Abb. 35: Tabelleneinträge in Folders-Tabelle

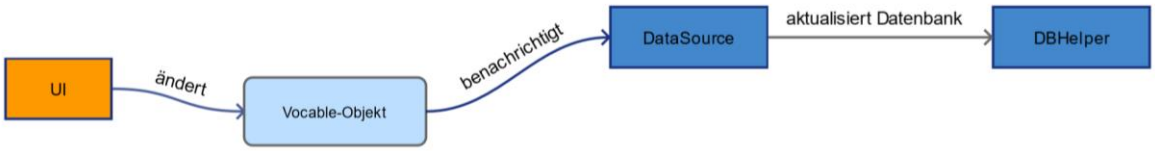
DataSource bildet diese Tabelleneinträge dann ebenfalls auf ein Objekt ab. Die Daten werden in Form von Folder-Objekten zurückgeliefert.



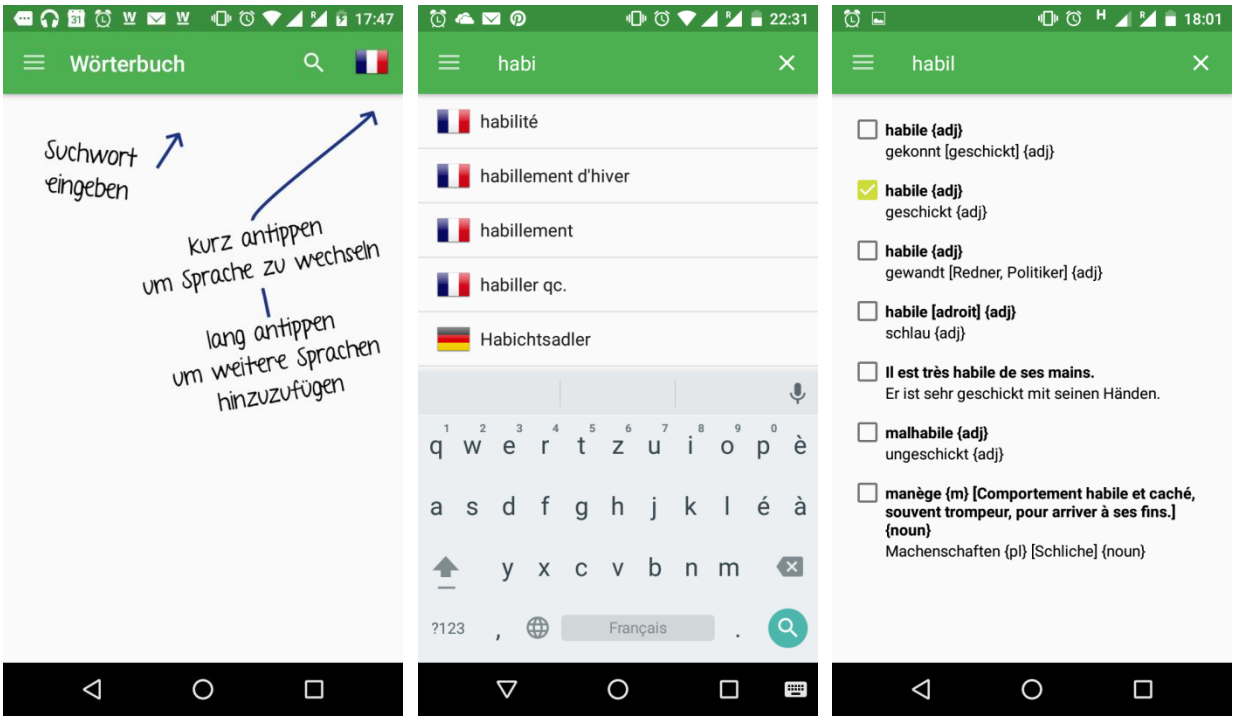
4.8.3 Daten ändern

Während des Erstellens der Vocable- oder Folder-Objekte registriert DataSource bei jedem erzeugten Objekt einen *onVocableDataChangedListener* bzw. *onFolderDataChangedListener*. Auf diese Weise wird DataSource benachrichtigt, wenn sich die Daten eines Objekts ändern. Tritt ein solches Ereignis ein, ruft DataSource die private Methode `updateVocableInDB` bzw. `updateFolderInDB` auf und schreibt die aktualisierten Daten zurück in die Datenbank.

Die Activity oder das Fragment, das Daten von DataSource empfängt, kann auf diese Weise direkt Daten in den Vocable- oder Folder-Objekten ändern, ohne sich Gedanken über die Synchronisation mit der Datenbank zu machen.



4.9 DictionaryFragment



SearchStartFragment **SearchSuggestionsListFragment** **SearchResultsFragment**

Abb. 36: Child-Fragments von DictionaryFragment

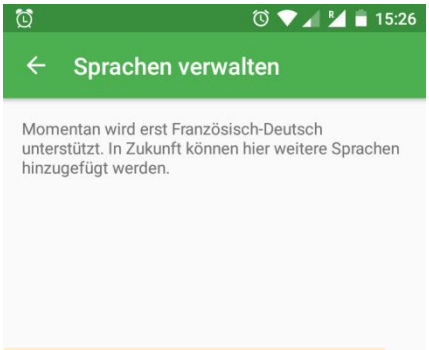


Abb. 37: LanguageControlActivty (Da die App momentan erst eine Sprache unterstützt, hat diese noch keine Funktion)

In der Wörterbuchansicht können Wörter gesucht und diese anschliessend direkt zum Lernvokabular hinzugefügt werden. Das *DictionaryFragment* selbst ist wiederum in drei Child-Fragments unterteilt.

Zuerst wird das *SearchStartFragment* angezeigt, welches dem Nutzer eine Hilfestellung zur Bedienung anzeigt: Durch kurzes Tippen auf das Flaggensymbol wechselt die Suchsprache von Französisch nach Deutsch oder umgekehrt. Tippt der Nutzer lange auf das Symbol, öffnet sich die *LanguageControlActivity* (Abb. 37)

Beginnt der Nutzer etwas in das Suchfeld einzutippen, wird das ChildFragment ausgetauscht – das *SearchStartFragment* wird durch das *SearchSuggestionsListFragment* ersetzt. Dieses kommuniziert im Hintergrund (via DataSource) mit der Datenbank und zeigt verschiedene Vorschläge zur Vervollständigung der aktuell eingetippten Zeichenfolge ein.

Wählt der Nutzer einen der Vorschläge aus oder tippt auf den Suchbutton auf der Tastatur, öffnet sich die Ansicht mit den Suchergebnissen – das *SearchSuggestionsListFragment* wird durch das *SearchResultsFragment* ausgetauscht, welches die Ergebnisse ebenfalls aus der Datenbank bezieht.

```
private void setChildFragment(int newChildFragmentId){

        Fragment fragment;
        FragmentManager fm = getChildFragmentManager();

        switch (newChildFragmentId){
            case SEARCH_START_FRAGMENT:
                // currentChildFragment durch SearchStartFragment austauschen
                            ...
            case SEARCH_SUGGESTIONS_LISTFRAGMENT:
                // Falls currentChildFragment = SearchSuggestionsListFragment
                      // currentChildFragment mit aktueller Sucheingabe updaten
                                ...
                // Falls nicht
                      // currentChildFragment durch
                         SearchSuggestionsListFragment austauschen
                                ...
            case SEARCH_RESULTS_FRAGMENT:
                // currentChildFragment durch SearchResultsFragment austauschen
                            ...
                // Tastatur schliessen
                            ...
            default:
                            ...
        }
        currentChildFragment = fragment;
}
```

Listing 38: Methode setChildFragment (Auszug aus DictionaryFragment)

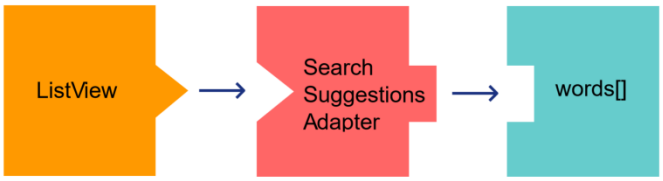
Die Klasse *DictionaryFragment* enthält eine Methode `setChildFragment(int newChildFragmentId)`. Diese wird immer dann aufgerufen, wenn sich der Suchstatus ändert. Ähnlich wie in der MainActivity werden hier Fragments über einen *FragmentManager* ausgetauscht, nur das hier ein *ChildFragmentManager* verwendet wird. Die Methode überprüft, welches Fragment gerade aktiv ist und wählt die passende Animation (Rückwärts- oder Vorwärts-Slide) für den Übergang zum neuen Fragment. Falls ein *SearchSuggestionsListFragment* angezeigt werden soll, es sich beim *currentChildFragment* jedoch schon um ein solches handelt, wird dieses nicht ersetzt, sondern lediglich die neue Sucheingabe über die Methode `setQuery(String query)` übermittelt. Mit dieser Angabe kann das *SearchSuggestionsListFragment* dann über DataSource einen neue

Datenbankabfrage durchführen und seinen Bildschirminhalt (Vorschlägeliste) aktualisieren.

SearchSuggestionsListFragment

Aufgrund einer Besonderheit möchte ich hier noch etwas genauer auf das *SearchSuggestionsListFragment* eingehen.

Das *SearchSuggestionsFragment* erbt von *ListFragment*, welches standardmässig schon eine *ListView* enthält, weswegen kein externes XML-File erstellt werden muss. Zur Anzeige von Listenelementen arbeitet die *ListView* immer mit Adaptern zusammen. Adapter dienen als Brücke zwischen dem Benutzeroberflächenelement *ListView* und der dahinterliegenden Datengrundlage. Bei einem Standard-Layout lässt sich einer der vorgefertigten List-Adapter verwenden, so wie das auch in der MainActivity für die seitliche Navigationsleiste geschieht (Kap. 4.7, Code Z. 23/24). Da im *SearchSuggestionsListFragment* neben den vorgeschlagenen Wörtern aber jeweils noch ein Flaggensymbol als Sprachangabe angezeigt werden soll, muss ein eigener Adapter programmiert werden:



In der `setQuery` Methode (welche von *DictionaryFragment* aus aufgerufen wird) wird via DataSource eine Abfrage gestartet, welche die Suchvorschläge in Form eines Word-Arrays zurückgibt. Mit dieser Liste wird ein neuer *SearchSuggestionsAdapter* erstellt. Anschliessend wird der alte Adapter der ListView durch den neuen Adapter ersetzt.

```
public void setQuery(String query){
        dataSource.getSearchSuggestions(query, new
                        DataSource.onSearchSuggestionsLoadedCallback() {
            @Override
            public void onSearchSuggestionsLoaded(Word[] words) {
                if (words != null) {
                    SearchSuggestionsAdapter adapter =
                            new SearchSuggestionsAdapter(mContext, words);
                    setListAdapter(adapter);
                }
            }
        });
}
```

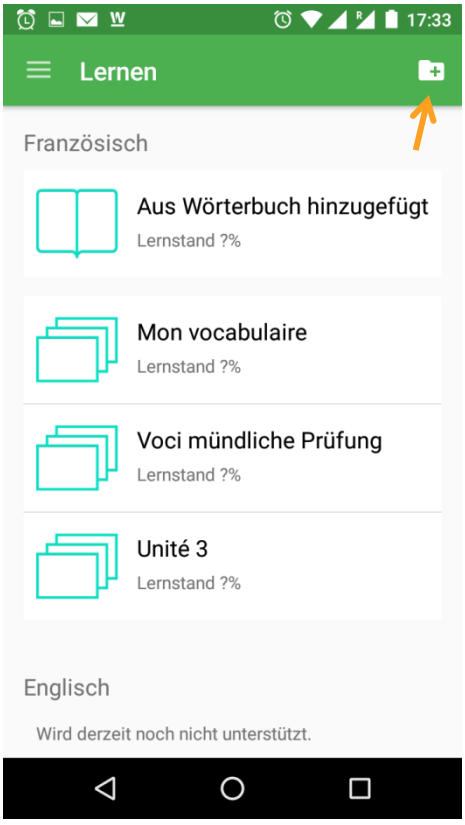
Listing 39: Methode setQuery (Auszug aus SearchSuggestionsListFragment)

```
1.  private class SearchSuggestionsAdapter extends BaseAdapter {
2.      private final LayoutInflater inflater;
3.      private final Word[] words;
4.
5.      public class ViewHolder {
6.          private ImageView ivLanguageIcon;
7.          private TextView tvWord;
8.      }
9.
10.     public SearchSuggestionsAdapter(Context context, Word[]words){
11.         inflater = LayoutInflater.from(context);
12.         this.words = words;
13.     }
14.
15.     @Override
16.     public int getCount() {
17.         return words.length;
18.     }
19.
20.     @Override
21.     public Object getItem(int position) {
22.         return words[position];
23.     }
24.
25.     @Override
26.     public long getItemId(int position) {
27.         return position;
28.     }
29.
30.     @Override
31.     public View getView(int position, View convertView, ViewGroup parent) {
32.         ViewHolder holder;
33.
34.         if(convertView == null){
35.             convertView = inflater.inflate(
36.                     R.layout.search_suggestions_list_item,
37.                     parent, false);
38.
39.             holder = new ViewHolder();
40.             holder.tvWord = (TextView) convertView.findViewById(
41.                     R.id.textview_word);
42.             holder.ivLanguageIcon = (ImageView) convertView.findViewById(
43.                     R.id.imageview_language_icon);
44.
45.             convertView.setTag(holder);
46.         }else {
47.             holder = (ViewHolder) convertView.getTag();
48.         }
49.
50.         Word wordItem = (Word) getItem(position);
51.         holder.tvWord.setText(wordItem.getString(Word.FORMAT_STANDARD));
52.         holder.ivLanguageIcon.setImageResource(
53.                 wordItem.getLanguageIconResourceId());
54.
55.         return convertView;
56.     }
57. }
```

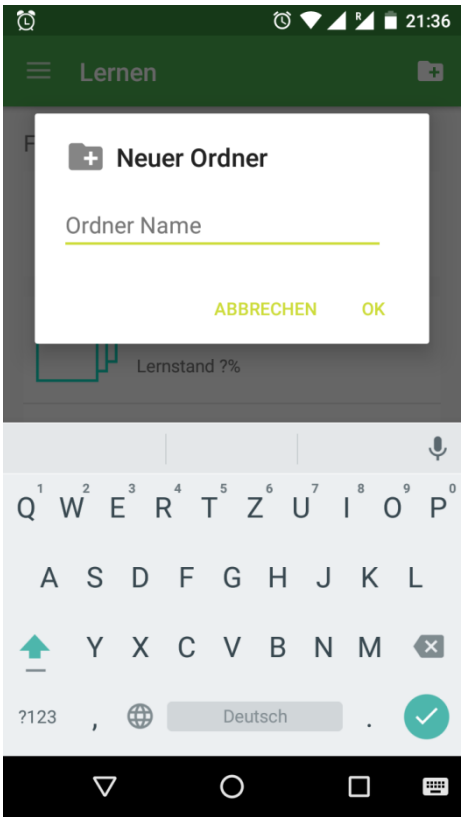
Listing 40: privater SearchSuggestionsAdapter

Der *SearchSuggestionsAdapter* überschreibt verschiedene Methoden der Superklasse *BaseAdapter*. Am wichtigsten ist die Methode `getView`, die immer dann von der *ListView* aufgerufen wird, wenn die *ListView* neue Listeneinträge zur Anzeige erzeugen muss. Dies geschieht einerseits zu Beginn, andererseits aber auch wenn der Nutzer in der Liste scrollt. Aus Performance-Gründen wird beim Scrollen jedoch nicht jedesmal ein neues Listenelement erstellt. Stattdessen wird ein bereits vorhandenes Element recycelt und mit neuem Text und neuem Bild gefüllt, deswegen der Name *convertView*. Damit dies möglich ist, wird in einem *ViewHolder* stets eine Referenz auf die *TextView* und auf die *ImageView* gehalten. Beim Recyclen eines Elements kann so sehr einfach wieder darauf zugegriffen werden.

4.10 UnitsOverviewFragment



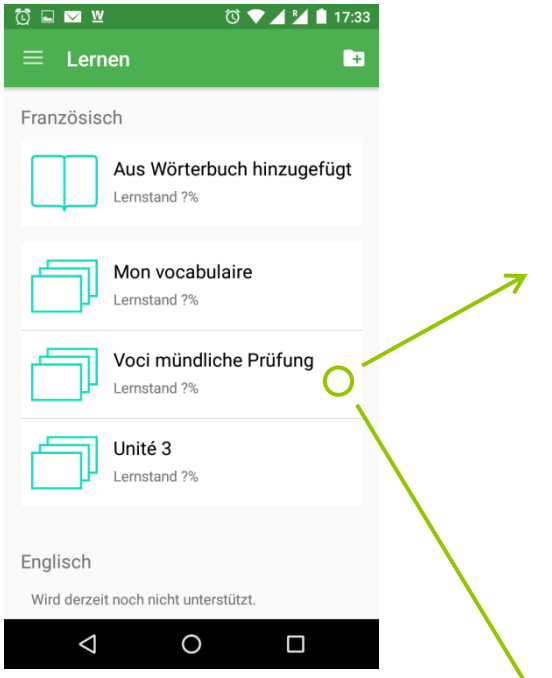
UnitsOverviewFragment



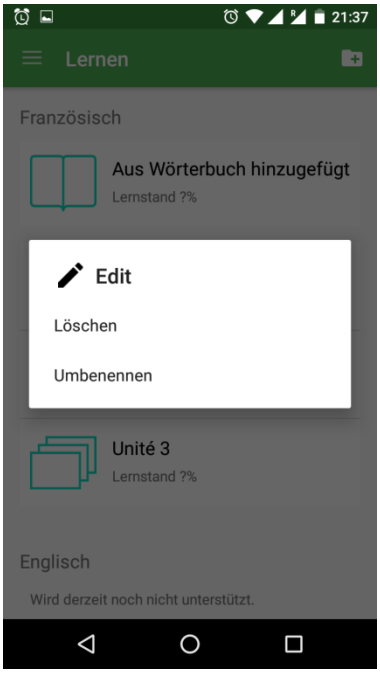
NewFolderDialogFragment

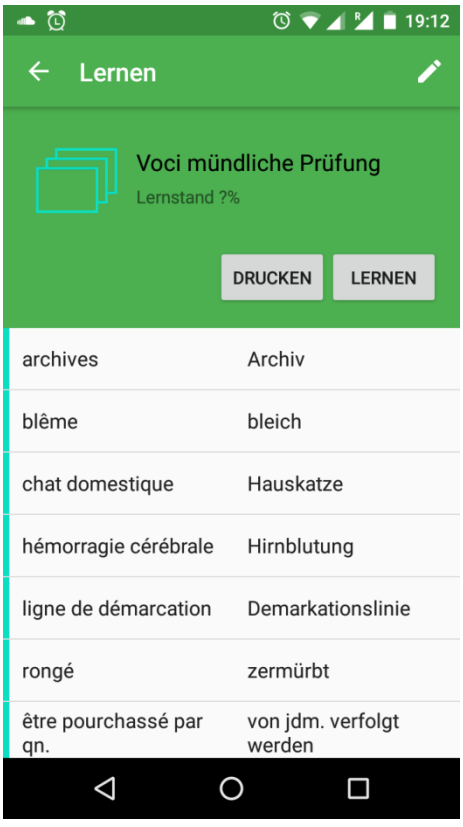
In der Lernansicht werden die verschiedenen Ordner angezeigt. Tippt man auf das *Ordner hinzufügen* Symbol (durch orangen Pfeil markiert), öffnet sich das *NewFolderDialogFragment*, wo der Nutzer den neuen Ordner benennen kann.

4.11 UnitViewerActivity



UnitsOverviewFragment



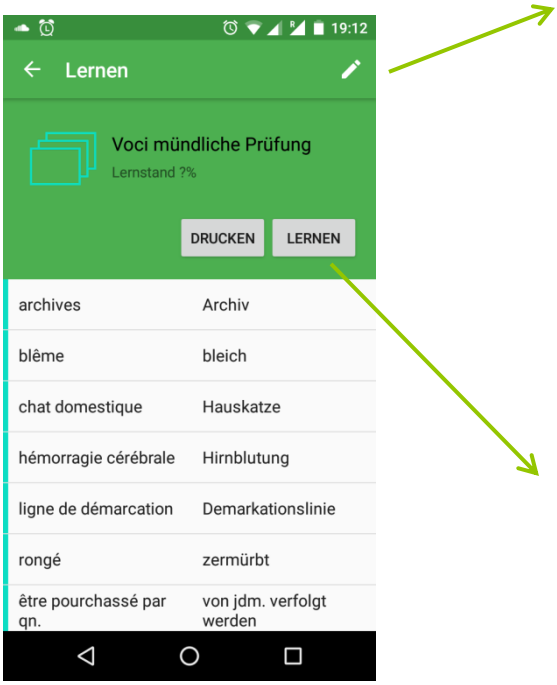


UnitViewerActivity

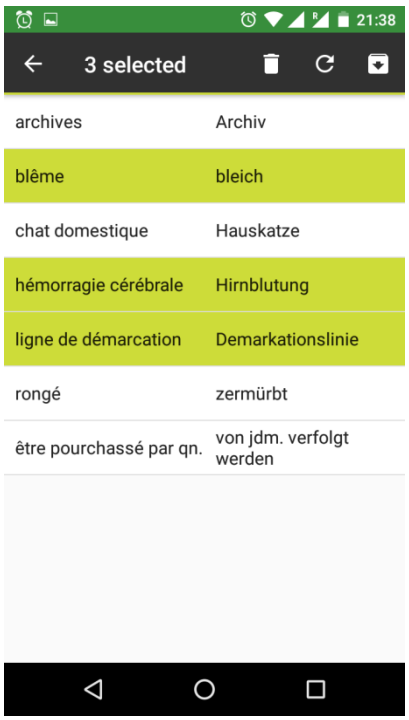
Durch langes Tippen auf einen Ordnereintrag öffnet sich ein Kontext-Menu, über welches der Nutzer den Ordner bearbeiten kann.

Durch normales kurzes Tippen gelangt der Nutzer zur *UnitViewerActivity*, welche den Ordner und alle darin enthaltenen Vokabeln anzeigt.

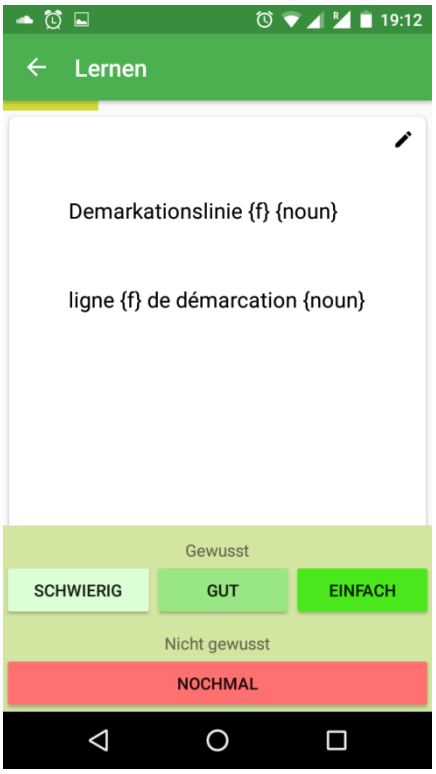
4.12 TestActivity



UnitViewerActivity



BatchEditVocablesActivity



TestActivity

Durch Tippen auf das *Bearbeiten* Symbol in der oberen rechten Ecke gelangt man zur *BatchEditVocablesActivity*, in welcher der Nutzer mehrere Vokabeln gleichzeitig Löschen, Auffrischen (Lernintervall zurücksetzen) oder in einen anderen Ordner verschieben kann.

Durch Tippen auf den *Lernen* Button gelangt der Nutzer zur *TestActivity* – dem Lernmodus der App. Durch einmaliges Tippen auf die Vokabelkarte wird die Lösung angezeigt und der Nutzer hat anschliessend die Möglichkeit, die Schwierigkeit der Antwort zu bewerten. Der hellgrüne Balken unterhalb der Kopfleiste gibt den Lernfortschritt an.

Die *TestActivity* ist nur für die Benutzeroberfläche zuständig. Für die Planung der Lernzeitpunkte von Vokabeln arbeitet es mit der *Scheduler* Klasse zusammen, welche den SM-2-Algorithmus implementiert. *TestActivity* bezieht von der Klasse *DataSource* eine Liste mit Vokabeln im ausgewählten Ordner. Diese Liste übergibt sie dann an die Klasse *Scheduler*, welche die Planung übernimmt.

4.13 Algorithmus zur Planung der Wiederholungszeitpunkte

Mein programmierter Algorithmus zur Planung der Wiederholungszeitpunkte ist eine Weiterentwicklung des von Piotr Woźniak entwickelten SuperMemo-2-Algorithmus. Der SuperMemo-Algorithmus liegt inzwischen zwar in der Version 17 vor[15], ist mathematisch jedoch so hochkomplex (adaptive Wahrscheinlichkeitskurven, Matrizen, etc.), dass ich diesen hier nicht verwende. Stattdessen habe ich mich dazu entschieden, den SM-2-Algorithmus als Grundlage zu verwenden und darauf aufbauend einen eigenen Algorithmus zu entwickeln.

SM-2-Algorithmus

Die Funktionsweise des SM-2-Algorithmus ist öffentlich im Internet ersichtlich[16]. Die Idee dieses Algorithmus besteht darin, dass jeder Vokabel ein individueller Easyness-Factor (kurz E-Factor) zugeordnet wird, der die Schwierigkeit der Karte beschreibt und bei der Berechnung des nächsten Lernintervalls miteinbezogen wird. Das Lernintervall (I) berechnet sich nach folgender Formel:

$$I(1) := 1$$
$$I(2) := 6$$
$$\text{für } n > 2 \qquad I(n) := I(n-1) * eFactor$$

Als erstes Lernintervall wird die Dauer von einem Tag gewählt, als zweites Intervall eine Dauer von 6 Tagen. Ab dem dritten Intervall wird die Berechnung mit dem E-Factor angewandt. Das neue Intervall ergibt sich aus dem letzten Intervall multipliziert mit dem E-Factor.

[15] Für weitere Informationen zum SM-17-Algorithmus: http://help.supermemo.org/wiki/SuperMemo_Algorithm#The_Algorithm
[16] https://www.supermemo.com/english/ol/sm2.htm

Nach jeder Abfrage bewertet der Nutzer die Qualität der Antwort anhand einer Skala von 0 bis 5.

q	Erklärung
0	Antwort falsch, absoluter Blackout
1	Antwort falsch, jedoch richtige Antwort erinnert
2	Antwort falsch, richtige Antwort einfach erinnert
3	Richtig, jedoch mit Schwierigkeiten beantwortet
4	Richtig, nach kurzem Zögern beantwortet
5	Richtig, Perfekte Antwort

Nur bei richtigen Antworten ($q \geq 3$) wird das nächste Intervall nach obiger Formel berechnet, bei einer falschen Antwort ($q < 3$) wird das Intervall wieder auf das Anfangsintervall $I(1)$ zurückgesetzt.

E-Factor Berechnung

Nach jeder Wiederholung wird der E-Factor angepasst. Anhand der Schwierigkeitseinschätzung wird mittels der folgenden Formel ein neuer E-Factor $eFactor'$ berechnet:

$$eFactor' := eFactor + (0.1 - (5 - q) * (0.08 + (5 - q) * 0.02))$$

q gibt dabei den Schwierigkeitsgrad in der Skala von 0 bis 5 an.

Vereinfacht bedeutet das, dass sich der E-Factor jeweils um folgenden Wert verändert:

q	$eFactor'$
0	-0.8
1	-0.54
2	-0.32
3	-0.14
4	0
5	+0.1

Es gilt allerdings die Bedingung, dass der Wert des E-Factors nie kleiner als 1.3 werden darf, um zu häufige Wiederholungen zu vermeiden.

Eigene Implementation

Auf einem Handydisplay ist nur beschränkt Platz vorhanden. Für den obigen Bewertungsansatz müssten nebst der Vokabelkarte selbst noch sechs Schaltflächen für die Einschätzung auf dem Bildschirm platziert werden, was relativ eng wäre. Die drei verschiedenen Falsch-Bewertungen sind nicht sehr aussagekräftig und eher wenig intuitiv, weswegen ich mich dazu entschieden habe, diese zwecks Benutzerfreundlichkeit zu einer Schaltfläche *Nochmal* zusammenzuführen:

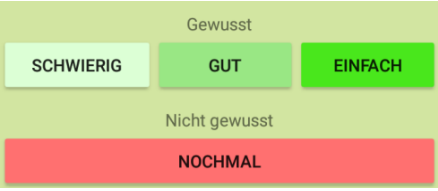


Abb. 41: Bewertungsmöglichkeiten in der App

Der SM-2-Algorithmus hat den Nachteil, dass er nicht flexibel auf unterschiedliche Lernzeiten des Nutzers reagieren kann. So fehlt eine angepasste Berechnung bei verspätetem Lernen. Damit der SM-2-Algorithmus gut funktioniert, muss sich der Nutzer nach den berechneten Lernintervallen richten.

Aus diesem Grund verwende ich für die Berechnung eine abgewandelte Formel, welche auch eine allfällige Verspätung miteinbezieht. Ich orientiere mich dabei an der Berechnungsmethode von Anki. Anki ist eine etablierte Open-Source Desktop-Anwendung zum Lernen von Karteikarten und eine der populärsten Anwendungen, die auf dem SM-2-Algorithmus aufbaut.

In Anki werden die Lernintervalle wie folgt berechnet[17]:

Einschätzung

Schwierig $I' = \left(I + \frac{delay}{4}\right) * 1.2$

Gut $I' = \left(I + \frac{delay}{2}\right) * eFactor$

Einfach $I' = (I + delay) * eFactor * easyBonus$

$easyBonus = 1.3$

$delay$ bezeichnet die Verspätung, also die Differenz zwischen dem geplanten Intervall und der effektiv verstrichenen Zeit.

[17] Quelle: Dokumentation http://ankisrs.net/docs/manual.html#frequently-asked-questions sowie Anki-Quellcode https://github.com/dae/anki/blob/master/anki/sched.py (Z. 885 – 888)

- Bei *Schwierig* wird das nächste Intervall ohne Verwendung des E-Factors berechnet. Das neue Intervall ist 1.2-mal grösser als das letzte, wobei im Falle einer Verspätung nur ein Viertel dieser miteinberechnet wird.
- Bewertet der Nutzer die Antwort mit *Gut*, wird die Hälfte der Verspätung mit angerechnet, anschliessend mit dem $eFactor$ multipliziert.
- Bei *Einfach* wird die gesamte Verspätung angerechnet und mit dem $eFactor$ multipliziert. Zusätzlich wird mit einem weiteren Faktor, dem $easyBonus$ multipliziert. Dieser beträgt im Normalfall 1.3.

Diese Berechnungsmethode hat ausserdem den Vorteil, dass die Bewertungen direkt in einem angepassten Intervall resultieren und nicht wie beim Original-SM-2-Algorithmus erst bei der nächsten Wiederholung Wirkung zeigen.

Wusste der Nutzer die Antwort nicht, wird das Intervall wie gehabt auf das Anfangsintervall zurückgesetzt. Allerdings unterscheiden sich die Start-Intervalle.

$I(1) := 10$ s

$I(2) := 10$ min

$I(3) := 1$ Tag

$I(4) := 4$ Tage

für $n > 4$ $I(n) :=$ Berechnung nach obiger Methode

In der *Scheduler* Klasse habe ich die Anki-Berechnungsmethode umgesetzt.

Unabhängig von der Berechnung der Intervalle bleibt jedoch ein Problem bestehen: Der Nutzer hat nicht immer Zeit, alle für einen Tag geplanten Vokabeln durchzuarbeiten. Daher sollte bei der Planung sichergestellt werden, dass die wichtigsten Vokabeln zuerst gelernt werden. Eine Vokabel, welche aufgrund eines viertägigen Intervalls auf den heutigen Tag geplant wurde, ist wichtiger, als eine Vokabel mit einem Intervall von einem Monat. Entscheidend für die Wichtigkeit der Vokabel ist die prozentuale Abweichung des effektiven Intervalls vom berechneten Intervall. Aus diesem Grund habe ich zur Planung der Reihenfolge einen sogenannten $dueFactor$ eingeführt:

$$dueFactor = \frac{scheduledInterval}{actualInterval}$$

Dieser drückt ein Verhältnis des geplanten Intervalls $scheduledInterval$ zur tatsächlich verstrichenen Zeit $actualInterval$ aus.

Ergibt der $dueFactor$ einen Wert kleiner als 1, bedeutet das, dass die Karte *due*, also fällig ist. Umso näher dieser Wert bei 0 liegt, desto grösser ist die prozentuale Abweichung und desto wichtiger ist die Vokabel.
Vokabeln mit einem $dueFactor$ grösser 1 sind nicht fällig.

In Java programmiert sieht das Ganze wie folgt aus:
Die *Vocable* Klasse enthält eine Methode `getDueFactor`, die mithilfe der Objektdaten den $dueFactor$ berechnet:

```
public float getDueFactor(Date now){
   double scheduledInterval = getScheduledInterval();
   double actualInterval = getActualInterval(now);

   if (scheduledInterval != 0 && actualInterval != 0){
       return (float) (scheduledInterval / actualInterval);
   }else {
       Log.e(TAG,"getDueFactor: infinite dueFactor should not happen. " +
               "Bad programming...");
       return Scheduler.DUE_FACTOR_INFINITE;
   }
}
```

Listing 42: Auszug aus *Vocable* Klasse

Um alle an einem Tag fälligen Vokabeln zu erhalten, setzt die *Scheduler* Klasse für *now* die Uhrzeit 3 Uhr morgens des nächsten Tages ein. Alle Vokabeln, die dann bei Aufruf von `getDueFactor(Date now)` einen Wert kleiner als 1 aufweisen, betrachtet *Scheduler* als fällig.

Anschliessend setzt *Scheduler* die aktuelle Zeit ein und ordnet die Vokabeln in aufsteigender Reihenfolge.

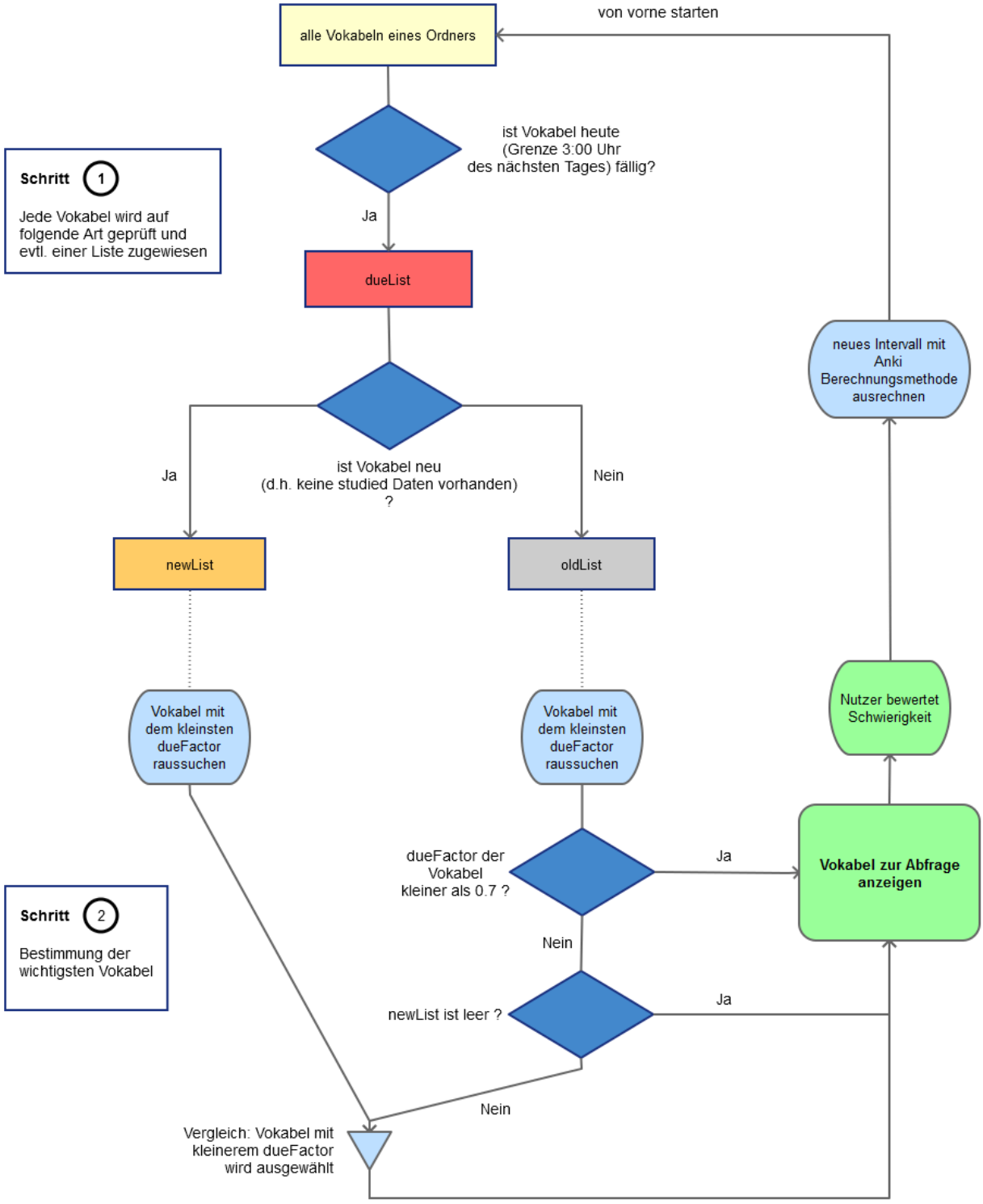
Es bleibt jedoch die Frage, wie *Scheduler* mit neu hinzugefügten Vokabeln umgeht. Da eine Vokabel während des Hinzufügens aus dem Wörterbuch schon ein erstes Mal ins Ultrakurzeitgedächtnisgedächtnis gelangt, habe ich mich dazu entschieden, für jede Vokabel ab diesem Zeitpunkt ein erstes Intervall ($I(1) = 10$ Sekunden) zu planen.

Wird eine grössere Anzahl neue Vokabeln hinzugefügt, führt dies jedoch oft zum Problem, dass die neuen Vokabeln einen sehr tiefen $dueFactor$ erhalten (ein Vielfaches von 10 Sekunden ist schnell erreicht) und mit diesem andere, bereits länger im Lernmodus befindliche Vokabeln konkurrieren. So würden zuerst nur die neuen Wörter gelernt und ältere vernachlässigt.

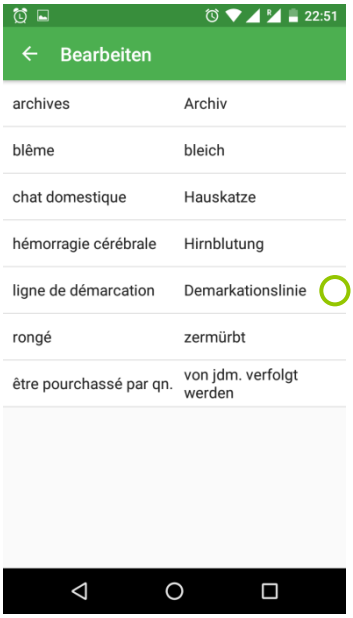
Um dies zu verhindern, habe ich die Bedingung eingeführt, dass erst dann neue Wörter abgefragt werden, wenn alle alten Wörter einen $dueFactor$ grösser als 0.7 aufweisen. So wird die in Kapitel 3.1 beschriebene Idee „Schnelllernmethode“ umgesetzt und neue Wörter portionsweise eingeführt, wodurch das Gehirn diese besser aufnehmen kann. Eine Vokabel, die als *Nicht-Gewusst* eingestuft wurde und daher für 10 Sekunden später geplant wurde, wird so auch spätestens nach 14 Sekunden wieder angezeigt, ohne von neuen Vokabeln mit tieferem $dueFactor$ verdrängt zu werden.

Der Grenzwert 0.7 stellt dabei einen Kompromiss zwischen schnellem Aneignen neuer Wörter und dem Nicht-Vernachlässigen alter Wörter dar. Der Wert 0.7 entspricht einer positiven Abweichung von 43% gegenüber dem geplanten Intervall. Alle verwendeten Faktoren sind allerdings zentral in Konstanten gespeichert. Dem Nutzer soll es in Zukunft ermöglicht werden, über die Einstellungen daran Anpassungen vornehmen zu können.

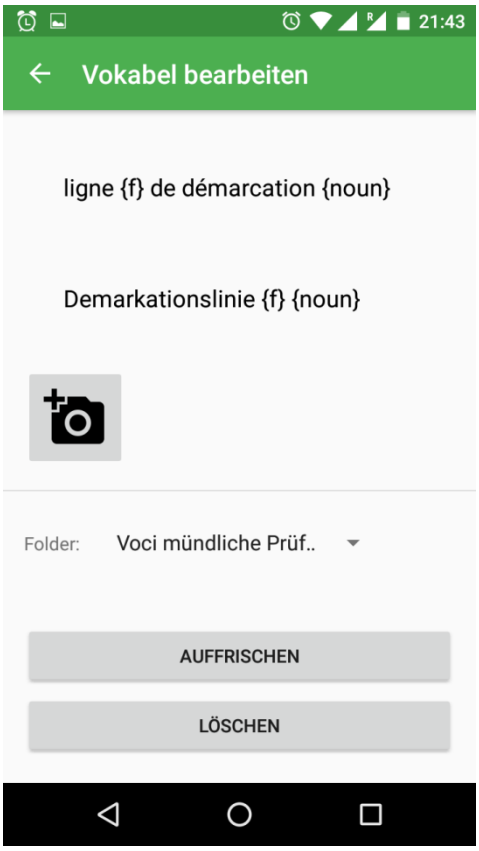
Als Zusammenfassung ist der Algorithmus zur Planung der Reihenfolge nachfolgend als Flussdiagramm dargestellt:



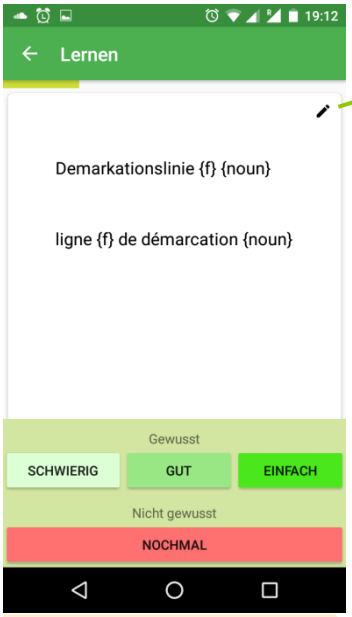
4.14 EditVocableActivity



BatchEditVocables-Activity



EditVocableActivity



TestActivity

In der *BatchEditVocablesActivity* kann man mehrere Vokabeln miteinander bearbeiten (durch langes Antippen). Bei kurzem Antippen einer Vokabel öffnet sich jedoch die *EditVocableActivity*, die es ermöglicht eine Vokabel einzeln in der Detailansicht zu bearbeiten.

Alternativ gelangt man auch direkt aus der *TestActivity* über das kleine Bearbeitungssymbol in der rechten oberen Ecke zur *EditVocableActivity*.

In der *EditVocableActivity* soll es künftig auch möglich sein, ein Bild zu einer Vokabel hinzuzufügen. Die Chancen stehen gut, dass ich die *EditVocableActivity* nochmals überarbeiten werde, da ich für diese nicht mehr viel Zeit zum Programmieren hatte.

4.15 Ausblick

Die Entwicklung der App wird wohl auch in Zukunft noch länger nicht abgeschlossen sein. Mir kommen ständig weitere Ideen in Sinn, die ich gerne noch umsetzen möchte.

Das Ziel ist es jedoch, bald eine Version in den Google Play Store einzustellen und damit einem grösseren Nutzerkreis öffentlich zugänglich zu machen. Bis zu diesem Zeitpunkt kann eine Testversion heruntergeladen werden (mehr dazu unter Kap. 8.1 Quellcode und App Download).

Als nächste Schritte bei der Entwicklung ist vorgesehen, die weiteren geplanten Module umzusetzen. Eine Druckfunktion liesse sich sehr gut über eine WebView realisieren. So könnte das Dokument in einfachem HTML gestaltet werden, wobei die restliche Arbeit (Erstellen eines druckbaren PDF-Files) dann durch die WebView erledigt wird.

Für eine kamerabasierte Texterkennungsfunktion möchte ich vorerst mit der Open-Source App TextFee zusammenarbeiten. In TextFee wählt man nach dem fertigen Importieren dann einfach unter *Text teilen* Vocado aus.

Nebst diesen Funktionen sollen die Suchfunktion sowie die interne Sprachverarbeitung verbessert werden und auch der Algorithmus zur Planung der Lernzeitpunkte weiterentwickelt werden.

Nach und nach möchte ich weitere der im Kapitel *Appkonzept* vorgestellten Ideen umsetzen.

5 Schlusswort

Rückblickend kann ich sagen, dass mir das Projekt insgesamt viel Spass gemacht hat. Ich hatte schon länger darüber nachgedacht, eine App zum Sprachenlernen zu programmieren, allerdings fehlte mir in der Freizeit die Zeit dazu. Durch die Maturaarbeit konnte ich dieses Projekt realisieren.

Der Lernforschungsteil und die Befragungen gaben mir viele Anregungen für die Funktionen einer Lernapp. Das gewonnene Wissen hat mir persönlich zudem beim Lernen für Prüfungen sehr weitergeholfen. Ich würde mir wünschen, dass Lernmethodik auch mehr im Unterricht thematisiert wird.

Im Programmierteil konnte ich die wichtigsten Ideen in einer App umsetzen. Ich hatte schon öfters programmiert, jedoch nie für Smartphones. Dass ich das Projekt so verwirklichen konnte, war nur möglich, weil ich bereits vor zwei Jahren begonnen hatte, mich in die Java Programmierung einzuarbeiten[18]. Durch das Lesen der beiden Bücher *Head First Android Development* und *Android 5, Apps entwickeln mit Android Studio* sowie diverser fachspezifischer Internetforen habe ich mich zusätzlich in die Android Programmierung eingearbeitet. Es ist mir gelungen, eine praktische App zu entwickeln, die Nutzern beim Lernen hilft, worauf ich stolz bin. Das Programmieren hat mir sehr Spass gemacht und ich habe viel dazu gelernt. Trotz Programmiererfahrung habe ich jedoch den Aufwand unterschätzt, den das Programmieren einer Smartphone-App mit sich bringt.

Bis zur Marktreife der App sind noch ein paar Monate mehr Zeit erforderlich. Jedoch ist sie bereits jetzt praktisch nutzbar – ich lerne inzwischen regelmässig damit.

Rückblickend auf das Projekt bin ich mit der Vorgehensweise sehr zufrieden. Für ein nächstes Mal würde ich aber wahrscheinlich ein weniger komplexes Thema auswählen, da das Projekt mit vertiefter Literaturrecherche und Programmierung doch sehr zeitintensiv war.

Ich möchte die App gerne weiterentwickeln. Mir kommen immer wieder neue Ideen in Sinn, die ich gerne verwirklichen würde. Ich bin überzeugt davon, dass die Effizienz mit digitalem Lernen verbessert werden kann. Allerdings hilft auch eine Lernapp wenig, wenn der schulische Druck noch weiter erhöht wird.

[18] Mittels des Buches *Programmieren lernen mit Java* und dem Internettutorial gailer-net.de

6 Dank

Ich möchte mich ganz herzlich bei all denen bedanken, die mich während der Entstehung dieser Arbeit unterstützt haben. Zuallererst danke ich Herrn Dr. Steiger, der meine Arbeit betreut, mir wichtige Hinweise gegeben, mich in meiner Vorgehensweise bestärkt und auch einen gewissen Freiraum gelassen hat.

Ein grosser Dank geht auch an meine Eltern, die meine Entwürfe regelmässig gegengelesen und korrigiert haben und mich motiviert haben, wenn mir das Schreiben schwierig fiel. Ich bin für ihre Unterstützung sehr dankbar.

Ich danke meiner Schwester Kathrin für die Gestaltung des App-Logos und ihre ehrliche Meinung.

Ausserdem möchte ich mich bei Corinne, Yannis, Kathrin, Jelena und Frederic für die Interviews zu ihren persönlichen Lernmethoden bedanken.

7 Bibliographie

7.1 Literaturverzeichnis

Lernforschung

Buzan, Tony: *Kopftraining, Eine Anleitung zum kreativen Denken*. Völlig überarbeitete, aktualisierte und erweiterte Auflage. München 1993

Ebbinghaus, Hermann: *Über das Gedächtnis, Untersuchungen zur experimentellen Psychologie*. Leipzig 1885

Frick, René / Mosimann, Werner: *Lernen ist lernbar, Eine Anleitung zur Arbeits- und Lerntechnik*. 1. Auflage. Aarau 1994

Leitner, Sebastian: *So lernt man lernen, Der Weg zum Erfolg*. 13. Auflage. Breisgau 1972.

Vester, Frederic: *Denken, Lernen, Vergessen*. Überarbeitete, erweiterte Auflage 2001. München 1975

Weitere Literatur

Krengel, Martin: *Bestnote, Lernerfolg verdoppeln, Prüfungsangst halbieren*. Berlin 2012

Schuster, Martin: *Optimal vorbereitet in die Prüfung, Erfolgreiches Lernen, richtiges Prüfungsverhalten, Angstbewältigung*. 2. aktualisierte und erweiterte Auflage. Göttingen 2014

Programmierung

Griffiths, Dawn & David: *Head First Android Development.* First Edition. Sebastopol 2015

Habelitz, Hans-Peter: *Programmieren lernen mit Java.* 1. Auflage. Bonn 2012

Künneth, Thomas: *Android 5, Apps entwickeln mit Android Studio*. 3. aktualisierte und erweiterte Auflage. Bonn 2015

7.2 Hinweis zu Internetquellen

Alle angegebenen Internetquellen und Links wurden zuletzt im November 2016 aufgerufen. Es ist möglich, dass sich Inhalte der Websites mit der Zeit verändern.

7.3 Abbildungen

Bei allen externen Grafiken und Bildern ist die Quelle unter der Abbildung angegeben. Abbildungen, bei denen keine Quelle angegeben ist, sind eigene Grafiken oder Screenshots.

8 Anhang

8.1 Quellcode und App Download

Der Quellcode ist zu umfangreich um ihn hier auflisten zu können (mehrere tausend Zeilen Programmiercode). Stattdessen kann er online unter folgendem Link eingesehen werden:

https://goo.gl/yaGJJ4

Ausserdem lässt sich die App dort auch als .apk-Datei herunterladen, die direkt auf einem Android Handy (ab Software-Version 5.0) installiert werden kann.

Interessierte können mir auch jederzeit gerne eine Mail an folgende Adresse schreiben:

jonas.wolter@sunrise.ch