

Europaschule
Maria-Wächtler-Gymnasium
Städt. Gymnasium für Mädchen und Jungen
mit deutsch-englischem Zweisprachenzug

QUADRA SOLVE

Im App Store weltweit verfügbare iOS-App zur Berechnung reeller und komplexer Lösungen quadratischer Gleichungen mit speziellem Fokus auf Barrierefreiheit



Projektarbeit von **Marvin von Hagen** (Q1)

M.vonHagen@iCloud.com / +49 151 22612364

Betreuende Lehrer: **Fr. Mielchen-Woköck & Hr. Lomann**

Projektkurs 2016 / 17

Kommentar zur Projektarbeit von Marvin von Hagen

Marvin hat mit der Programmierung einer App zur Lösung quadratischer Gleichungen unter dem Aspekt der Barrierefreiheit und dem Ziel der Veröffentlichung im App Store zusammen mit seinem Projektpartner Vincent van Oepen eine anspruchsvollen Aufgabe als Projektkursthema gewählt. Dazu musste sich Marvin zunächst in die geeignete Programmiersprache einarbeiten.

In der vorgelegten schriftlichen Projektarbeit hat Marvin seinen Arbeitsanteil bei der Entwicklung der App und auch die Schritte zur Veröffentlichung im App sorgfältig dokumentiert.

In den Beratungen diskutierte Aspekte, z.B. zur Optimierung der Barrierefreiheit, wurden in der vorgelegten Version überzeugend umgesetzt.

Note der schriftlichen Arbeit: sehr gut (14 Pkt.)

Marvin hat die Beratungsangebote regelmäßig genutzt und Rückmeldungen zu den aktuellen Ergebnissen seiner Arbeit zuverlässig durchgeführt und diese auch schriftlich vorgelegt.

Note für die Sonstige Mitarbeit: sehr gut (14 Pkt.)

Gesamtnote: sehr gut (14 Pkt.)

Essen, den 22.06.2017

U. Tielchen-Wöcke

John C. Stennis

Inhaltsverzeichnis

- 3 Einleitung
- 5 Xcode & Swift
- 8 pq-Formel-App
- 10 Design & UI
- 12 Kompatibilität
- 15 Komplexe Zahlen
- 16 Der QuadraSolve-Algorithmus
- 19 Der QuadraSolve-Algorithmus - Erläuterung
- 22 Weltweite Distribution im App Store
- 27 Vermarktung
- 29 Update 1.1
- 30 QuadraSolve 1.2 *Darstellung des Endprodukts*
- 33 Eigener Arbeitsteil
- 33 Nachwort
- 34 Danksagung
- 35 Glossar
- 38 Anhang
- 39 Versicherung über die selbstständige Anfertigung
- 40 Literaturverzeichnis

Einleitung

Seit über fünf Jahren festigt sich mein Wunsch, beruflich in der Informatik tätig zu werden. Als ich zu meinem zwölften Geburtstag einen iPod Touch geschenkt bekam, erschloß sich mir eine völlig neue Welt - ein portables Gerät, mit dem ich überall auf der Welt mit allen Personen kommunizieren kann, das alles weiß und nie langweilig wird. Die mobile Technik wirkt auch heute noch wie ein Wunder.

Ich habe immer alles hinterfragt und zu verstehen versucht und Erwachsene konnten mir das meiste erklären. Bei scheinbar magischen Glasscheiben, die eine schier unendliche Funktionsvielfalt bieten, stoßen sie jedoch meist an ihre Grenzen. Wie kann ein Prozessor schneller rechnen als jeder Mensch? Wie kann durch die von der Sonne ausgestrahlten Energie meine Musik erklingen? Und vor allem: Wie kann ich in meinem Leben an solchen Entwicklungen mitwirken?

Diese Faszination bestimmt bis heute mein Leben. Solchen Fragen nach dem „Warum?“ wird meist nur in der theoretischen Physik nachgegangen. Heutzutage baut die Informatik auf den physikalischen Gesetzen auf und nutzt diese logisch: Während die Programmierung der ersten Computer noch mehr physikalisch-technisches Verständnis verlangte, ist dies im 21. Jahrhundert dank Compilern und Entwicklungsumgebungen nicht mehr vonnöten. Das bedeutet, dass sich in modernen Sprachen wie Swift programmieren lässt, ohne sich dafür mit den grundlegenden Eigenschaften eines Computers befassen zu müssen.

Während ich auch ein Studium der theoretischen Physik aufgrund der Chance auf Forschung an aktuell unlösablen Problemen in Betracht ziehe, sehe ich in der Informatik und speziell der Softwareentwicklung die vielversprechenderen Berufsperspektiven. Zudem kann naturwissenschaftliche Forschung im Gegensatz zur Programmierung trotz langjähriger und intensiver Arbeit ergebnislos bleiben und folglich zur Frustration der daran beteiligten ForscherInnen führen.

Da das Interesse schon länger bestand, entschied ich mich für Informatik als zweites Wahlpflichtfach und beschäftige mich im Zuge dessen seit der achten Klasse mit Programmieren. Zwar haben wir mit Java auch eine Programmiersprache kennengelernt, mit der sich ernstzunehmende Programme entwickeln ließen, allerdings blieb es dabei meist bei Banalitäten wie einen Käfer auf einem schachbrettartigem Spielfeld Pilze essen zu lassen und ein Dreieck Strichmännchen zeichnen zu lassen.

Um die Arbeit professioneller Entwickler kennenzulernen, führte ich mein Praktikum am Ende der Einführungsphase in dem Informatikunternehmen „Brockhaus AG“ durch. Dort habe ich innerhalb von zwei Wochen zwei Raspberry Pis (Einplatinencomputer) so konfiguriert und programmiert, dass sie bei Anschluss ans Stromnetz eigenständig hochfahren, einen Browser starten, die netzwerkinterne Verbindung zum anderen Pi aufbauen und dann im Vollbildmodus eine proprietäre Videokonferenz starten konnten. Dieses System wird auch heute noch zur Kommunikation zwischen den Standorten Lünen und Dortmund verwendet und war eigentlich als Projekt für die nächsten sechs Monate geplant. Als „Dankeschön“ schenkte mir die Brockhaus AG am Ende meines Praktikums das Buch „iOS-Apps programmieren mit Swift“¹.

Der Projektkurses hat mich aufgrund der vielfältigen Themenmöglichkeiten im MINT-Bereich direkt überzeugt. Hierbei war es vor allem von Vorteil, dass wir zunächst etwas Zeit für eine fundierte Themenwahl eingeräumt bekamen. Mit Vincent fand ich einen ebenso naturwissenschaftlich-technisch interessierten Partner, der zudem seit vielen Jahren ein guter Freund ist. In den Sommerferien entschieden wir uns, unseren Projektkurs für die Entwicklung einer iOS-App zu nutzen.

Mit der Entwicklung eigenständiger Anwendungen - nicht der Programmierung innerhalb anderer Programme - hatte ich zu Beginn dieses Projektkurses allerdings noch keinerlei Erfahrung. Ziel dieses Projektkurses ist die Auseinandersetzung mit der Appentwicklung für iOS, wovon ich mir aufschlussreiche Erkenntnisse für meine berufliche Zukunft erhoffe. Mit dem am Ende meines Praktikums erhaltenen Buchs hatten wir direkt entsprechende Fachlektüre zum Starten unseres Projektkurses.

Xcode & Swift

Zur Vorbereitung auf den Projektkurs lasen wir das Buch „iOS Apps programmieren mit Swift“. Zunächst machten wir uns mit unserer Integrierten Entwicklungsumgebung (IDE), Xcode, vertraut. Im Folgenden werden die wichtigsten Bereiche Xcodes vorgestellt:

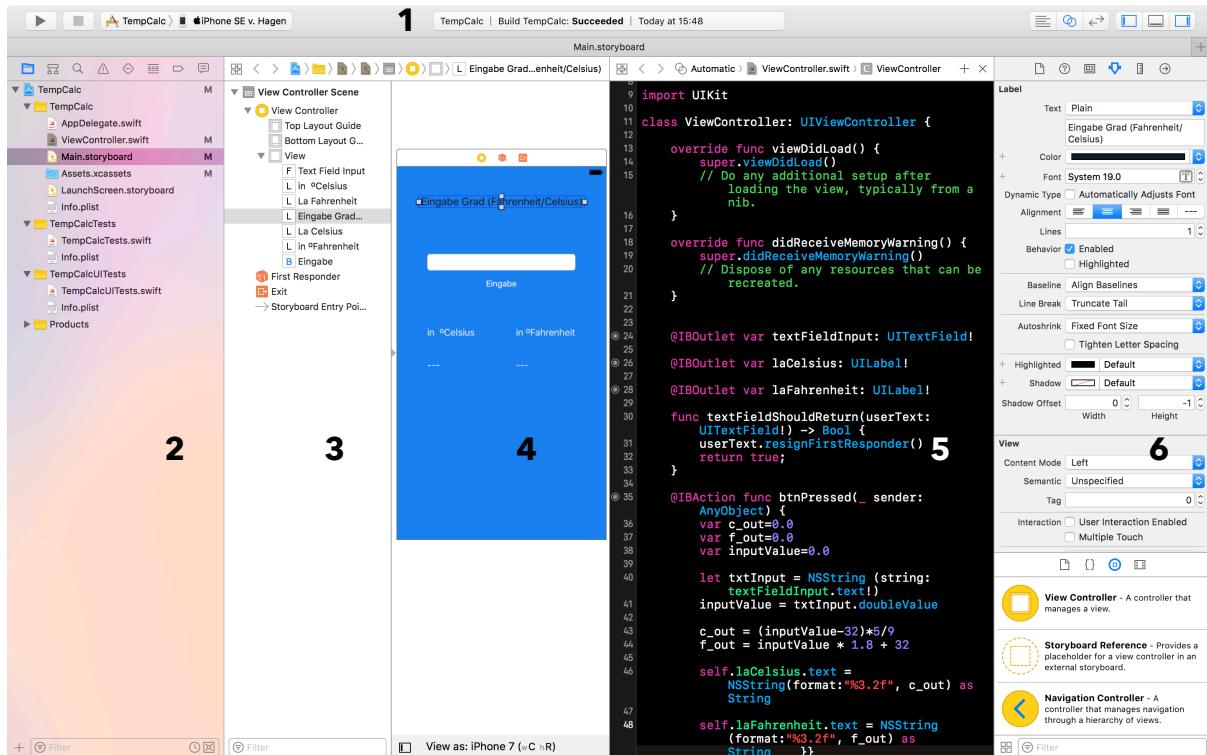


Abb. 1: Xcodes Benutzeroberfläche / Temperaturrechnerapp

1. Toolbar

In der Toolbar wird im mittigen Textfeld der Name des gerade geöffneten Projekts und der Status, in dem es sich befindet, angezeigt. Außerdem wird dort durch ein rotes Ausrufezeichen signalisiert, wenn sich Fehler im Quellcode befinden. Des Weiteren gibt es ein gelbes Ausrufezeichen, das auf diverse Dinge hinweist, die jedoch nicht immer Fehler sind. Beispielsweise wird mir bei QuadraSolve (s. S. 12 ff.) gemeldet, dass das oben stehende $ax^2+px+q=0$ auf dem iPhone anders aussieht, als es aktuell im Main.storyboard dargestellt wird. Dies ist von mir jedoch durchaus beabsichtigt und kann demnach ignoriert werden. Am linken Ende der Toolbar lässt sich die App auf dem jeweils ausgewählten Gerät ausführen, simulieren und stoppen. Am rechten Ende der Toolbar kann man zwischen verschiedenen Ansichten in Xcode wechseln, z.B. lassen sich so Main.storyboard und ViewController.swift parallel anzeigen.

2. Project Navigator

Hier lässt sich das ganze Projekt auf Dateiebene einsehen. Codedateien zum Beispiel enden auf „.swift“, das UI auf „.storyboard“. Je nachdem, welche Datei man ausgewählt hat, ändert sich auch die Ansicht in Xcode.

3. Project und Target-Übersicht

Hier findet sich u.a. eine Auflistung aller im Storyboard befindlichen Objekte und ihrer zugehörigen Constraints.

4. Interface Builder

Im Interface Builder lässt sich das UI anpassen und verändern. Man kann zum Beispiel neue Objekte aus der Object Library einbauen. In der Object Library befinden sich Objekte wie Textfelder, Views und Codeschnipsel.

5. Codefenster

Wählt man im Project Navigator „ViewController.swift“ aus, gelangt man in das Codefenster. Dort spielt sich die wesentliche Programmierung ab, indem beispielsweise Objekte aus dem Interface Builder mit den Outlets im Quellcode verknüpft werden können, um sie im Quellcode gezielt ansprechen zu können.

6. Inspektoren

Mit den insgesamt sechs Inspektoren lassen sich die Eigenschaften des aktuell im Interface Builder ausgewählten Objekts einsehen und verändern. Dazu zählen beispielsweise die typographische Gestaltung eines Labels, die von Vincent zur Lokalisation benötigten Metadaten und die Verwaltung der von mir konfigurierten Constraints.

Im Anschluss an die Einführung in die Benutzeroberfläche Xcodes, findet sich ein Tutorial, nach dessen Anleitung wir die in Abb. 1 dargestellte Temperaturrechnerapp entwickelten. Man gab eine Zahl an und bekam das Ergebnis sowohl in °Celsius als auch in °Fahrenheit.

Dabei verstanden wir aber nur die einzelnen im Tutorial vorgegebenen Schritte allerdings nur sehr bedingt. Nachdem wir nun also mit unserer Programmierumgebung umzugehen wussten, mussten wir uns eingestehen, immer noch nicht über ausreichend tiefgründige Kenntnisse bezüglich der Programmiersprache

Swift zu verfügen, um in der Lage zu sein, eine eigene App von Grund auf zu programmieren.

Deshalb entschieden wir uns, uns näher mit den bisher nur flüchtig überflogenen Seiten des Buchs zu beschäftigen. Dazu nutzten wir die Projekttage im September und lasen hunderte Seiten, die im Besonderen Variablen, Konstanten, Zuweisungen, Datentypen, Enumerationen, Arrays, Dictionaries, Operatoren, Schleifen und Funktionen thematisieren.

pq-Formel-App

Ziel dieses Projektkurses war entgegen des nach einem Tutorial entwickelten Temperaturrechner eine komplett eigenständig entwickelte App, die im besten Fall sogar veröffentlicht werden könnte. Um eine App zu entwickeln, wird jedoch zunächst eine Idee bzw. ein Plan benötigt. Wie dies zustande kam, werde ich im Folgenden erläutern.

Da ich meinem in Baden-Württemberg lebenden, blinden Cousin in letzter Zeit viel bei Hausaufgaben und Klausurvorbereitung im Fach Mathematik helfen musste, wusste ich, dass Menschen, die Gleichungen nicht visuell sehen oder vorstellen können, v.a. mit Brüchen extreme Probleme haben.

Sein iPhone hilft ihm bereits im Alltag, weshalb er beispielsweise als einer der 2000 Beta-Tester die neue App der Deutschen Bahn „Barrierefrei“ testet, da ihm ohne sie die Nutzung des öffentlichen Personennahverkehrs unmöglich wäre.

Aus der festgestellten Problematik resultierend, habe ich mich dann gefragt, weshalb es keine einfache App gibt, die es ihm ermöglicht, quadratische Gleichungen ohne die Mitternachtsformel und anschließende für Blinde extrem aufwendige Äquivalenzumformungen zu lösen.

Mit allen neu gelernten Fähigkeiten fingen wir also an, eine zunächst nicht barrierefreie App zur Berechnung der pq-Formel zu entwickeln. Kurz nach den Herbstferien sah unsere PQ-Formel-App dann wie in Abb. 2 dargestellt aus. Die aufgetretenen und behobenen Probleme in ihrer Gesamtheit darzustellen, würde

•••◦ Drillisch ⌂ 21:27 1 0 ✖ 57% 🔋

$$x^2 - px + q = 0$$

3

2

-1.0 or -2.0

[Calculate](#)

1	2 ABC	3 DEF
4 GHI	5 JKL	6 MNO
7 PQRS	8 TUV	9 WXYZ
,	0	✖

Abb. 2: pq-Formel-App

über den Rahmen dieser Projektarbeit hinausgehen, weshalb die folgende Übersicht sich exemplarisch auf bedeutsame Probleme beschränkt.

- Die berechneten Ergebnisse waren nicht immer richtig.
- Alle UI-Elemente außer der Tastatur befinden sich bei anderen Displaygrößen an anderen, falschen Stellen.
- VoiceOver funktioniert nicht, beispielsweise lässt sich die (falsche) Gleichung nicht vorlesen.
- In den Textfeldern lassen sich höchstens zweistellige Zahlen darstellen.
- Bei Verwendung von Kommata stürzt die App sofort ab.
- Es lassen sich keine negativen Werte eingeben.
- Das Layout ist nicht ansprechend, zudem sind alle Schaltflächen zu klein.
- Absturz bei fehlenden Werten für p oder q.

Design & UI

Um die aufgeführten Probleme zu lösen und die App zu verbessern, habe ich die App Anfang November zunächst dunkel gestaltet und die falsche LaTeX-Gleichung entfernt. Statt den Parametern p und q Werte zuzuweisen, konnte der Nutzer in dieser App-Version direkt in die Lücken in der Gleichung Zahlen einsetzen. Obwohl ich mir das in der Theorie vorher optisch und funktionell ansprechender vorgestellt hatte, als den „Umweg über Parameter“ zu gehen, erwies sich dieses UI als unpraktisch: aufgrund der geringen Breite eines iPhones ließen sich im sichtbaren Bereich nur extrem kleine Zahlen eingeben und die meisten Nutzer wussten ohne Anleitung nicht, wo sie die Werte einzugeben hatten, da die Textfelder nicht entsprechend hervorstachen.

Zudem erwies sich mein Code als fehlerhaft, da bei vielen Werten die Ergebnisse nicht stimmten: in Abb. 3 stimmen zwar die Beträge, aber das Vorzeichen ist falsch. Der Wert -4 konnte auch nur per Copy & Paste eingesetzt werden, da es in der App keine native Möglichkeit gab, negative Werte für p und q einzusetzen.

Um große Werte für die Parameter eingeben zu können, gestaltete ich das Layout erneut um und setzte die entsprechenden Textfelder übereinander. Wenn quadratische Gleichungen in der Form $x^2 + k_2x + k_1 = 0$ vorliegen, ist die bisher verwendete Schreibweise $x^2 + px + q = 0$ üblich, bei quadratischen Gleichungen mit zusätzlichem Parameter ($k_3x^2 + k_2x + k_1 = 0$) wird hingegen $ax^2 + bx + c = 0$ häufiger verwendet. Demnach entsprechen die von unserer App lösbarer

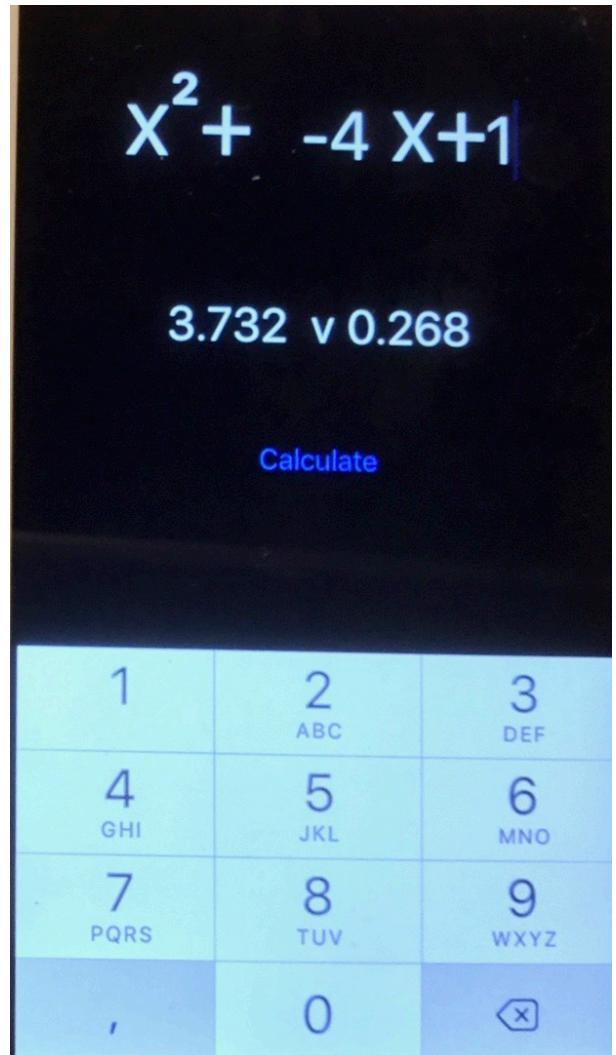


Abb. 3: pq-Formel-App

Gleichungen letzterer Form mit den Parametern a , b und c . Mathematisch und syntaktisch spielt die Benennung der Parameter keine Rolle und verwirrt im Zweifelsfall nur mathematisch unerfahrene Nutzer, die sich eher eine „App zur Lösung der pq-Formel“ als eine App zur Lösung reeller und komplexer Nullstellen quadratischer Gleichungen wünschen. Um es diesen Nutzern einfacher zu machen, entschied ich mich für die Nutzung der weiter verbreiteten Parameter p und q für k_2 und k_1 und a für k_3 . Wie in Vincents Projektarbeit nachvollzogen werden kann, integrierte er dazu ein drittes Textfeld in den Code und teilte die Werte von p und q durch den im neuen Textfeld angegebenen Wert.

Nachdem die LaTeX-Gleichung der ersten Version durch eine Gleichung ohne Variablen ersetzt wurde, nutzte ich fortan ein UILabel mit dem entsprechenden Text „ $ax^2+px+q=0$ “, das mehrere Vorteile mit sich bringt: Es lässt sich leichter VoiceOver-kompatibel gestalten, sorgt dank gleicher Schriftart für ein einheitlicheres und ansprechendes Aussehen und bietet eine intuitivere Verwendung als der einem Lückentext ähnelnde Vorgänger.

Um die visuelle Verbindung von Textfeldwerten und der obigen Gleichung zu schaffen, färbte ich die drei Variablen gemäß der iOS Human Interface Guidelines² ein: a Rot (R255/G59/B48), p Gelb (R255/G204/B0) und q Blau (R0/G122/B255). Zusätzlich färbte ich die Tastatur dunkel. Wie sich das Design notwendigerweise verbessern musste, erfahren Sie im nächsten Kapitel **Kompatibilität**.

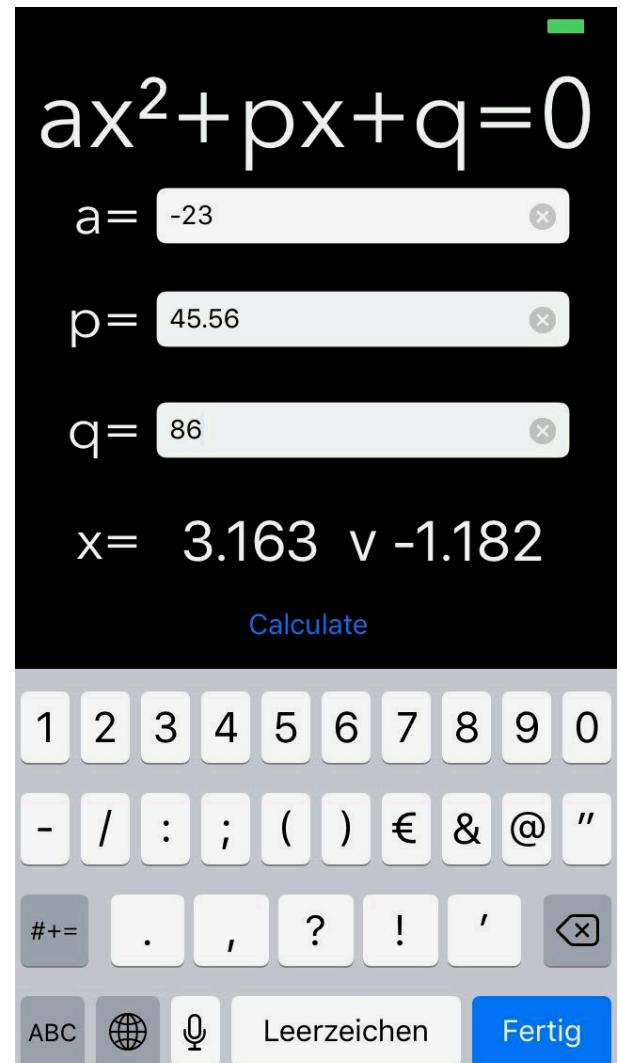


Abb. 4: erster Prototyp QuadraSolves

Kompatibilität

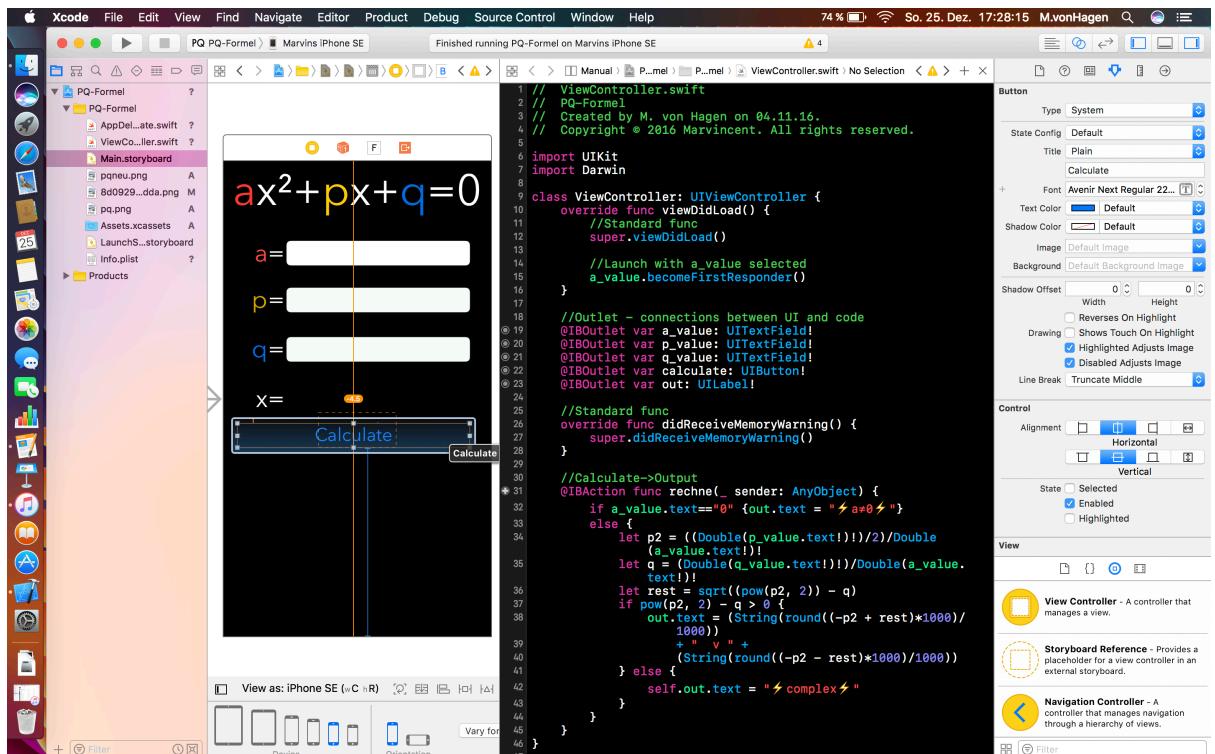
Um die App auch auf anderen iOS-Geräten als den weniger verbreiteten 4" iPhones verfügbar zu machen, waren Fine Grain Changes und Constraints³ vonnöten. Im Main.storyboard muss dazu ein Constraint geschaffen werden, der sich nach den Positionen anderer Objekte, der Bildschirmmitte und den vertikalen Begrenzungen richtet. Nach langem Probieren hatte ich endlich ein Gefühl dafür, welche Pixelanzahl zu welcher Verschiebung von Objekten führt und die App lief auf anderen Displaygrößen genauso gut.

```

Constraints
ax2+px+q=0.leading = leading + 10
ax2+px+q=0.top = Top Layout Guide.bottom - 15
ax2+px+q=0.centerX = centerX
trailing = ax2+px+q=0.trailing + 10
Set of Numbers.centerY = centerY
Calc.centerX = centerX
Calc.centerY = centerY
Calc.leading = Set of Numbers.trailing + 8
Out.centerX = centerX
trailing = Out.trailing
Out.top = Calc.bottom - 5
Out.leading = leading

```

Abb. 5: Constraints

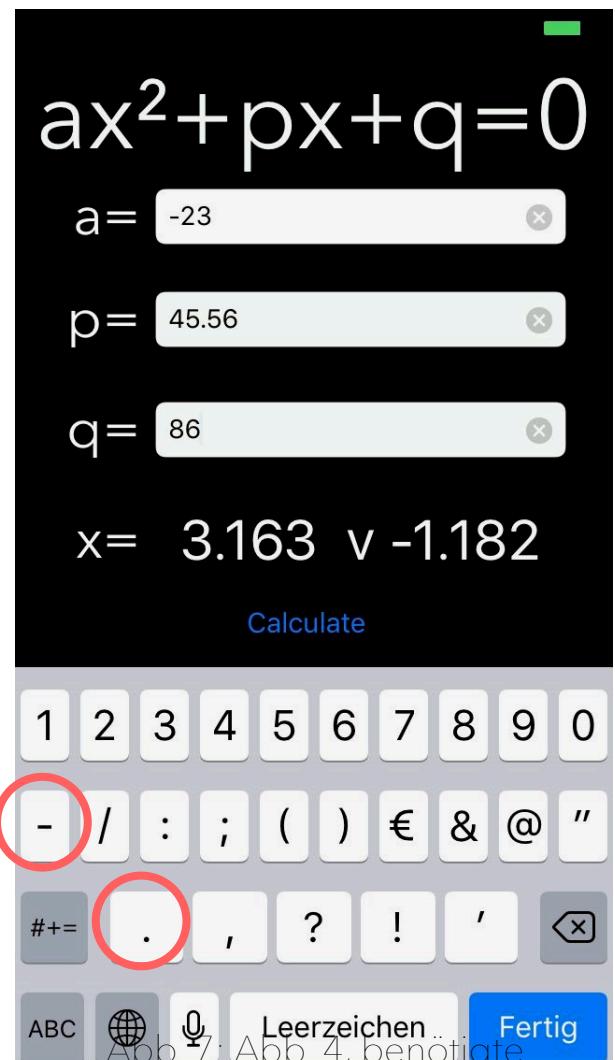


Da wir auf diesem Stand bei der Bedienung mithilfe von VoiceOver zunächst keine Probleme feststellen konnten, ließen ich die App meinen Cousin Tom testen, der sie bisher noch nicht kannte und dem ich keine weiteren Informationen - außer dass man mit der App quadratische Gleichungen lösen könne - gab, um zu erfahren, wie intuitiv sie sich mit VoiceOver bedienen lässt. Dabei fiel mir eins besonders auf: Während wir mit VoiceOver den Bildschirm nach Objekten

abtasteten und so mit dem Finger über der Schaltfläche waren, mit der wir interagieren wollten, benutzt Tom das iPhone gewissermaßen eindimensional. Er nimmt die dargestellten Informationen ähnlich auf, wie wir ein Buch lesen und wischt nur nach rechts und links, statt mit dem Finger direkt dahin zu gehen, wo das Objekt ist. So geht er Objekt für Objekt zeilenweise von oben links nach unten rechts durch und weiß dabei nie, wo sich das Auswahlfeld (s. Abb. 19) befindet. Die von mir vorgestellte Nutzungsvariante scheint nur Sehbehinderten mit schwachen Augen möglich zu sein, da sich Blinde das Layout schlicht nicht vorstellen können.

Dadurch traten mehrere Probleme auf:

- I. Bei dem Versuch, negative Werte oder Dezimalzahlen einzugeben, verlor er sich ständig in den vielen Sonderzeichen, die eigentlich nicht nötig sind (siehe Abb. 7). Um dieses Problem zu beheben, musste ich die alte Dezimaltastatur wieder aktivieren. Das Problem dieser zudem - durch weniger, nur benötigte Knöpfe - wesentlich ästhetischeren Lösung ist zum einen, dass sich auf ihr nur das Dezimaltrennzeichen der Sprache des iPhones zugehörigen Region befindet. Das heisst, dass in einem Land wie beispielsweise England unten rechts auf der Tastatur das dortige Dezimaltrennzeichen, der Punkt, vorzufinden ist, wohingegen sich in Deutschland nur Kommata eingeben lassen. Bei Eingabe eines Kommas stürzte die App aber unmittelbar ab und fast alle Lösungen, Kommata innerhalb eines Strings intern durch Punkte zu ersetzen und dann erst in Doubles zu konvertieren, waren veraltet



Sonderzeichen markiert

und verursachten Probleme. Nach langer Recherchearbeit fand ich bei Stackoverflow⁴ endlich, wonach ich suchte:

```
let aP = String(aStr.characters.map {$0 == "," ? "." : $0})
```

(Variablenamen ans Projekt angepasst)

- was dann entsprechend im Code integriert werden musste und damit möglich machte, Kommata als Dezimaltrennzeichen zu nutzen. Zum Anderen bietet die Dezimaltastatur aus unerklärlichen Gründen kein Minuszeichen und so konnten nur positive Werte eingegeben werden. Um das zu verhindern, fügte ich Plus/Minus-Buttons ein. Die Funktionen `wert.sign()` (s. Abb. 8A, Z. 33-58) sind so verknüpft, dass sie bei Klicken des Vorzeichens des jeweiligen Wertes aufgerufen werden. In der if-Schleife wird dann geprüft, was aktuell im Button steht und dementsprechend gewechselt. In der durch den „Berechne“-Button ausgelösten Funktion werden dann die Vorzeichen ausgelesen und die ausgelesenen Werte für a, p und q mit -1 multipliziert oder auch nicht.

- II. Nach Eingabe aller Werte klickte Tom auf den „Berechne“-Button und suchte das Ergebnis anschließend vergeblich bei den Zahlen auf der Tastatur, weil das Ausgabefeld für die Ergebnisse erst nach Berechnen des Ergebnisses von VoiceOver erkannt wird und dann in der Reihenfolge vor dem Button selbst erscheint. Das ist vergleichbar damit, dass der Schluss eines Buches erst nach Fertiglesen plötzlich am Anfang erscheint, wo er vorher nicht war, als man es las. Zur Behebung dieses Problems wechselten das Ausgabefeld und der „Berechne“-Button den Ort (siehe Abb. 4 -> Abb. 20) und dementsprechend musste ich wieder alle Constraints neu definieren.
- III. Beim Vorlesen von nicht ganzzahligen Ergebnissen las VoiceOver den Punkt als Tausendertrennzeichen statt Dezimaltrennzeichen. Um das zu beheben, musste ich lediglich die bereits vorhandene Funktion übertragen, die auch schon dafür sorgt, dass Kommata als Dezimaltrennzeichen akzeptiert werden.

Komplexe Zahlen

Bisherige Funktionalität: Sehende wie Blinde müssen $x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$ und

$$x_{1,2} = \frac{-p \pm \sqrt{p^2 - 4aq}}{2q}$$
 nicht mehr auswendig lernen und können schnell, einfach

und innerhalb eines ästhetischen UIs quadratische Gleichungen lösen. Um die Funktionalität weiter zu erhöhen und auch Leuten, die die entsprechenden Gleichungen auswendig können, neben dem erhöhten Komfort und der schnelleren Lösung einen Nutzen zu bieten, nahm ich mir vor, den gesamten bereits von mir verfassten Quellcode (im vorherigen Kapitel **Kompatibilität** ersichtlich) neu zu schreiben, sodass neben reellen auch komplexe Lösungen berechnet werden können. Entgegen der bisherigen Arbeit, in der es hauptsächlich um die App-Entwicklung mit Xcode ging, musste ich dafür keine Grundlagen lernen und ausprobieren, sondern vielmehr „wirklich“ programmieren und einen möglichst effizienten Algorithmus zu entwickeln, der wirklich alle Möglichkeiten für Lösungen (z.B. iPhone-Sprache aus einem Land mit Punkt als Dezimaltrennzeichen, 2 komplexe Lösungen, $x \in \mathbb{C}$) abdeckt.

Innerhalb des im folgenden Kapitel beschriebenen Codes wird folgende Lösungsmethode zur Lösung quadratischer Gleichungen verwendet:

Zunächst werden p und q (im zuvor vom Algorithmus bestätigten Fall $a \neq 0$, da es sonst zu einem Absturz käme) durch a geteilt. Dann wird die Diskriminante $dis = p^2 - 4q$ berechnet.

Ist die Diskriminante positiv, ergeben sich die beiden reellen Lösungen nach der

$$pq\text{-Formel zu: } x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

Ist die Diskriminante 0, sind die beiden Lösungen der obigen Formel identisch.

Ist die Diskriminante negativ, ergeben sich die beiden komplexen Lösungen zu:

$$x_{1,2} = -\frac{p}{2} \pm i \cdot \sqrt{q - \left(\frac{p}{2}\right)^2}$$

Wie ich das in der Programmiersprache Swift genau umgesetzt habe, ist im folgenden Kapitel zu lesen.

Der QuadraSolve-Algorithmus

Der Code der mit dem Main.storyboard verknüpften Datei ViewController.swift:



The screenshot shows the Xcode interface with the ViewController.swift file open. The window title is "iPhone SE v. Hagen". The status bar indicates "Finished running QuadraSolve on iPhone SE". The code editor shows the Swift code for the ViewController class, which handles UI interactions like button presses and manages outlets for UI elements.

```
1 // ViewController.swift
2 // QuadraSolve
3 // Created by M. von Hagen on 04.11.16.
4 // Copyright © 2016 COD3LT / Marvincent. All rights reserved.
5
6 import UIKit
7 import Darwin
8
9 class ViewController: UIViewController {
10     //Wird bei Erscheinen des Views ausgeführt
11     override func viewDidLoad() {
12         super.viewDidLoad()
13         a_value.becomeFirstResponder()
14     }
15
16     override var preferredStatusBarStyle: UIStatusBarStyle {
17         return .lightContent
18     }
19
20     // Outlets - Verbindung zwischen UI und Code
21     @IBOutlet var a_value: UITextField!
22     @IBOutlet var p_value: UITextField!
23     @IBOutlet var q_value: UITextField!
24     @IBOutlet var a_sign: UIButton!
25     @IBOutlet var p_sign: UIButton!
26     @IBOutlet var q_sign: UIButton!
27     @IBOutlet var out: UILabel!
28     @IBOutlet var calc: UIButton!
29     @IBOutlet var setOfNumbers: UIButton!
30
31     // Vorzeichen
32     @IBAction func a_sign(_ sender: Any) {
33         if a_sign.title(for: .normal)=="+{
34             a_sign.setTitle("-", for: .normal)
35         }
36         else {
37             a_sign.setTitle("+", for: .normal)
38         }
39     }
40
41     @IBAction func p_sign(_ sender: Any) {
42         if p_sign.title(for: .normal)=="+{
43             p_sign.setTitle("-", for: .normal)
44         }
45         else {
46             p_sign.setTitle("+", for: .normal)
47         }
48     }
49
50     @IBAction func q_sign(_ sender: Any) {
51         if q_sign.title(for: .normal)=="+{
52             q_sign.setTitle("-", for: .normal)
53         }
54         else {
55             q_sign.setTitle("+", for: .normal)
56         }
57     }
58
59     @IBAction func setOfNumbers(_ sender: Any) {
60         if setOfNumbers.title(for: .normal)=="R{
61             setOfNumbers.setTitle("C", for: .normal)
62         }
63         else {
64             setOfNumbers.setTitle("R", for: .normal)
65         }
66     }
67 }
```

Abb. 8A: ViewController.swift

```

 66 }
67 }
68 // Ausgabe
69 @IBAction func calc_button(_ sender: AnyObject) {
70
71     // Fehlt etwas?
72     if (a_value.text?.isEmpty)! || (p_value.text?.isEmpty)! || (q_value.text?.isEmpty)! {
73
74         if (p_value.text?.isEmpty)! && (q_value.text?.isEmpty)! && (a_value.text?.isEmpty)! {
75             self.out.text = "a = ?; p = ?; q = ?"
76             return
77         }
78
79         if (a_value.text?.isEmpty)! && (p_value.text?.isEmpty)! {
80             self.out.text = "a = ?; p = ?"
81             return
82         }
83
84         if (a_value.text?.isEmpty)! && (q_value.text?.isEmpty)! {
85             self.out.text = "a = ?; q = ?"
86             return
87         }
88
89         if (p_value.text?.isEmpty)! && (q_value.text?.isEmpty)! {
90             self.out.text = "p = ?; q = ?"
91             return
92         }
93
94
95         if (a_value.text?.isEmpty)! {
96             self.out.text = "a = ?"
97             return
98         }
99
100        if (p_value.text?.isEmpty)! {
101            self.out.text = "p = ?"
102            return
103        }
104
105        if (q_value.text?.isEmpty)! {
106            self.out.text = "q = ?"
107            return
108        }
109
110    }
111
112    // Auslesen der Textfelder
113    var a: Double
114    var p: Double
115    var q: Double
116
117
118    var aStr = a_value.text!
119    let aP = String(aStr.characters.map {$0 == "," ? "." : $0})
120    if a_sign.title(for: .normal)=="+"{
121        a = Double(aP)!
122    }
123    else {
124        a = -1*Double(aP)!
125    }
126
127    var pStr = p_value.text!
128    let pP = String(pStr.characters.map {$0 == "," ? "." : $0})
129    if p_sign.title(for: .normal)=="+"{
130        p = Double(pP)!/a
131    }
132    else {
133        p = (-1*Double(pP)!)/a
134    }
135
136    var qStr = q_value.text!
137    let qP = String(qStr.characters.map {$0 == "," ? "." : $0})
138    if q_sign.title(for: .normal)=="+"{
139        q = Double(qP)!/a
140    }
141    else {
142        q = (-1*Double(qP)!)/a
143    }
144
145    // Ausgabe
146    if a==0 {
147        out.text = "a ≠ 0"
148    } else {
149        let dis = pow(p, 2) - 4*q
150        let uSqrt = pow(p/2, 2) - q
151        var r1 = String(round((-p/2 + sqrt(uSqrt))*1000)/1000)
152        var r2 = String(round((-p/2 - sqrt(uSqrt))*1000)/1000)
153
154        let lan = calc.title(for: .normal)
155        var comma = false
156        if lan == "Berechnen" || lan == "Calculer" || lan == "Calcolare" || lan=="Calcular" || lan ==
157            "подсчитывать" || lan == "Izračunati" || lan == "Beräkna" || lan == "Hesaplamak" || lan == "Obliczać"
158            || lan == "Calcular" {
159                comma = true
160            }
161    }

```

Abb. 8B: ViewController.swift



The screenshot shows the Xcode IDE interface with the file 'ViewController.swift' open. The code is written in Swift and handles various calculations and language translations for a quadratic solver application.

```
157     comma = true
158 }
159 if comma==true {
160     r1 = String(r1.characters.map {$0 == "." ? "," : $0})
161     r2 = String(r2.characters.map {$0 == "." ? "," : $0})
162 }
163
164 if dis > 0 {
165     out.text = "x1=\(r1) v x2=\(r2)"
166 }
167 else if dis == 0{
168     out.text = "x= \(r1)"
169 }
170 else if setOfNumbers.title(for: .normal)== "C"{
171     var u = String(round(-p/2*1000)/1000)
172     var b = String(round(sqrt(-1*uSqrt)*1000)/1000)
173
174     if comma==true {
175         u = String(u.characters.map {$0 == "." ? "," : $0})
176         b = String(b.characters.map {$0 == "." ? "," : $0})
177     }
178
179     if p != 0{
180         out.text = "x= \((u) ± \((b)i"
181     }
182     else {
183         out.text = "x= ± \((b)i"
184     }
185 }
186 else if setOfNumbers.title(for: .normal)== "R"{
187
188     if lan == "Berechne" {
189         out.text = "Keine Lösung"
190     }
191     if lan == "Calculate" {
192         out.text = "No Solution"
193     }
194
195     if lan == "Calculer" {
196         out.text = "Pas de solution"
197     }
198
199     if lan == "計算" {
200         out.text = "沒有解決方案"
201     }
202
203     if lan == "Calcular" {
204         out.text = "Sin solución"
205     }
206     if lan == "计算" {
207         out.text = "没有解决方案"
208     }
209
210     if lan == "Calcolare" {
211         out.text = "Nessuna soluzione"
212     }
213
214     if lan == "подсчитывать"{
215         out.text = "Нет решения"
216     }
217
218     if lan == "Calcular"{
219         out.text = "Sem solução"
220     }
221
222     if lan == "計算する"{
223         out.text = "ソリューションなし"
224     }
225
226     if lan == "Izračunati"{
227         out.text = "Nema rješenja"
228     }
229
230     if lan == "Beräkna"{
231         out.text = "Ingen lösning"
232     }
233
234     if lan == "Hesaplamak"{
235         out.text = "Çözüm yok"
236     }
237
238     if lan == "عَدْ"{
239         out.text = "لَا يَعْدُ"
240     }
241
242     if lan == "Obliczać"{
243         out.text = "Brak rozwiązania"
244     }
245
246 }
247 }
248 }
249 }
```

Abb. 8C: ViewController.swift

Der QuadraSolve-Algorithmus - Erläuterung

- 6 Importiert die Klasse *UIKit*, um mit dem Main.storyboard kommunizieren zu können.
- 7 Importiert die Klasse *Darwin*, um auf mathematische Funktionen wie beispielsweise *pow()* zugreifen zu können.
- 9 Definiert die Klasse *ViewController* als Vererbung der vordefinierten Klasse *UIViewController*.
- 11 Definiert die Funktion *viewDidLoad()*, deren Code automatisch bei Erscheinen des Views ausgeführt wird.
- 12 Ruft die Methode *viewDidLoad()* der Superklasse *UIViewController* auf, um den View zu initialisieren.
- 13 Beim Öffnen QuadraSolves - ohne dass es bereits vorher im Hintergrund geöffnet war - wird automatisch das Textfeld für den Wert der Variable *a* ausgewählt, sodass direkt auf der Tastatur getippt werden kann.
- 16 Die Farbe der Status Bar wird von standardmäßigem Schwarz zu Weiß geändert, um trotz des schwarzen Hintergrund sichtbar zu sein.
- 21 Alle *@IBOutlets* werden definiert und sind per UI innerhalb Xcodes mit den entsprechenden UI-Objekten im Main.storyboard verbunden, was an den Punkten links der Zeilenangabe ersichtlich ist.
- 33 Die Funktionen der Vorzeichenbuttons werden definiert und per *@IBAction* ähnlich wie die *@IBOutlets* mit den jeweiligen Objekten im Main.storyboard verbunden, sodass bei Berührungen die jeweiligen Funktionen ausgeführt werden. Dabei wird zunächst geprüft, ob der aktuelle Titel (also das, was der Button anzeigt) ein „+“ ist. Falls dies *true* ist, wird der Titel zu „-“ geändert. Falls es nicht *true* ist, wird der Titel entsprechend zu „+“ geändert.
- 60 Definiert die Funktion *setOfNumbers()*, die den Button zur Festlegung der Lösungsmenge für x festlegt. Dabei gehe ich nach dem selben Schema wie bei den vorigen Vorzeichenbuttons vor.
- 69 Definiert die Funktion des „Berechne“-Buttons, der die Ausführung des auf den nächsten beiden Seiten folgenden Codes initiiert.

- 72 Prüft, ob und wenn ja welche Textfelder leer sind, und gibt eine dementsprechende Ausgabe des Schemas „p=?, q=?“. Hierzu lesen Sie Ausführlicheres in Vincents Projektarbeit.
- 114 Initialisierung der drei Variablen des Typs Double **a**, **p** und **q**
- 118 Der Variablen **aStr** (aString) wird der ungewrappte String-Wert des String Optionals **a_value.text** - des im a-Textfeld stehendes Strings - zugewiesen.
- 119 Der Konstanten **aP** (aPoint) wird ein String zugewiesen, der der Version **aStrs** entspricht, in der durch den nachstehenden Befehl alle Chars des Strings ausgelesen und im Fall eines Kommas durch Punkte ersetzt wurden, um eine spätere Konvertierung zu einem Double zu ermöglichen.
- 120 Zunächst wird der Titel des links neben dem Textfeld stehenden Buttons ausgelesen und, falls es sich dabei um „+“ handelt, wird der in Z. 114 initialisierten Variable **a** der Wert der zum Double-Optional konvertierten Variable **aP** zugewiesen und anschließend ungewrappt. Ist der Titel des Buttons nicht „+“, geschieht dasselbe, jedoch wird der Double-Wert mit -1 multipliziert.
- 127 Das ab Z. 118 Beschriebene wird nun mit den Variablen **p** und **q** durchgeführt, jedoch werden die abschließenden Double-Werte durch **a** geteilt.
- 146 Es wird geprüft, ob **a** gleich Null ist, und gegebenenfalls erfolgt die Ausgabe „**a**≠0“. Andernfalls wird der folgende Code ausgeführt.
- 149 Die Konstante **dis** (Diskriminante) wird berechnet, um herauszufinden, wieviele reelle und komplexe Nullstellen die quadratische Gleichung aufweist.
- 150 Die Konstante **uSqrt** (underSquareRoot) wird berechnet, die der Rechnung unter der Wurzel der pq-Formel entspricht.
- 151 Der Variablen **r1** (realSolution1) wird der zum String konvertierte Wert der größeren Lösung der pq-Formel zugewiesen, wobei das Double-Ergebnis der Rechnung zunächst mit 1000 multipliziert wird, dann als ganze Zahl gerundet wird und abschließend wieder durch 1000 dividiert wird, wodurch **r1** auf drei Nachkommastellen gerundet wird. Anschließend wird der Variable **r2** nach dem selben Prozedere die kleinere Lösung der PQ-Formel zugewiesen.

- 154 Der Konstanten *lan* (language) wird der Titel des „Berechne“-Buttons zugewiesen.
- 155 Der Variablen **comma** wird der boolsche Wert **false** zugewiesen.
- 156 Falls *lan* einem der möglichen „Berechne“-Button Titel entspricht, die bei iOS-Geräten angezeigt werden, auf denen eine Sprache installiert ist, deren Ursprungsland als Dezimaltrennzeichen das Komma festgelegt hat, wird *comma* der Wert true zugewiesen.
- 159 Falls **comma** true ist, werden nach bereits beschriebenem Verfahren die Chars der Zeichenketten **r1** und **r2** durchgegangen und im Fall eines Punktes durch ein Komma ersetzt.
- 165 Falls **dis** positiv ist und damit zwei verschiedene reelle Lösungen für die eingegebene quadratische Gleichung existieren, werden **r1** und **r2** mit Hilfe von String-Interpolation in den String eingesetzt, dem der Text des Ausgabefelds **out** (output) gleichgesetzt wird.
- 168 Fall **dis** gleich Null ist und damit genau eine reelle Lösung für die quadratische Gleichung existiert, wird die Lösung nach selbem Schema ausgegeben.
- 171 Falls **dis** negativ ist und demnach keine reellen Lösungen für x existieren, wird geprüft, ob als Lösungsmenge die komplexen Zahlen eingestellt sind und in diesem Fall wird der folgende Code ausgeführt.
- 172 Der Variablen **u** (upFront) wird der zum String konvertierte nach bereits beschriebenem Verfahren gerundete Wert des bei komplexen Lösungen vorderen Teils zugewiesen. Nach gleichem Schema wird der Variablen **b** (behind) der hintere Teil als gerundeter String zugewiesen.
- 175 Falls **comma** true ist, werden **u** und **b** - genau wie in Z. 159 **r1** und **r2** - in die hier herkömmliche Form mit dem Komma als Dezimaltrennzeichen gebracht.
- 180 Falls **p** ungleich Null ist, werden nach beschriebenen Verfahren sowohl der reelle Teil als auch der komplexe Teil mit i ausgegeben. Andernfalls weist die Lösung der quadratischen Gleichung keinen reellen Teil auf und dementsprechend wird nur der komplexe Teil ausgegeben.
- 187 Falls die reellen Zahlen die Lösungsmenge für x sind, wird „Keine Lösung“ in der jeweiligen Sprache ausgegeben; näheres hierzu in Vincents Projektarbeit.

Weltweite Distribution im App Store

Um unseren Traum, die eigens entwickelte App auf über einer Milliarde iOS-Geräten weltweit nutzbar zu machen, zu verwirklichen, mussten wir sie im App Store anbieten, da dieser für Konsumenten die einzige Möglichkeit darstellt, Applikationen auf iOS-Geräten zu installieren. Dazu wird eine Teilnahme am kostenpflichtigen Dienst „Apple Developer Program“⁵ vorausgesetzt. Ich teilte mir den Preis von 99 € für eine Jahresmitgliedschaft mit Vincent, woraufhin uns der Zugang zu iTunes Connect⁶, Apples Tool zum Verwalten von im App Store angebotenen Inhalten wie in diesem Fall Apps, freigeschaltet wurde:

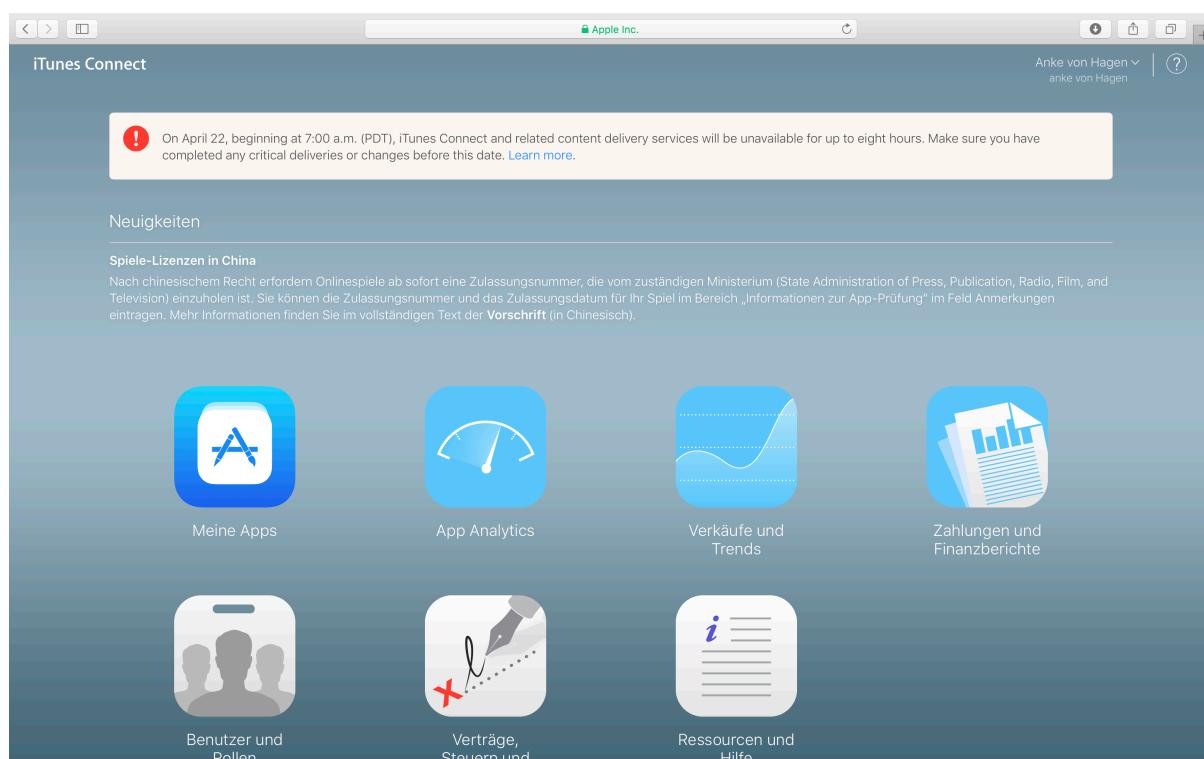


Abb. 9: iTunes Connect

Um zu erläutern, welche Schritte ich zur Veröffentlichung im App Store vornehmen musste, stelle ich hier die einzelnen Bereiche von iTunes Connect vor:

Meine Apps

Zunächst ist eine Übersicht der Projekte, an denen man aktuell arbeitet, zu sehen; neben der Möglichkeit, ein neues Projekt zu starten. Wählt man nun ein Projekt - wie in diesem Fall QuadraSolve - aus, kann man alle die Distribution dieser App betreffenden Informationen bearbeiten. In Zusammenarbeit mit Vincent verfasste ich die App Store-Beschreibung in deutsch und englisch (s. S. 30) und Schlüsselwörter „Quadratische Gleichungen, $ax^2+px+q=0$, $x^2+px+q=0$, komplex, i,

iOS-App 1.1

Bereit zum Verkauf

Neues in dieser Version

- A bug concerning the layout has been fixed
- Improvements concerning extremely high and low numbers: autoshrinking in case of long variables; extreme long solutions are now shown in scientific notation instead of running out to the sides
- Optical improvements like the variably sized $ax^2+px+q=0$ and a new font
- Increased efficiency due to String interpolation

App Previews and Screenshots

iPhone iPad

5,5-Zoll Display

Medienverwaltung

Abb. 10: iTunes Connect: Meine Apps

VoiceOver, PQ Formel, Mitternachtsformel, PQ. Dazu stellten wir uns die Frage, wonach Nutzer im App Store suchen, wenn sie eine App wie QuadraSolve gebrauchen könnten. Des Weiteren luden wir 34 Screenshots aller Displaygrößen in iTunes Connect, um Nutzern jeder möglichen Displaygröße eine geeignete Vorschau zu präsentieren und sie beim Aufruf der Produktseite möglichst erfolgreich zu einem Download zu motivieren. Da eine Support-URL gefordert wurde, erstellten wir eine Twitter-Seite⁷ (siehe Anhang), zudem erstellte ich den Alias "QuadraSolve@iCloud.com" für meine eMail-Adresse. Da die App hier zur Prüfung und anschließenden Veröffentlichung dem App Store Team übermittelt

Name	Creation Date	Version
QuadraSolve	23.04.2017, 17:10	1.1 (1)
QuadraSolve	23.04.2017, 16:55	1.1 (1)
QuadraSolve	29.01.2017, 17:03	1.0 (1)
QuadraSolve	29.01.2017, 16:42	1.0 (1)
QuadraSolve	29.01.2017, 16:37	1.0 (1)

Archive Information

QuadraSolve
23.04.2017, 17:10

Upload to App Store...

Validate... Export...

Details

Version 1.1 (1)
Identifier com.M.vonHagen.Quadra...
Type iOS App Archive

Download dSYMs...

Description

Abb. 11: Xcode Archives

wird, findet sich hier auch die aktuelle Build, nachdem sie zuvor mit Xcode hochgeladen wurde.

Zunächst musste ich in Xcode unter Product QuadraSolve archivieren. Daraufhin stand mir die archivierte Build QuadraSolves unter Archives für folgende Aktionen zur Verfügung (s. Abb. 11).

Ich wählte Upload to App Store, was die App allerdings nicht in den App Store lädt, sondern nur die Build innerhalb von iTunes Connect hochlädt und dem dort entsprechend erstellten Projekt mit demselben Bundle-Identifier zuweist.

App Analytics

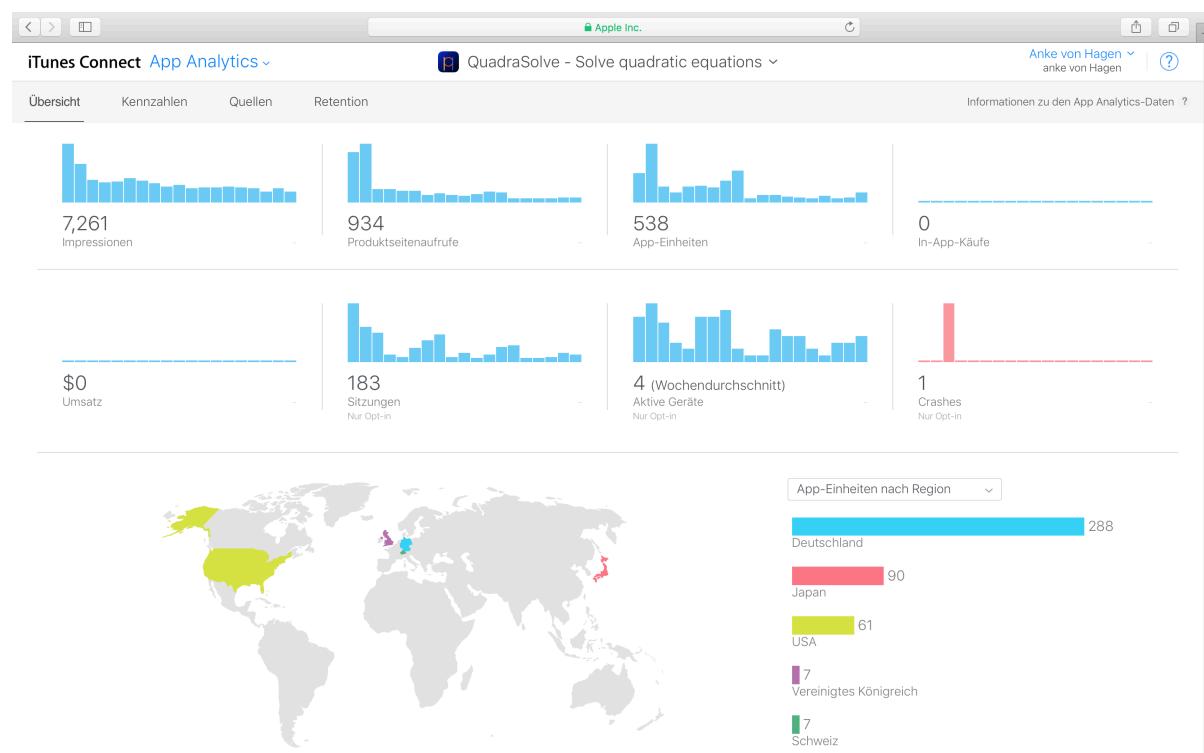


Abb. 12: iTunes Connect: App Analytics

Neben den hier abgebildeten Statistiken lässt sich in den App Analytics ebenso einsehen, dass 71% der QuadraSolve-Installationen auf iPhones erfolgen und 29% auf iPads und viele weitere Daten, die größtenteils aber nicht wirklich von Bedeutung sind. Viele Statistiken wie „Sitzungen“, „Aktiven Geräten“ und „Retention“ beziehen sich nur auf „Opt-In“-Geräte, die Geräte, bei deren Inbetriebnahme zugestimmt wurde, Diagnose- und Nutzungsinformationen mit App-Entwicklern (wie in diesem Fall uns) zu teilen. Bedauerlicherweise wurde diese Option nur bei weniger als jedem zehnten Gerät mit QuadraSolve-Installation gewählt, womöglich aus Sorge vor Datenmissbrauch.

Verkäufe und Trends

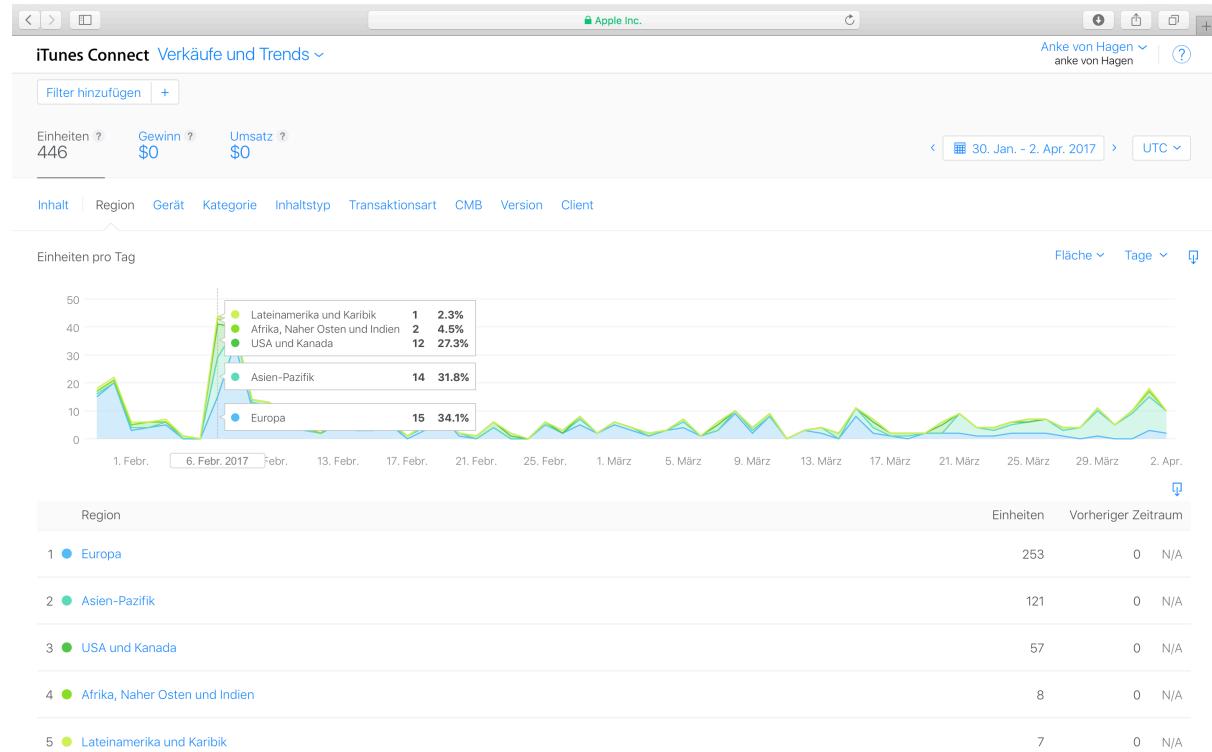


Abb. 13: iTunes Connect: Verkäufe und Trends

Nach den unterschiedlichsten Faktoren gefiltert, lassen sich die Downloads jedes Tages darstellen. Dabei ist beispielsweise einzusehen, dass QuadraSolve bereits aus 43 Ländern aller Kontinente heruntergeladen wurde.

Zahlungen und Finanzberichte

Am Verlauf der heruntergeladenen Einheiten pro Tag bei Verkäufe und Trends können Sie sehen, dass wir an einigen Tagen keinen Download verzeichnen konnten. Das sind die Tage, an denen QuadraSolve für 0,99€ zum Verkauf stand. Da wir dementsprechend noch keinen Umsatz erzielen konnten, ist diese Rubrik für uns aktuell noch irrelevant.

Benutzer und Rollen

Die Rollen Rechtliches, Administrator, Finanzen, App-Manager, Entwickler, Marketing, Verkauf und Berichte können den verschiedenen Teammitgliedern zugewiesen werden; dementsprechend habe ich Vincent hier als Entwickler hinzugefügt. Zudem besteht die Möglichkeit, mithilfe von TestFlight⁸ Betatester hinzuzufügen, was in unserem Fall aber nicht benötigt wurde, da wir uns nach vorgenommenen Änderungen die überarbeiteten Dateien schickten und über Xcode auf unseren Geräten testeten.

Verträge, Steuern und Bankverbindungen

Hier musste ich diverse Verträge ausfüllen, in denen nach zahlreichen Daten wie dem Beruf meiner Mutter gefragt wurde und in denen ich z.B. der amerikanischen Steuerbehörde zusagen musste, dass ich nichts im App Store anbieten werde, an dem jemand mitgewirkt hat, der länger als 182 Tage im letzten Jahr in den USA war. Nachdem sich unterschiedliche Steuerbehörden und das App Store Team der Korrektheit jener Verträge vergewisserten, war meine Apple ID Ende Januar für den Verkauf im App Store zugelassen.

Vermarktung

Nachdem ich im vorangegangenen Kapitel bereits auf die Downloadzahlen, die weltweite Reichweite und weitere Statistiken eingegangen bin, werde ich in diesem Kapitel darstellen, wie wir es geschafft haben, über 500 Leute aus 43 Nationen dazu zu bringen, QuadraSolve zu nutzen.

Icon

Bei der Suche nach einer bestimmten App ist neben dem dargestellten Screenshot und dem Namen der wohl wichtigste Faktor bei der Entscheidung, ob man auf die Produktseite geht oder nicht, das Icon.

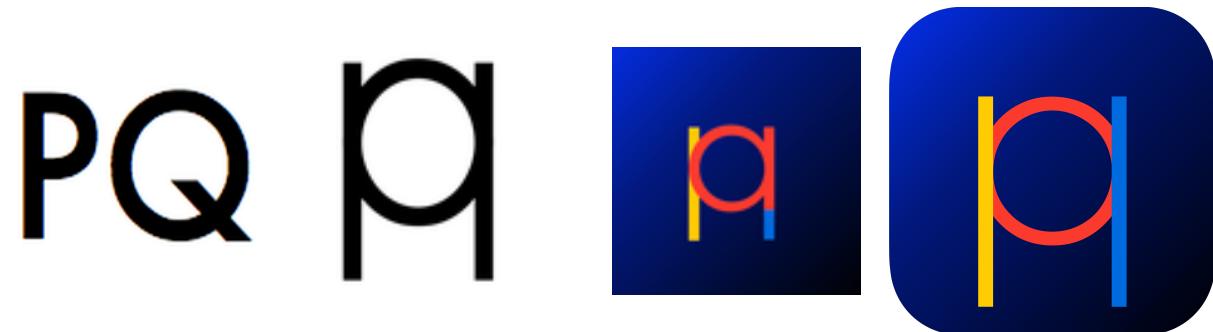


Abb. 14-17: Entwicklung des QuadraSolve-Icons

Hier sind die vier wichtigsten Versionen meines Icon-Designs dargestellt. Das markante Muster mit dem Kreis und den beiden Linien vereint die drei Parameter a , p und q und besitzt einen hohen Wiedererkennungswert. Dabei wird in den beiden neueren Versionen auf die innerhalb der App benutzten Farben für die jeweiligen Parameter zurückgegriffen.

Die Icons habe ich mit Pages erstellt und mit der macOS-App Vorschau auf 1.536x1.536 Pixel skaliert. Dann habe ich sie bei <http://makeappicon.com> hochgeladen, die mir daraufhin eine .zip-Datei

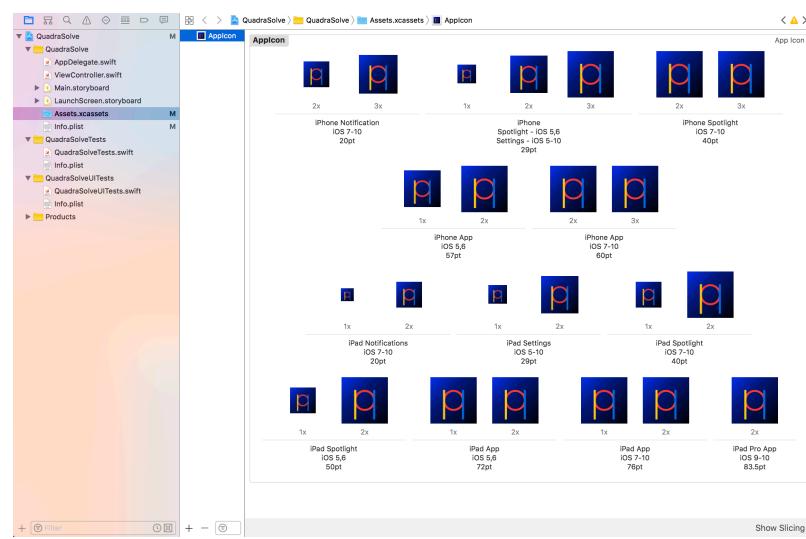


Abb. 18: Assets.xcassets

mit allen benötigten Größen geschickt haben. Dann musste ich sie alle in Xcode in der Datei *Assets.xcassets* hinzufügen. Wie im rechst eingebundenen Screenshot ersichtlich, werden intern nur quadratische App-Icons akzeptiert, die dann aber immer in der typischen, gerundeten „Appform“ angezeigt werden. Für diese Projektarbeit erstellte ich dennoch eine gerundete Version mithilfe eines Onlinetools⁹.

Werbung

Um QuadraSolve zu verbreiten, warben wir sowohl in der Schule als auch in unserem privaten Umfeld für QuadraSolve. In der Schule sind wir zunächst zum Schulleiter des Maria-Wächtler-Gymnasiums, Herrn Korthaus, gegangen, um ihm unsere App vorzustellen. Gemeinsam erstellten wir eine Liste aller Lehrer, die aktuell in der 8.-10. Klasse Mathematik unterrichten.

Daraufhin sprachen wir mit diversen Lehrern und besuchten beispielsweise eine 9. Klasse, die von Herrn Müller unterrichtet wird. Obwohl unsere primäre Intention in der Verbreitung unserer App lag, begeisterten sich diverse MINT-interessierte Schüler mehr für das Konzept eines Projektkurses und die damit verbundene Aussicht auf eine eigene, im App Store weltweit verfügbare App.

Als wir in unserem Mathematik LK QuadraSolve vorstellten, erklärte sich Frau Füth bereit, eine Rundmail in der Fachschaft Mathe zu schreiben. Daraufhin luden sich mehrere Lehrer unsere App herunter undgaben uns ebenso positives Feedback.

In meinem privaten Umfeld empfahl ich die App, dabei natürlich auch meinem Cousin Tom. Er zeigt sich immer noch so begeistert wie letztes Jahr, als ich ihm das erste Mal von meiner Idee erzählte, und empfahl sie seinen Freunden. Im Literaturverzeichnis findet sich der entsprechende Link¹⁰.

Update 1.1

- In der englischen Version sind nun in der Ausgabe Punkte und nicht wie bisher Kommata

Dazu musste ich lediglich „Calculate“ an der Stelle, an der die „Kommaländer“ definiert werden, entfernen.

- Behebung eines Bugs, durch den das Layout gelegentlich „durcheinander“ kam

Dazu musste ich lediglich den richtigen Haken in Xcode unter Targets →QuadraSolve →Deployment Info deaktivieren.

- Verbesserung im Umgang mit extrem großen oder kleinen Zahlen:

Automatische Verkleinerung der Schriftgröße bei längeren Zahlen und extrem große oder kleine Lösungen werden fortan in wissenschaftlicher Notation ausgegeben, statt wie bisher an beiden Seiten aus dem Bild zu laufen

Viele optische Verbesserungen wie z.B. die variable Größe des oben stehenden „ $ax^2+px+q=0$ “ und die neue Schriftart

Ich habe diverse Einstellungen im Project Inspector verändert und im Size Inspector neue Constraints erstellt.

- Um den Code effizienter zu gestalten und zu verkürzen, wurde String-Interpolation implementiert
- Die Status Bar ist nun weiß und damit sichtbar

Dies ging mithilft eines bei Stackoverflow¹¹ gefundenen Codeschnipsels (siehe **Der QuadraSolve-Algorithmus**)

Des Weiteren hat Vincent die Anzahl der unterstützten Sprachen von 11 auf 15 Sprachen erhöht, da QuadraSolve zum Zeitpunkt des Updates bereits in 43 Nationen aller Kontinente heruntergeladen wurde.

Nachdem die Build der neuen Version 1.1 fertig war, lud ich sie wie bei Version 1.0 mit Hilfe des Application Loaders in iTunes Connect und übermittelte sie schließlich nach ziemlich den selben Schritten dem App Store Team. All das ging nun allerdings erheblich schneller, da ich bereits vorher genau wusste, was ich zu tun hatte.

QuadraSolve 1.2 Darstellung des Endproduktes

Deutsche/Englische App Store-Beschreibung

Keine Lust mehr, wertvolle Zeit für das Lösen quadratischer Gleichungen zu verschwenden?

Mit QuadraSolve gehören Fehler und ungelöste komplexe Ergebnisse der Vergangenheit an!

Besondere Aufmerksamkeit haben wir der VoiceOver-Kompatibilität geschenkt, um die Applikation barrierefrei zu gestalten und auch visuell eingeschränkten Personen die Nutzung zu ermöglichen.

Das minimalistische, intuitive Design ist selbsterklärend:

Ein Tipp auf ein Textfeld und schon kann man den entsprechenden Wert eingeben. Klicke auf das "+" neben dem Textfeld um das Vorzeichen zu ändern. Jetzt nur noch auf "Berechnen" drücken und schon werden alle Lösungen für x angezeigt, darunter die komplexen Lösungen (i).

QuadraSolve wurde von zwei Schülern ohne jegliche Hilfe Erwachsener entwickelt. Die 15 wichtigsten Sprachen werden unterstützt, falls Sie noch eine wünschen oder uns Verbesserungsvorschläge für die App mitteilen möchten, freuen wir uns, von Ihnen kontaktiert zu werden:

QuadraSolve@iCloud.com

M. von Hagen & V. von Oepen

Fed up with using your valuable time for solving quadratic equations?

Use this easy-to-use app to save time, effort and avoid mistakes when dealing with $ax^2+px+q=0$. Special emphasis was put on making our application accessible for visually impaired people, as it is completely compatible with VoiceOver.

The minimalist, intuitive design explains itself: Just click on a text field to enter the value and on the plus on the left to change its sign. A click on the "Calculate" Button will instantly reveal all solutions for x, including the complex roots (i).

QuadraSolve was completely developed and designed by two underage students without any aid of adults. It supports the 15 most important languages, if you wish another one or have ideas how to improve this app feel free to contact us:

QuadraSolve@iCloud.com

M. von Hagen & V. von Oepen

Update 1.2

In den bisherigen Versionen QuadraSolves wurden Nutzer, die noch nie von komplexen Zahlen gehört haben, ohne Vorwarnung mit ihnen konfrontiert, wenn die zu lösende quadratische Gleichung keine reellen Lösungen aufwies. Frau Mielchen-Woköck wies mich darauf hin, dass der Nutzer die Möglichkeit haben sollte, bei komplexen Ergebnissen nur die Ausgabe „Keine Lösung“ zu erhalten.

Ich realisierte dies mithilfe eines Buttons, der das aktuell eingestellte Mengenzeichen anzeigt. Nutzer, die nur mit reellen Zahlen arbeiten möchten, kennen wahrscheinlich das Zeichen \mathbb{R} nicht und klicken demnach auch nicht darauf (s. Abb. 19). Nutzer, die auch mit komplexen Zahlen arbeiten möchten, verstehen das Zeichen und durch einen Klick erscheinen auch die komplexen Lösungen. Zudem setzte ich auch kleinere Verbesserungsvorschläge wie z.B. $x_1=\dots$ v $x_2=\dots$ statt wie bisher $x=\dots$ um.

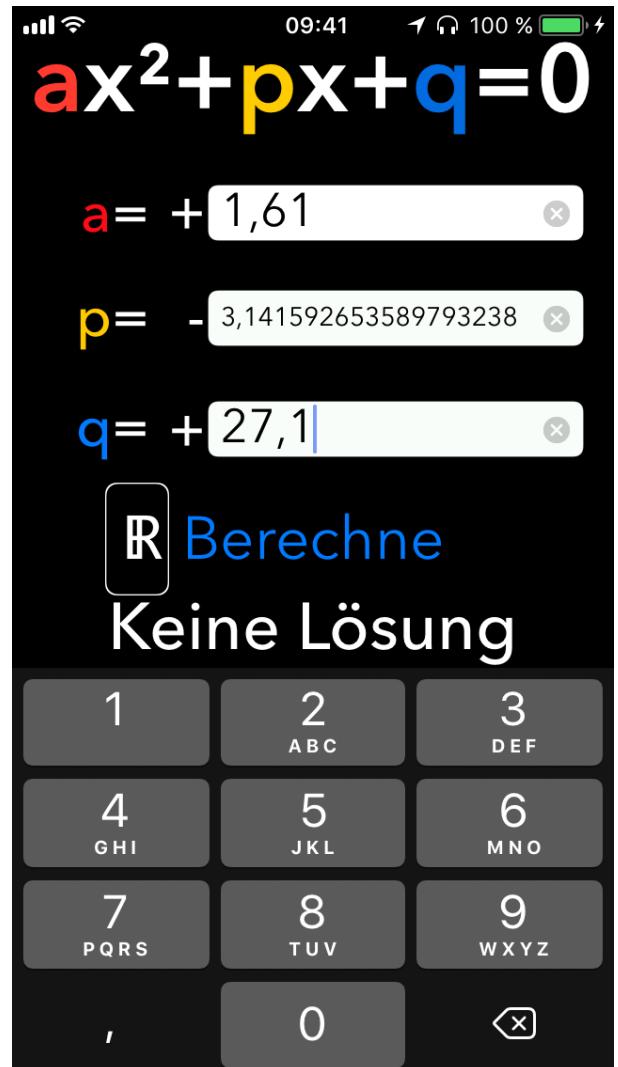


Abb. 19: QuadraSolve - keine reellen Lösungen, VoiceOver an

Bedienung

Während die Bedienung als Sehender intuitiv und selbsterklärend sein dürfte (Werte für Parameter eingeben, ggfs. Vorzeichen und Lösungsmenge anpassen, „Berechne“ drücken), benutzen sie Blinde wie auf den Seiten 12/13 dargestellt - sie wischen mithilfe von VoiceOver ein gerundetes Rechteck nach rechts und links, das den Inhalt der aktuellen Schaltfläche vorliest. Bei Abb. 19 befindet es sich beispielsweise um den Lösungsmengenbutton und liest in der vom Nutzer eingestellten Sprache „Menge der reellen Zahlen“ vor. Ansonsten gestaltet sich die Bedienung ähnlich wie bei Sehenden.

Screenshot zur Veranschaulichung der endgültigen Version

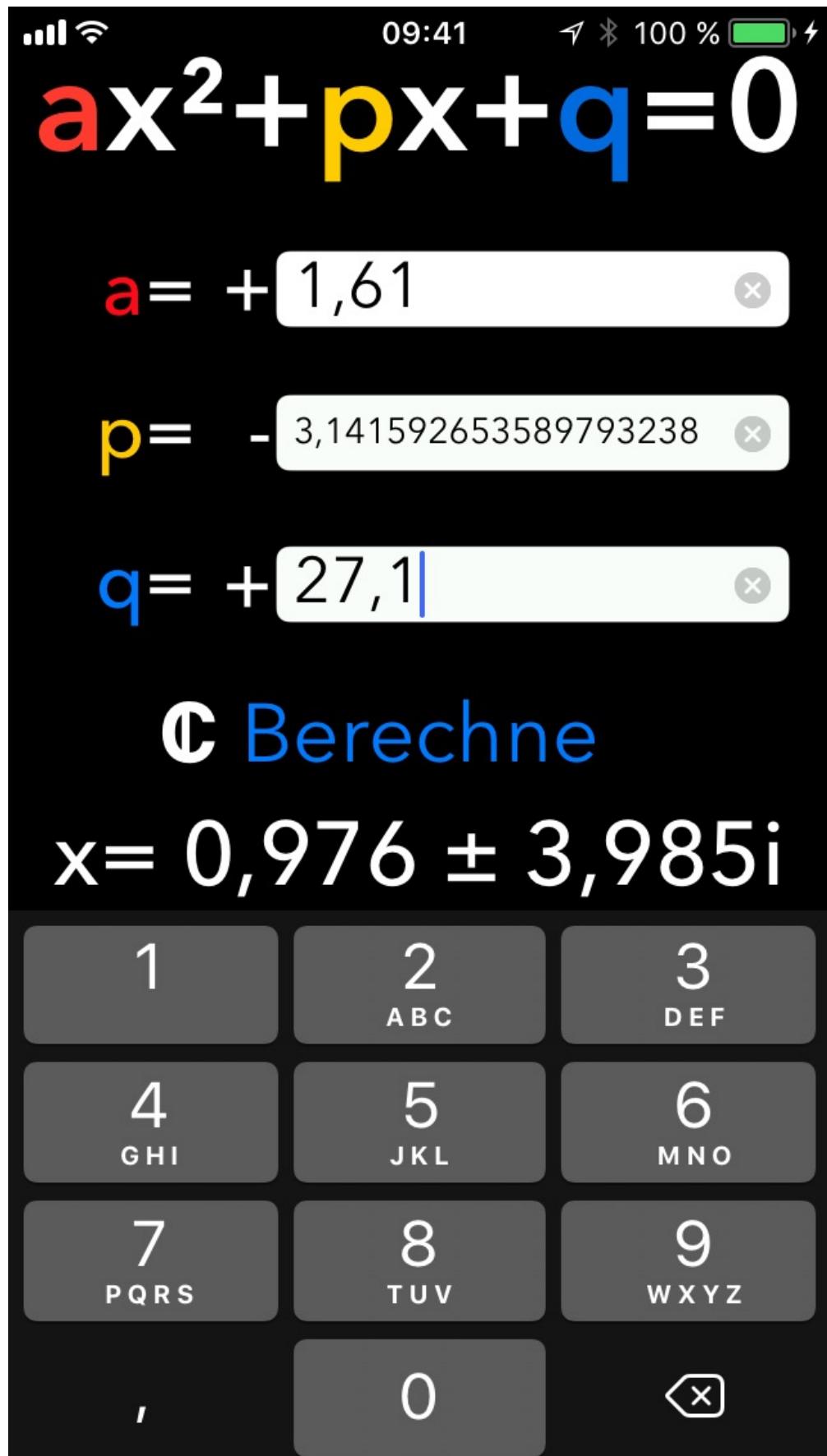


Abb. 20: QuadraSolve - 2 komplexe Lösungen, $x \in \mathbb{C}, 4"$

Eigener Arbeitsteil

Nach der gemeinsamen Auseinandersetzung mit Xcode und Swift entwickelte ich unterschiedliche Versionen einer App zur Berechnung der pq-Formel. Im weiteren Verlauf entwickelte ich die App QuadraSolve und designe sie gemäß den iOS Human Interface Guidelines¹². Ich programmierte fast den gesamten Code selbstständig, der alle reellen und komplexen Lösungen für quadratische Gleichungen ermitteln kann. Des Weiteren passte ich das Main.storyboard mit Constraints an alle Displaygrößen an. Zusätzlich gestaltete ich alles VoiceOver-kompatibel und passte es nach Tests mit Blinden entsprechend an. Danach befasste ich mich mit iTunes Connect und distribuierte QuadraSolve durch den App Store weltweit. Anschließend verbesserte ich alle von Nutzern bemerkten Fehler durch das bereits erfolgte Update 1.1. Am Ende setzte ich die von Frau Mielchen-Woköck geäußerten Erweiterungsvorschläge in Form der Version 1.2 Build 2 um.

Nachwort

Im Nachhinein denke ich mir, dass ich, wenn man mir jetzt ein Jahr Zeit gäbe, eine komplexere App mit mehr Funktionen entwickeln könnte. Allerdings wies ich zu Beginn dieses Projektkurses keine Erfahrung in nahezu allen benötigten Gebieten auf und demnach bin ich sehr zufrieden mit meiner geleisteten Arbeit, da wirklich alle meine Ziele wie z.B. die Fähigkeit, am Ende eigene Apps entwickeln zu können und vielleicht sogar selbst eine im App Store anzubieten, erreicht wurden. Innerhalb von weniger als einem Jahr haben wir von Grund auf eine eigenständige App entwickelt und sie im App Store weltweit zur Verfügung gestellt. Allein in den ersten beiden Monaten luden sie über 500 Kunden herunter, als wäre sie Produkt eines professionellen IT-Unternehmens. Ich bin jetzt in der Lage, in erheblich kürzerer Zeit Apps zu entwickeln, verfüge über ein tiefgründiges Verständnis von objektorientierter Programmierung und beherrsche alles, was zur App-Entwicklung und -Distribution mit Xcode und Swift nötig ist. Ich habe mich mit der Funktionsweise von Algorithmen auseinandergesetzt und lange daran gearbeitet, meinen eigenständig entwickelten Algorithmus effizienter und klarer zu gestalten.

Im November werde ich voraussichtlich mit Simon Bohnen, der gerade an der Android-Portierung arbeitet, an Jugend forscht¹³ teilnehmen und im Zuge dessen auch nach Ende dieses Projektkurses weiter an QuadraSolve arbeiten.

Das alles hat mir sehr viel Spaß bereitet und meine Zukunftspläne mehr geformt alles alles Bisherige, sodass ich mich entschieden habe, Informatik zu studieren. Zur weiteren Vorbereitung werde ich in den nächsten Monaten an diversen Events wie Code+Design¹⁴ und Jugend Hackt¹⁵ teilnehmen. Voraussichtlich werde ich dann zum Wintersemester nächsten Jahres mein Bachelorstudium in Informatik an der TU München beginnen. Je nach Verlauf werde ich mich innerhalb der Informatik spezialisieren und das Masterstudium in den USA absolvieren, mit dem ich dann in die Berufswelt einsteige. Diese Projektarbeit ist die wichtigste Prüfung in meiner gesamten Schullaufbahn und selbst wenn ich mich in zehn Jahren im Silicon Valley bewerben sollte, wird diese Projektarbeit Erwähnung erhalten.

Danksagung

An dieser Stelle möchte ich mich bei den beiden betreuenden Lehrern bedanken, ohne die meine Projektarbeit in dieser Form nicht möglich gewesen wäre. Ein ganzes Jahr lang standen wir in regelmäßigm Kontakt und erhielten immer wieder neue Denkanstöße.

Dabei möchte ich mich insbesondere bei Frau Mielchen-Woköck bedanken, mit der ich in den ersten Monaten in wöchentlichem Abstand E-Mails über die aktuelle Lage austauschte. Zudem erklärte sie sich mehrfach zu Beratungsgesprächen bereit, in der sie mir ihre Verbesserungsvorschläge genauestens erläuterte. Dazu gehört u.a. die Idee des von mir umgesetzten Buttons, der unerfahrene Nutzer vor Verwirrung durch komplexe Ergebnisse schützt.

Auch Herrn Lomann möchte ich meinen Dank aussprechen. Als ich zu Beginn dieses Projektkurses nahezu keine Erfahrung bezüglich Appentwicklung aufwies, konnte er Vincent und mich bestens beraten, welche Vorstellungen und Ziele im Bereich des Möglichen liegen.

Glossar

Elementare Datentypen

Bool / Boolean

Boolesche Aussagevariable (true; false)

Char

Zeichen (z.B. 'Q'; '9'; , 😊'; ...)

Double

Gleitkommazahl (z.B. -3.1415926; 14.0; 1.2; ...)

Int / Integer

Ganze Zahl (z.B. -3; 14; 0; ...)

String

Zeichenkette (z.B. „x = -0,5 ± 0,866i“; „QuadraSolve“; „😊“; ...)

Optional

Optionals besitzen wie Booleans lediglich zwei Zustände: *Set* / *not Set*. Wird einer Variable/Konstante beispielsweise ein Wert zugewiesen, bei dem sich der Compiler vorher nicht sicher sein kann, dass die Zuweisung erfolgreich funktioniert, wird entweder *not Set* zugewiesen oder gegebenenfalls *Set*, jedoch wird „darunter“ noch der entsprechende Wert gespeichert. Möchte man nun auf den unter *Set* zugewiesenen Wert zugreifen und hat vorher ausgeschlossen, dass sich der Optional im *not Set*-Zustand befinden könnte, lassen sie sich unwrappen.

Objektorientierte Programmierung (OOP)

Klasse

Definition eines Objekttyps, in dem zur Verfügung stehende Methoden und Attribute festgelegt werden → *Projektarbeit*

Objekt

Instanz bzw. Exemplar einer Klasse (oder eines Datentyps) → *meine Projektarbeit*
„QuadraSolve - Projektarbeit von Marvin von Hagen“

Attribut

In Klassen festgelegte Eigenschaften, deren Werte sich in verschiedenen Instanzen gleicher Klassen unterscheiden → *Seitenlänge: Int; Inhalt: String*

Methode

Funktion einer Klasse, durch die verschiedene Objekte „Arbeit verrichten“ und untereinander kommunizieren können → *Bewerten*

Parameter

Übergabewert, der Attribute bestimmter Objekte spezifiziert und von einer Methode an eine andere übergeben wird → *Note*

Beispiel zum besseren Verständnis

Wenn wir uns jetzt in einem auf objektorientierter Programmierung basierten Programm befänden, könnte es beispielsweise die Klasse „Projektarbeit“ geben. Von dieser Klasse erstelle ich nun das Objekt „QuadraSolve - Projektarbeit von Marvin von Hagen“. Dabei lege ich fest, dass die Seitenlänge meiner Instanz den Int-Wert 40 aufweist und der Inhalt als String gespeichert wird. Da in der Klasse „Projektarbeit“ die Methode „Abgabe“ definiert ist, kann ich Ihnen die Eigenschaften meiner Projektarbeit übermitteln.

Mit Ihren Methoden lesen Sie jetzt meine Projektarbeit und übermitteln mir den Rückgabeparameter „Note“, den Sie mit Ihrer in der Klasse „Lehrer“ definierten Methode „Bewerten“ anhand der von mir übermittelten Daten bilden könnten.

Sonstiges

Build

App in bestimmter Entwicklungsstufe, z.B. App-Version 1.1 Build 2, in der aber nicht weitere Informationen wie die App Store Beschreibung enthalten sind.

Xcode

Apples integrierte Entwicklungsumgebung (IDE), mit der sich Applikationen für die Betriebssysteme macOS (Mac-Computer), iOS (iPhone, iPad, iPod touch), watchOS (Watch) und tvOS (TV) in diversen Programmiersprachen entwickeln lassen.

Swift

Apples 2014 vorgestellte objektorientierte Programmiersprache, mit der sich Programme für die obigen Betriebssysteme programmieren lassen.

User Interface (UI) / Graphical User Interface (GUI)

Sichtbare Oberfläche, mit der der Endnutzer interagiert.

Compiler

Übersetzt den Programmcode aus für Menschen direkt verständlichen Sprachen wie Swift in die Programmiersprache des jeweiligen Computers.

Dateien

Main.storyboard

Primäre .storyboard Datei, also das UI, das bei QuadraSolve während der Nutzung zu sehen ist.

Viewcontroller.swift

Mit dem Main.storyboard verknüpfte Codedatei.

Assets.xcassets

Hilfsdatei mit Icons etc..

QuadraSolve.xcodeproj

Die Datei, in der das gesamte QuadraSolve Xcodeprojekt mit Main.storyboard, Assets.xcassets, info.plist, Viewcontroller.swift, etc. enthalten ist.

Anhang

Screenshot unserer Twitterseite

COD3LTA
@COD3LTA

🔗 [itunes.apple.com/app/apple-stor...](https://itunes.apple.com/app/apple-store/)

4 FOLGE ICH 2 FOLLOWER

TWEETS TWEETS & ANTWORTEN MEDIEN GEFÄLLT MIR

COD3LTA @COD3LTA · 30.01.17

➡️ M. von Hagen / V. von Oepen - QuadraSolve - Solve quadratic equations - [itunes.apple.com/app/apple-stor...](https://itunes.apple.com/app/apple-store/)
#iTunes #QuadraSolve

◀️ ↕ ❤️ 2 ⌂

Alle in dieser Projektarbeit befindlichen Abbildungen sind selbstgeschossene Screenshots.

Versicherung über die selbstständige Anfertigung

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst, keine anderen als die angegebenen Hilfsmittel benutzt und die Stellen, die im Wortlaut oder im Inhalt aus anderen Werken entnommen wurden, mit genauer Quellenangabe kenntlich gemacht habe. Ungekennzeichnetes Übernehmen fremder Texte oder Informationen kann dazu führen, dass die Arbeit mit „Ungenügend“ bewertet wird.

Velbert, den 11.06.2017

Literaturverzeichnis

¹ Christian Bleske „iOS-Apps programmieren mit Swift“, ISBN: 978-3864904387

² <https://developer.apple.com/ios/human-interface-guidelines/visual-design/color/>

³ <http://www.stanford.edu/class/cs193p/cgi-bin/drupal/>

<https://developer.apple.com/videos/play/wwdc2016/222/>

<https://itunes.apple.com/de/course/developing-ios-10-apps-with-swift/id1198467120>

⁴ <http://stackoverflow.com/questions/24200888/any-way-to-replace-characters-on-swift-string>

⁵ <https://developer.apple.com/programs/>

⁶ <https://itunesconnect.apple.com/>

⁷ <https://twitter.com/QuadraSolve>

⁸ <https://developer.apple.com/testflight/>

⁹ <http://thomasfinch.me/iosicon/>

¹⁰ <https://itunes.apple.com/app/apple-store/id1196212823?pt=118542579&ct=vHagen&mt=8kiii>

¹¹ <http://stackoverflow.com/questions/38862208/preferredstatusbarstyle-removed-in-swift-3>

¹² <https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/>

¹³ <https://www.jugend-forscht.de/teilnahme/fachgebiete/mathematikinformatik.html>

¹⁴ <https://code.design/camps/>

¹⁵ <https://jugendhackt.org/events/west/>