HOCHSCHULE
ESSLINGEN

Bachelor-Thesis

Flutter vs. Kotlin-Multiplatform: Untersuchung der Eignung der Plattformen für den Relaunch der mQuest App

im Studiengang
Softwaretechnik und Medieninformatik
am 29. Januar 2024

vorgelegt von
Ayleen Weiß

Erstprüfer: Prof. Dr.-Ing. Andreas Rößler
Zweitprüfer: Prof. Dr.-Ing. Reiner Marchthaler

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 29. Januar 2024 _____
Unterschrift

Nutzung von KI-Hilfsmitteln

Bei der Erstellung dieser Arbeit wurden Hilfsmittel wie ChatGPT [1] und DeepL [2] verwendet. Dabei wurden keine kompletten Texte durch KI-Hilfsmittel erstellt. Die Verwendung beschränkt sich auf folgende Anwendungsfälle:

- DeepL Translator wurde zur Übersetzung von Texten, bzw. Textabschnitten genutzt.
- DeepL Write wurde gelegentlich genutzt, um einzelne Textabschnitte bzw. Formulierungen zu verbessern.
- ChatGPT wurde bei der Literaturrecherche zur Zusammenfassung von Texten eingesetzt, um schneller erkennen zu können, ob sich eine intensivere Beschäftigung mit einem Text lohnt.
- ChatGPT wurde zur schnelleren Erstellung von Codeabschnitten eingesetzt.

Zusammenfassung

Der Einsatz plattformübergreifender Entwicklungs-Frameworks birgt zeitliche und finanzielle Potenziale, wenn Apps für mehr als eine Zielplattform entwickelt werden sollen.

Um ein Unternehmen bei der Auswahl eines geeigneten Frameworks zu unterstützen, wird ein Entscheidungsinstrument in Form einer Matrix zur Bewertung dieser Technologien entwickelt. Die im Folgenden dargestellten Methoden sollen die spezifischen Anforderungen und Erfahrungen des jeweiligen Unternehmens berücksichtigen.

Die Arbeit beinhaltet eine umfassende Literaturrecherche zum Thema Cross-Plattform-Entwicklung, um die Entwicklung und den aktuellen Stand in diesem Bereich zu verstehen. Außerdem werden die technischen Rahmenbedingungen untersucht. Aufgrund einer Vorauswahl durch das betreuende Unternehmen liegt der Fokus auf Flutter und Kotlin Multiplatform. Ergänzt wird dies durch einen Überblick über weitere aktuell relevante Frameworks sowie einen Blick auf Transpiler-Technologien, die zur Konvertierung von Swift- in Kotlin-Code und vice versa eingesetzt werden.

Die Untersuchung wird durch die Durchführung von Experteninterviews mit wichtigen Stakeholdern des beteiligten Unternehmens vertieft. Diese Interviews werden mittels qualitativer Inhaltsanalyse ausgewertet und liefern wertvolle Einblicke in die praktischen Herausforderungen und Präferenzen des Unternehmens.

Zusätzlich wird ein Fragebogen an die Entwickler des Unternehmens verteilt, um quantitative Daten über allgemeine Entwicklungspraktiken und spezifische Erfahrungen im Zusammenhang mit der Entwicklung der bestehenden App zu sammeln.

Aus diesen Ergebnissen werden die Kriterien und deren Gewichtung abgeleitet, die in die Entscheidungsmatrix einfließen.

Der praktische Teil der Abschlussarbeit umfasst die Entwicklung von Prototypen mit Flutter und Kotlin Multiplatform. In der Entscheidungsmatrix werden nun die Kriterien für jedes Framework auf Basis der Erfahrungen aus der Prototyp-Entwicklungsphase bewertet.

Die Arbeit schließt mit einer Analyse der Ergebnisse, die eine Empfehlung für die Auswahl des Frameworks und Strategien für die Migration zur zukünftigen App enthält.

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während meines Studiums und der Anfertigung meiner Bachelorarbeit unterstützt haben.

Zunächst gebührt mein Dank Herrn Prof. Dr.-Ing. Andreas Rößler, der mich als Erstprüfer an der Hochschule Esslingen während meiner Bachelorarbeit betreut und unterstützt hat. Für die hilfreichen Anregungen und die ermutigende Rückmeldung während der Erstellung dieser Arbeit bin ich sehr dankbar.

Ebenso danke ich Herrn Prof. Dr.-Ing. Reiner Marchthaler, der die Begutachtung meiner Arbeit als Zweitprüfer übernommen hat.

Ein besonderer Dank gilt auch Herrn Dominic Schmidt, der mir als Firmenbetreuer seitens der cluetec GmbH während des gesamten Forschungs- und Schreibprozesses mit wertvollen Hinweisen zuverlässig zur Seite stand. Seine fachliche Expertise und konstruktive Kritik waren für die Erstellung dieser Arbeit sehr wertvoll.

Darüber hinaus möchte ich mich bei dem gesamten Team der cluetec bedanken, das mir während dieser Zeit sein Wissen und seine Zeit zur Verfügung gestellt hat.

Ebenfalls danken möchte ich meinen Eltern und meiner Lebensgefährtin für ihre unermüdliche Unterstützung, Ermutigung und Liebe während meiner gesamten akademischen Laufbahn. Ohne sie wäre dieser Weg sehr viel schwieriger gewesen.

Ein großes Dankeschön geht auch an meine Freunde, die immer ein offenes Ohr für mich hatten und mir in stressigen Zeiten zur Seite standen. Ebenso an meine Kommilitonen, die mich auf meinem Weg begleitet und unterstützt haben. Ihre Freundschaft und moralische Unterstützung waren eine unschätzbare Hilfe.

Inhaltsverzeichnis

Abbildungsverzeichnis iii

Tabellenverzeichnis iv

Codeverzeichnis v

Abkürzungsverzeichnis vi

Glossar vii

1 Einleitung 1
- 1.1 cluetec GmbH und die mQuest® Produktfamilie . . . 1
- 1.2 Motivation . . . 1
- 1.3 Wissenschaftliche Frage . . . 2
- 1.4 Vorgehensweise . . . 2

2 Grundlagen 4
- 2.1 Evolution der mobilen Computerplattformen . . . 4
- 2.2 Die Herausforderung: Entwicklung für mehrere Plattformen . . . 5
- 2.3 Die ersten Cross-Plattform Frameworks . . . 6
- 2.4 Die aktuelle Situation bei Cross-Plattform Frameworks . . . 6

3 Stand der Technik 9
- 3.1 Technische Konzepte von Cross-Plattform Frameworks . . . 9
  - 3.1.1 Deklarative UIs . . . 9
  - 3.1.2 Kotlin Multiplatform . . . 10
  - 3.1.3 Flutter . . . 12
  - 3.1.4 React Native . . . 14
  - 3.1.5 .NET MAUI (ehemals Xamarin) . . . 15
- 3.2 Code-Transpiler . . . 16
  - 3.2.1 Transpiler von Kotlin nach Swift und vice versa . . . 16
  - 3.2.2 Transpiler bei cluetec . . . 19
- 3.3 Literaturrecherche . . . 20
  - 3.3.1 Arbeiten zu Cross-Plattform-Entwicklung . . . 21

3.3.2 Performancebetrachtungen . . . 23

4 Methoden **25**
4.1 Befragung von Experten . . . 25
4.2 Prototypen . . . 27

5 Ergebnisse **29**
5.1 Experteninterviews . . . 29
5.1.1 mQuest-App . . . 30
5.1.2 Organisatorische Themen . . . 34
5.1.3 Kernaspekte der Softwareentwicklung . . . 37
5.1.4 Frameworkbezogene Themen . . . 41
5.2 Auswertung der Fragebögen . . . 43
5.3 Evaluation der Prototypen-Entwicklung . . . 55
5.3.1 Programmiersprache . . . 55
5.3.2 Dependency-Management . . . 56
5.3.3 Tooling . . . 56
5.3.4 Performance, Effizienz . . . 57
5.3.5 Reifegrad des Frameworks . . . 58
5.3.6 Dokumentation . . . 59
5.3.7 Modularität . . . 60
5.3.8 Code-Wiederverwendbarkeit . . . 61
5.3.9 Weitere Plattformen . . . 62
5.3.10 Plattformspezifische UI . . . 63
5.3.11 Plattformspezifischen Code integrieren . . . 65
5.3.12 Ressourcen, Lokalisierung . . . 66
5.3.13 Testing . . . 67
5.3.14 Debugging . . . 68
5.3.15 Resultat . . . 69
5.3.16 Deployment . . . 69

6 Schlussbetrachtung **70**
6.1 Entscheidungsmatrix . . . 70
6.2 Fazit . . . 72
6.3 Ausblick . . . 72

Literaturverzeichnis **74**

Abbildungsverzeichnis

Abbildung 2.1 Weltweite Smartphone Verkaufszahlen 5
Abbildung 2.2 Stack Overflow Trends 7
Abbildung 2.3 Nutzung von Cross-Plattform-Frameworks für Mobile Apps 7
Abbildung 2.4 Chronologischer Überblick Cross-Plattform Frameworks 8

Abbildung 4.1 Wireframe der Prototypen-App 28

Abbildung 5.1 Mindmap zu den mQuest-App Themen 30
Abbildung 5.2 Mindmap zu organisatorischen Themen 34
Abbildung 5.3 Mindmap zu Kernaspekten der Softwareentwicklung 37
Abbildung 5.4 Mindmap zu frameworkbezogenen Themen 41
Abbildung 5.5 Frage 1 - Probleme mit der bisherigen App-Architektur 43
Abbildung 5.6 Frage 2 - Bevorzugtes Lizenzmodell des Entwicklungs-Frameworks 44
Abbildung 5.7 Frage 3 - Bedeutung der Community 44
Abbildung 5.8 Frage 4 - UI-Konzept der neuen App 45
Abbildung 5.9 Frage 5 - Bedeutung der Usability 45
Abbildung 5.10 Frage 6 - Einheitliche User Experience 46
Abbildung 5.11 Frage 7 - Performance der UI 46
Abbildung 5.12 Frage 8 - Bewertung plattformübergreifender Frameworks 47
Abbildung 5.13 Frage 9 - Erfahrungen mit Cross-Plattform Frameworks 48
Abbildung 5.14 Frage 10 - Bevorzugte Programmiersprachenks 49
Abbildung 5.15 Frage 11 - Anforderungen an eine Programmiersprache 50
Abbildung 5.16 Frage 12 - Transpiler als Alternative 51
Abbildung 5.17 Frage 13 - Zukünftige Teststrategie 52
Abbildung 5.18 Frage 14 - Neue Technologien 52
Abbildung 5.19 Frage 15 - Bevorzugte Entwicklungsumgebung 53
Abbildung 5.20 Frage 16 - Feature Parity 53
Abbildung 5.21 Frage 17 - IT-Security 54
Abbildung 5.22 Frage 18 - Barrierefreiheit 54
Abbildung 5.23 Mono-Repository 60
Abbildung 5.24 Multi-Repository 61
Abbildung 5.25 Selbsterstelltes BottomBar-Widget für iOS 64

Tabellenverzeichnis

Tabelle 3.1 Liste verwandter Arbeiten 23
Tabelle 3.2 Liste der Arbeiten mit Performance-Untersuchungen 24

Tabelle 6.1 Entscheidungsmatrix 70

Codeverzeichnis

3.1 Konvertierungslogik in Swift geschrieben 17
3.2 Konvertierungslogik zu Kotlin transpiliert (SequalsK) 18
3.3 Konvertierungslogik zu Kotlin transpiliert (ChatGPT) 18

5.1 Aufruf plattformspezifischer UI-Widgets 63
5.2 Flutter: Custom Widget 64

Abkürzungsverzeichnis

API	Application Programming Interface
CLI	Command Line Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
JS	JavaScript
JVM	Java Virtual Machine
MSR	Mining Software Repository
MVP	Minimal Viable Product
PDA	Personal Digital Assistant
PWA	Progressive Web Apps
SDK	Software Development Kit
SPA	Single Page Applications
UI	User Interface

Glossar

Engine Zentrales Software-Modul oder System, das eine spezielle Funktion oder Aufgabe ausführt und häufig als Grundlage für den Aufbau weiterer Funktionen oder Systeme dient.

Features Funktionen, die ein aus Hardware und Software bestehendes System bietet.

Lexer Ein Lexer, Teil eines Compilers, ist ein Tool, das den Quellcode in Tokens (kleinste Syntaxeinheiten wie Schlüsselwörter, Operatoren) zerlegt, um die Analyse und Übersetzung des Codes zu erleichtern.

LLVM LLVM (Ursprünglich "Low Level Virtual Machine") ist eine Sammlung modularer Compiler- und Toolchain-Technologien, die eine Vielzahl von Programmiersprachen und Architekturen unterstützt. Die Entwicklung begann im Jahr 2000 an der University of Illinois und hat inzwischen weite Verbreitung in Industrie und Wissenschaft gefunden.

MVU MVU (Model-View-Update) ist ein Architekturmuster für UIs. Im Model ist der State der Anwendung hinterlegt. Die Ansicht (View) stellt die UI basierend auf dem Model dar. Die Update-Funktion aktualisiert den State auf der Basis von Nachrichten, die durch die Interaktion des Benutzers mit der UI ausgelöst werden.

Parser Ein Parser ist ein wesentlicher Bestandteil eines Compilers, der die durch den Lexer erzeugten Tokens analysiert und deren syntaktische Struktur überprüft. Er stellt sicher, dass die Token-Kombinationen den Regeln der Programmiersprache entsprechen und bildet daraus einen Syntaxbaum, der für die weitere Verarbeitung des Codes im Compiler verwendet wird.

Stable Build Bezeichnet eine ausgiebig getestete Version einer Software, die als zuverlässig und frei von kritischen Fehlern gilt, und die für den allgemeinen Gebrauch oder die Produktion empfohlen wird (im Gegensatz zu Beta-Versionen oder Nightly Builds).

State Der Zustand eines Software-Systems, bezieht sich auf die momentan gespeicherten Informationen oder Daten eines Systems, die dessen aktuelles Verhalten oder Ausgabe charakterisieren.

Target Bezeichnet die spezifische Plattform oder Umgebung, für die eine Software kompiliert wird, wie z.B. ein bestimmtes Betriebssystem oder eine Hardware-Architektur.

Thread Kleinste Einheit eines Prozesses, die unabhängig ausgeführt werden kann. Ermöglicht es, mehrere Aufgaben gleichzeitig innerhalb eines Programms auszuführen.

Transpiler Ein Transpiler oder Quellcode-zu-Quellcode-Compiler ist ein Werkzeug, das Code einer Programmiersprache in äquivalenten Code einer anderen Sprache umwandelt.

Wasm WebAssembly ist ein kompaktes binäres Befehlsformat, das in modernen Browsern ausführbar ist und mit Unterstützung des W3C-Konsortiums als Standard vorangetrieben wird [3].

1 Einleitung

Angesichts der Vielzahl von heute verfügbaren Entwicklungs-Frameworks stellt sich die Frage, wie man das Richtige auswählt. Tauchen Sie mit dieser Arbeit in die spannende Welt der Cross-Plattform-Entwicklung ein, um mehr zu erfahren.

Lernen Sie zunächst die betreuende Firma und das Softwareprodukt kennen, bevor wir einen Blick auf die wissenschaftliche Frage stellen, die mit dieser Arbeit beantwortet werden soll.

1.1 cluetec GmbH und die mQuest® Produktfamilie

Die *cluetec GmbH* (im Folgenden nur *cluetec* genannt) wurde im Jahr 2000 von den drei geschäftsführenden Gesellschaftern Tobias Craffonara, Matthias Helferich und Dr. Thomas Rieger gegründet. Sie ist ein in Karlsruhe ansässiger Spezialist für Beratung, Konzeption und Entwicklung von Enterpriseanwendungen, Portalen und mobilen Apps. Im Fokus steht die Erfassung, Verwaltung, Verarbeitung, Präsentation und Verteilung von Daten, on- und offline, lokal und mobil [4]. Einen großen Teil davon deckt die *mQuest®* Produktfamilie ab, eine mobile Befragungs- und Erhebungssoftware, bestehend aus der PC-Anwendung *QuestEditor*, den Apps für die mobilen Betriebssysteme iOS und Android, einem Backend zur Auswertung und Persistierung der Daten, sowie einem Kundenportal mit umfangreichen Visualisierungsmöglichkeiten. Im Jahr 2017 kam die Komponente *mQuest® Audit* dazu, welche das System um ein Auditierungsplattform erweitert, die eine komplette Digitalisierung von Auditierungen erlaubt.

1.2 Motivation

Mobile Anwendungen gehören heute nicht nur zum Standardangebot von Softwareunternehmen, vielmehr hat sich seit Beginn der 2010er Jahre eine *Mobile First* genannte Strategie weit verbreitet. Der Begriff Mobile First wurde 2009 von Luke Wroblewski [5] geprägt und 2010 von Google-CEO Eric Schmidt auf der Mobilfunkmesse in Barcelona als neues Unternehmensprinzip ausgerufen [6].

Bei der cluetec reicht die Erfahrung mit mobilen Anwendungen zurück bis ins Jahr 2001, als die ersten Apps für *Palm OS®* und *Windows Mobile®* entwickelt wurden. Die hierfür in Java geschriebene Anwendungslogik, intern *mobilecore* genannt, wurde stets weiterentwickelt und bildet noch immer das Rückgrat für die heutigen Apps. 2010 wurde auf Basis dieses mobilecore die erste Android App entwickelt. Da zur Entwicklung von Android-Anwendungen zu dieser Zeit ebenfalls Java verwendet wurde, konnte mit überschaubarem Aufwand einerseits eine Benutzerschnittstelle (im Folgenden immer **UI** genannt), andererseits ein Modul zur Synchronisation der Daten mit dem Backend und ein Persistence-Layer an den mobilecore angedockt werden. 2012 wurde eine iOS-App entwickelt und wie in Kapitel 3.2.2 beschrieben, konnte der mobilecore auch hierfür wiederverwendet werden. Es ist jedoch unklar, wie lange dieses Konzept in dieser Weise weitergeführt werden kann. Daher sollen die Möglichkeiten eines Relaunches auf Basis eines plattformübergreifenden Entwicklungs-Frameworks geprüft werden.

1.3 Wissenschaftliche Frage

Die vorliegende Arbeit soll die Fragestellung klären, wie sich die Eignung eines Cross-Plattform-Entwicklungs-Frameworks (im folgenden stets zu Cross-Plattform-Frameworks verkürzt) für die Neuentwicklung[1] einer bestehenden Anwendung struktuiert evaluieren lässt. Der Fokus liegt dabei auf Anwendungen, die nicht dem Spiele-Genre zuzuordnen sind, da hier ganz eigene Frameworks zum Einsatz kommen.

Das Ergebnis soll eine Entscheidungsmatrix für den allgemeinen Einsatz sein. Daraus wird dann ein Vorschlag für den konkreten Fall „mQuest“ abgeleitet.

1.4 Vorgehensweise

Im den nächsten Kapiteln werden zunächst die Grundlagen zum Thema Cross-Plattform-Entwicklung erläutert. Es wird ein Blick auf die Historie und den aktuellen Stand der Technik geworfen. Die gegenwärtig relevanten Frameworks und Technologien werden vorgestellt und die dahinter stehenden technischen Konzepte erläutert. Ein Blick auf die bisher veröffentlichten Arbeiten zum Thema Cross-Plattform-Entwicklung rundet das Verständnis des Themas ab.

1 Es ist im Einzelfall zu definieren, ob es sich um eine komplette Neuentwicklung handeln soll, oder ob bestehende Codefragmente übernommen werden. Nicht Gegenstand der Betrachtung ist die Frage, ob das bestehende UI-Design übernommen werden soll oder ob im Zuge der Neuentwicklung auch ein neues Bedienkonzept implementiert wird.

Die Methoden zur Beantwortung der wissenschaftlichen Fragestellung werden im darauf folgenden Teil behandelt. Dazu zählen eine die Anforderungsanalyse auf Basis von Expertenbefragungen und die prototypische Implementierung der wichtigsten **Features** in Testapplikationen, die mit ausgewählten Frameworks erfolgen wird.

Den Abschluss bildet die Auswertung der Untersuchungsergebnisse und eine daraus abgeleitete Entscheidungsmatrix sowie eine konkrete Empfehlung für die cluetec hinsichtlich der Auswahl eines Frameworks.

2 Grundlagen

In diesem Kapitel werden die Grundlagen und die historische Entwicklung von mobilen Computerplattformen und von Cross-Plattform-Entwicklungs-Frameworks analysiert. Dann wird die heutige Situation betrachtet. Dies ist wichtig für das Verständnis der weiteren Ausführungen zum Thema Cross-Plattform-Entwicklung, welche sich in Kapitel 3 finden.

2.1 Evolution der mobilen Computerplattformen

Im Hinblick auf mobile Computerplattformen beleuchtet dieser Abschnitt die Entwicklung der letzten zwei Jahrzehnte, sowohl in Bezug auf Betriebssysteme als auch auf Cross-Plattform-Frameworks.

Abgesehen von den ersten portablen Computern, die bereits in den 1980er Jahren auf den Markt kamen, aber mit einem Gewicht von teilweise mehreren Kilogramm noch recht mühsam zu transportieren waren, läuteten die späten 1990er und frühen 2000er Jahre den Beginn mobiler Computerplattformen ein, welche bequem in der Hand zu halten sind. Dazu gehören Handheld-PCs, **PDA**s und Smartphones. Zu den anfangs hierfür eingesetzten Betriebssystemen zählen PalmOS (1996 [7]), Symbian (1998 [8]) und die frühen mobilen Windows Plattformen (1996 als Windows CE gestartet, bis 2000 zu Pocket PC weiterentwickelt und 2003 zu Windows Mobile umbenannt [9]), wie auch BlackBerry OS (1999 [10]).

2007 betrat Apple den Markt mit iOS (zunächst als iPhone OS gestartet [11]), welches auf dem Kernel des Desktop-Betriebssystems macOS aufsetzt. 2008 erschien das erste Smartphone mit dem Betriebssystem Android, ein auf dem Linux Kernel basierendes System als Open-Source Software, dessen Entwicklung von der *Open Handset Alliance*, und in diesem Rahmen vor allem durch Google vorangetrieben wird [12]. Microsoft unternahm 2010 mit Windows Phone den Versuch, sich mit einer neuentwickelten Plattform Marktanteile im umkämpften Smartphone-Markt zu sichern [13]. Allerdings stieg die Dominanz von Android in den darauffolgenden Jahren so stark an, dass sich daneben nur noch Apples iOS Plattform halten konnte (siehe Abbildung 2.1).

Diese Auflistung erhebt nicht den Anspruch auf Vollständigkeit, versucht jedoch die wichtigesten mobilen Betriebssysteme zu nennen.

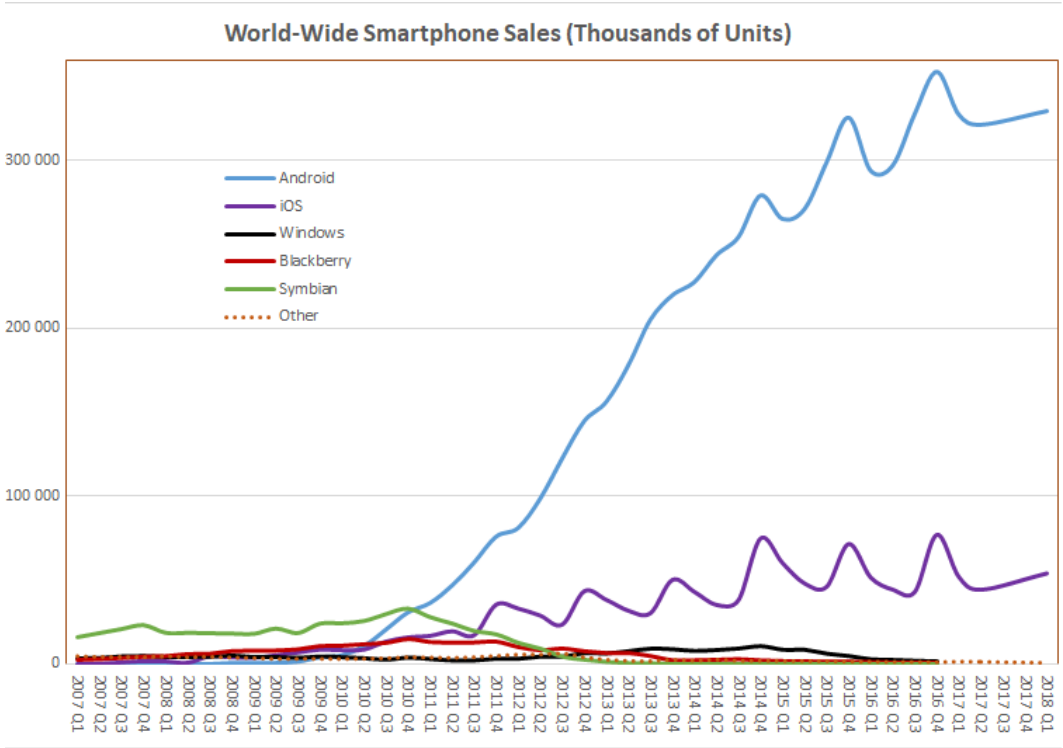

Abbildung 2.1: Weltweite Smartphone Verkaufszahlen, [14]

Anmerkung: In der Legende sind Betriebssysteme angegeben, die Zahlen beziehen sich auf die Anzahl der verkauften Geräte mit diesen Betriebssystemen.

2.2 Die Herausforderung: Entwicklung für mehrere Plattformen

Im Laufe dieser Entwicklungen gab es also zu jeder Zeit mindestens zwei wichtige mobile Betriebssysteme am Markt. Für Firmen und Software-Entwickler ergab sich daraus ein Problem: Die Erstellung separater Anwendungen für mehrere Betriebssystem-Plattformen führt unweigerlich zu einer Verschwendung von Entwicklungszeit und -aufwand sowie zu

erhöhten Wartungskosten [15]. Abhilfe versprechen Cross-Plattform-Frameworks, die es Entwicklern ermöglichen, Code einmal zu schreiben und ihn für mehrere Plattformen bereitzustellen.

2.3 Die ersten Cross-Plattform Frameworks

Xanthopoulos und Xinogalos [15] beschreiben in ihrer Arbeit frühe Cross-Plattform Entwicklungsansätze und erwähnen dabei folgende Konzepte:

- Web Apps (Anwendungen die in einem Browserfenster laufen und dynamisch aus dem Web geladen werden)
- Hybrid Apps (Web Apps in Containern, die eine Distribution über die App Stores erlauben)
- Interpretierte Apps (Code wird auf der Zielplattform durch einen Interpreter verarbeitet)
- Generierte Apps (Für jede der Zielplattformen wird nativer Code erzeugt)

Zu diesen frühen Cross-Plattform-Frameworks gehören Tools wie PhoneGap (2009) und Ionic (2013), die auf HyperText Markup Language (**HTML**) und Cascading Style Sheets (**CSS**) für das UI-Rendering setzen und JavaScript (**JS**) für die Programmierung der Logik verwenden. Beide erzeugen hybride Apps.

Ein Beispiel für das Konzept der interpretierten App ist Titanium SDK (2010). Hier wird die gesamte App in JavaScript geschrieben und auf dem Zielsystem von einer JavaScript Engine interpretiert [16].

Als frühen Vertreter der generierten Apps gab es mit Mono (2004) ein Open Source Projekt, welches als Basis Microsofts .NET Framework nutzt. Hieraus entwickelte sich Xamarin (2012), welches nun in .NET MAUI (2022) einen Nachfolger bekommen hat.

2.4 Die aktuelle Situation bei Cross-Plattform Frameworks

Zu den derzeit prominentesten Frameworks gehören Flutter (2017) und React Native (2015), wie zum einen die statistischen Auswertungen von Stack Overflow (siehe Abbildung 2.2) und JetBrains (siehe Abbildung 2.3) zeigen, zum anderen haben sich in den letzten Jahren mehrere Publikationen gerade mit diesen beiden Frameworks beschäftigt (siehe z.B. Wu, 2018 [17]; Biørn-Hansen et al., 2020 [18]; Armagan, 2020 [19]; Nawrocki et al., 2021 [20]; Mota, 2021 [21]; Oliveira, 2023 [22] und weitere).

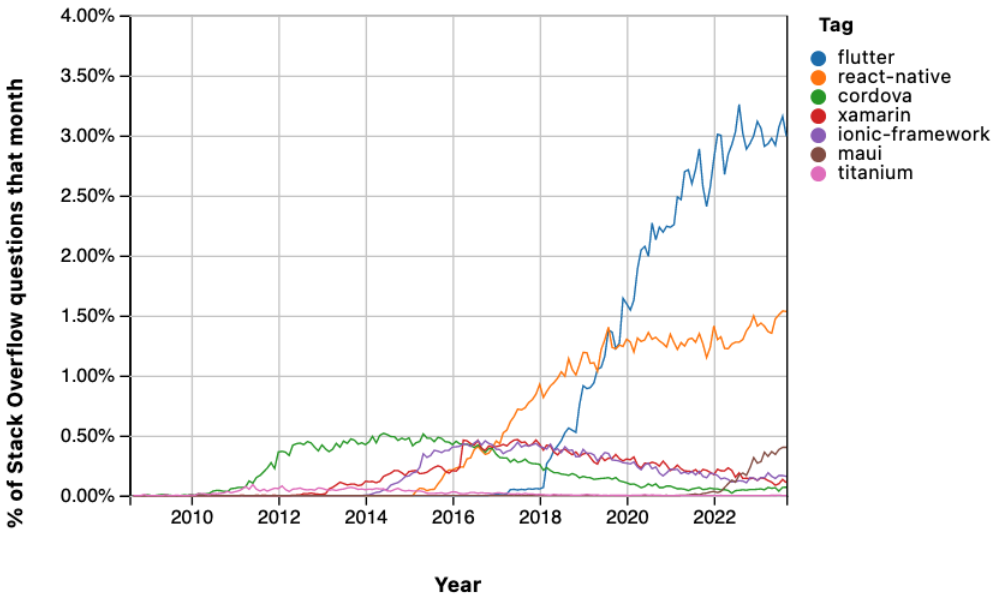

Abbildung 2.2: Stack Overflow Trends, [23]

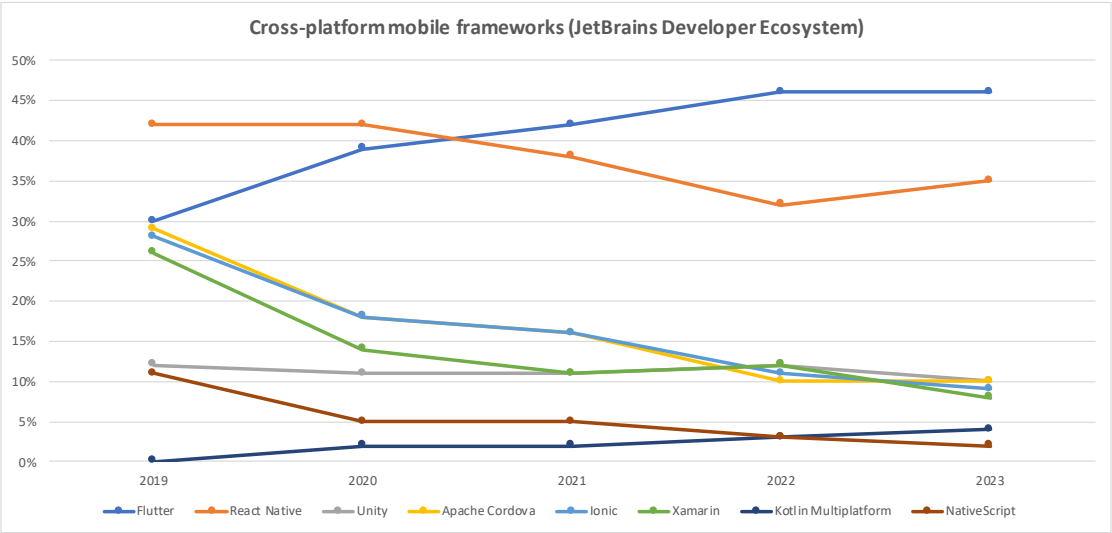

Abbildung 2.3: Nutzung von Cross-Plattform-Frameworks für Mobile Apps von 2019 bis 2023 (Grafik erstellt aus statistischen Daten der Fa. JetBrains [24])

2017 begann die Entwicklung an Kotlin Multiplatform, ein Software Development Kit der Firma JetBrains. In der Literatur ist Kotlin Multiplatform noch wenig vertreten, bisher haben sich vor allem Abschlussarbeiten damit beschäftigt, siehe Fayzullaev, 2018 [25], Evert, 2023 [26] und Skantz, 2023 [27].

Fayzullaev erwähnt in seiner Thesis ein weiteres Framework namens Multi-OS Engine, dessen Status allerdings etwas unklar ist [25]. Auch wenn es nicht offiziell abgekündigt wurde, so sind Updates eher selten, im September 2022 wurde angekündigt, das Projekt wiederzubeleben, seither ist aber nichts geschehen [28]. Auf Stackoverflow wurde die letzte Frage zu diesem Framework im Jahr 2018 gestellt. Daher findet dies hier keine weitere Beachtung.

RAD Studio [29] ist eine Entwicklungsumgebung für die Sprache Delphi, die mit Erscheinen der Version XE4 im Jahr 2013 [30] ebenfalls die Entwicklung von Anwendungen für Android und iOS unterstützt. Betrachtet man jedoch die Umfragen zu den derzeit beliebtesten Programmiersprachen auf Stackoverflow [31] und IEEE Spectrum [32], so muss man zu dem Schluss kommen, dass Delphi eher einen Exotenstatus einnimmt. Daher wird auch dies hier nicht näher betrachtet werden.

Abbildung 2.4 zeigt eine Übersicht über die Entwicklung einiger Cross-Plattform Frameworks, die heute noch gepflegt und weiterentwickelt werden (ohne Anspruch auf Vollständigkeit).

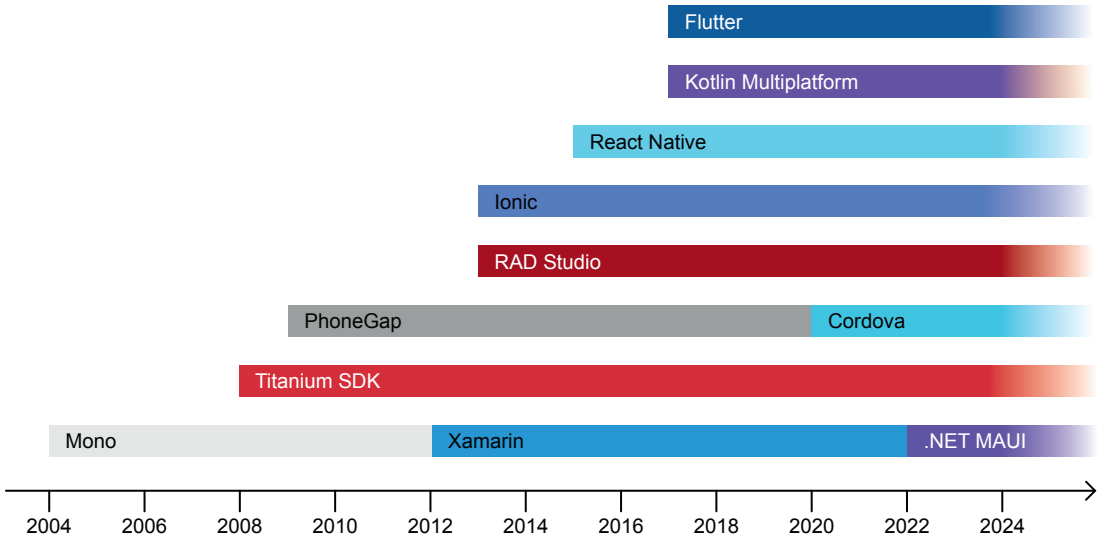

Abbildung 2.4: Chronologischer Überblick Cross-Plattform Frameworks

3 Stand der Technik

Die Softwareentwicklung ist einem ständigen Wandel unterworfen und die Paradigmen ändern sich in regelmäßigen Abständen. Die Entscheidung für eine Plattform erfordert ein gutes Verständnis des aktuellen Stands der Technik. Um ein fundiertes Wissen über die technischen Konzepte hinter den Cross-Plattform Frameworks zu erlangen, werden die derzeit gefragtesten und vielversprechendsten Frameworks in ihrer Funktionsweise untersucht. Die Literaturrecherche gibt einen Überblick über die bisherigen Arbeiten auf diesem Gebiet und zeigt auf, welche Schwerpunkte dort gesetzt wurden.

3.1 Technische Konzepte von Cross-Plattform Frameworks

Wie in Kapitel 2.3 beschrieben, gibt es verschiedene Ansätze für die plattformübergreifende Entwicklung. Da nur die generierten Apps eine hohe Performance aufweisen, einen nativen Eindruck bei der Bedienung der UI vermitteln sowie den vollen Zugriff auf die Hardware der Geräte und den vollen Datenzugriff ermöglichen [15, Tabelle 2], wird im Folgenden vor allem auf diesen Ansatz näher eingegangen.

Zunächst wird jedoch ein seit einigen Jahren stark an Bedeutung gewinnendes Konzept der Programmierung von UIs erläutert.

3.1.1 Deklarative UIs

In den letzten Jahren ist ein Wandel in der Programmierung mobiler Apps zu beobachten. Es gibt einen Trend zur Nutzung neuer, moderner und ausdrucksstarker Programmiersprachen (zum Beispiel Swift und Kotlin), die die bisherigen, aus den 1980er und 1990er Jahren stammenden Sprachen ablösen. Darüber hinaus ist ein Übergang von imperativen zu deklarativen UI-Frameworks wie SwiftUI und Jetpack Compose zu beobachten. Dies verspricht eine Steigerung der Produktivität, da Apps mit weniger Code und damit schneller programmiert werden können [33].

Bei der imperativen UI-Programmierung legen die Entwickler nicht nur fest, wie die UI aussehen soll, sondern auch, wie dieses Aussehen erreicht wird. Dies beinhaltet oft auch die manuelle Aktualisierung der UI und die Handhabung von Änderungen des

State[2]. Deklarative Programmierung ist ein Programmierparadigma, welches sich auf die Beschreibung dessen konzentriert, was erreicht werden soll, und es dem System überlässt, die Details der Ausführung zu bestimmen [34].

UI als Code

Eine deklarative Programmierung von UIs ermöglicht es Entwicklern also, das Erscheinungsbild und das Verhalten von UIs mithilfe von Codekonstrukten zu beschreiben, der Schwerpunkt liegt dabei eher auf dem „Was“ als auf dem „Wie“ [34]. Zentraler Bestandteil dieses Ansatzes ist das Konzept einer einzigen Quelle der Wahrheit [35] für den State der UI. Ereignisse, wie z. B. Benutzerinteraktionen, können diesen State verändern und das Framework veranlassen, die UI automatisch zu aktualisieren. Das Framework selbst verwaltet diese Aktualisierungen und optimiert das Rendering, um Änderungen des State effizient wiederzugeben. Ein Beispiel hierfür ist SwiftUI [36], im Folgenden wird aufgezeigt, dass das Paradigma der deklarative UI-Programmierung auch in den betrachteten Cross-Plattform Frameworks genutzt wird, oder zumindest als Option zur Verfügung steht.

3.1.2 Kotlin Multiplatform

Das Framework Kotlin Multiplatform, abgekürzt KMP ermöglicht das Teilen von Code über verschiedene Plattformen hinweg, einschließlich Desktop und Server. Mit der Entwicklung von Kotlin/**Wasm** und Compose Multiplatform (siehe Kapitel 3.1.2) strebt JetBrains an, Kotlin zur ersten Wahl für universelles Code-Sharing zu machen [37].

Die Sprache: Kotlin

Kotlin ist eine statisch typisierte Programmiersprache, die sowohl objektorientierte als auch funktionale Programmierung ermöglicht [38]. Sie wurde erstmals 2011 als neue Sprache für die Java Virtual Machine (JVM) vorgestellt und wird von der Firma JetBrains entwickelt. Die Version 1.0 erschien 2016. Nach Aussage von Dmitry Jemerov[3] verwendete JetBrains zu dieser Zeit hauptsächlich Java für die Entwicklung seiner IntelliJ-basierten IDEs, wollte aber die Produktivität durch den Wechsel zu einer ausdrucksstärkeren Sprache steigern, ohne dabei die Java-Interoperabilität oder die Kompiliergeschwindigkeit zu

2 In dieser Arbeit wird durchgängig der Begriff State anstelle von Zustand verwendet, da er in der Softwarewelt gängig ist.

3 Dmitry Jemerov war bis August 2023 Entwickler bei JetBrains, wo er unter anderem das Team leitete, welches das IntelliJ IDEA Plugin für Kotlin entwickelte [39].

beeinträchtigen [40]. Kotlin übernimmt zunehmend die Rolle von Java in der nativen Android-Entwicklung, seit Google auf der Entwicklerkonferenz I/O 2019 eine Kotlin-first Strategie für Android verkündet hat [41, Kap. 1.2.1] und [42].

Funktionsweise Kotlin Multiplatform

KMP ermöglicht es, die Anwendungslogik zwischen Android und iOS (und ggf. weiteren Plattformen) zu teilen, während die UI nativ für jede Plattform erstellt wird, um ein optimales Benutzererlebnis zu gewährleisten (hierfür wird es zukünftig mit Compose Multiplatform eine Alternative geben, siehe unten). Die gemeinsame Logik beinhaltet Kernfunktionen wie Datenverarbeitung und -validierung, Berechnungen, Kommunikation zwischen Systemen (beispielsweise HTTP und API-Calls), State-Management sowie die Implementierung von Sicherheitsmaßnahmen.

Bei einem klassischen KMP Projekt gibt es daher einen Bereich (common) für den gemeinsam genutzten Code und jeweils eigene Ordner für das Android und das iOS Projekt. Alternativ kann die Anwendungslogik als Multiplatform Bibliothek erstellt und in Android Studio und Xcode (als XCFramework) eingebunden werden.

Für Android wird der Kotlin Code in Bytecode für die Java Virtual Machine (JVM) übersetzt, für iOS wird er mit **LLVM** zu nativem Maschinencode kompiliert. Letzteres wird auch als Kotlin Native bezeichnet, damit können prinzipiell verschiedene ***Targets*** angesprochen werden [43].

Ausblick: UI mit Compose Multiplatform

Während mit den bisherigen KMP Projekten nur die Anwendungslogik geteilt werden kann, und für das UI auf den Zielplatformen jeweils nativer Code geschrieben werden muss, geht Compose Multiplatform einen Schritt weiter und erlaubt es, auch das UI mit einer gemeinsamen Codebasis zu erstellen. Dafür wird auf das in der Android-Entwicklung bereits bekannte Jetpack Compose zurückgegriffen. Compose folgt dem Prinzip der deklarativen UI Programmierung, ist zum Zeitpunkt der Abfassung dieser Arbeit für die Plattform iOS jedoch noch im Alpha Stadium [44].

3.1.3 Flutter

Flutter unterstützt neben den mobilen Plattformen Android und iOS auch Desktop, Web und bestimmte Embedded Plattformen [45].

Die Sprache: Dart

Dart ist eine multiparadigmatische Sprache mit optionaler dynamischer Typisierung, die in funktionalem Stil programmiert werden kann, aber nicht muss [46, S. 56]. Dart wurde 2011 vorgestellt und 2013 in der Version 1.0 veröffentlicht. Ursprünglich für die strukturierte Webprogrammierung gedacht [47], wird Dart heute als client-optimierte Sprache für schnelle Anwendungen auf jeder Plattform bezeichnet [48]. Dart ist vor allem durch seine Verwendung im Flutter-Framework bekannt, kann aber auch in anderen Szenarien wie CLI und Serveranwendungen eingesetzt werden [49]. Im Angular-Framework kann Dart anstelle von JavaScript zur Entwicklung von Webanwendungen verwendet werden [50].

Funktionsweise Flutter

Zentral für Flutter ist die Flutter-**Engine**. Sie stellt die Low-Level-Implementierung von Flutters Kern-**API** bereit, einschließlich Grafik-Rendering (über *Impeller* auf iOS und bald auf Android, sowie *Skia* auf anderen Plattformen[4].), Textlayout, Datei- und Netzwerk-I/O, Unterstützung für Barrierefreiheit, Plugin-Architektur sowie eine Dart-Laufzeitumgebung und Kompilierungswerkzeuge. Für jede Zielarchitektur gibt es einen plattformabhängigen *Embedder*. Entwickler interagieren mit dem *Framework*, dies beinhaltet:

- Grundfunktionen wie Animationen und Gesten
- Eine Rendering-Schicht für Layouts
- Eine Widget-Schicht für Komposition (siehe Seite 13) und reaktive Programmierung
- Material- und Cupertino-Bibliotheken für plattformspezifische Designsprachen

4 Impeller wurde entwickelt, da es mit Skia gewisse Probleme beim ersten Aufruf der App gab. Bis die Ansichten zum ersten mal vollständig gerendert waren, gab es aufgrund der Shader-Kompilierung oft ruckelige Animationen, den sog. *shader compilation jank*

Während das Kern-Framework kompakt ist, erweitern diverse Pakete aus dem Ökosystem dessen Funktionen, etwa für In-App-Zahlungen und State-Management [51].

Bei der Entwicklung mit Flutter kann *Stateful Hot Reload* genutzt werden. Wenn Änderungen am Code vorgenommen werden, sind die Auswirkungen sofort nach dem Speichern im Simulator sichtbar, ohne dass Flutter den State verwirft. Ein Beispiel wäre eine Anwendung, die eine Benutzeranmeldung erfordert. Eine Seite, die mehrere Ebenen tiefer in der Navigationshierarchie liegt, kann im laufenden Betrieb geändert und neu geladen werden, ohne dass die Anmeldedaten erneut eingegeben werden müssen. In der Flutter-Dokumentation wird auch von der Kombination „des vorherigen Zustands mit neuem Code“ gesprochen [52].

UI in Flutter - Widgets und Komposition

Flutter setzt bei der Erstellung von UIs stark auf sogenannte Widgets. Das sind kleine Bausteine, die ineinander verschachtelt werden. Wie beim Dekorator-Muster (einem der bekannten Software-Entwurfsmuster) ändert ein Widget, das ein anderes umgibt, dessen Aussehen. So kann zum Beispiel ein Text-Widget in ein Center-Widget eingebettet werden, um seine Ausrichtung zu ändern, und dieses Konstrukt kann in ein Padding-Widget eingebettet werden, um es mit einem Abstand zu versehen. Diese Erstellung größerer Konstrukte aus kleinen Bausteinen wird als Komposition bezeichnet, ein Konzept, welches aus dem Bereich der Objektorientierung bekannt ist [53, S. 98].

Flutter ist ein reaktives, deklaratives UI-Framework. Der Entwickler erstellt eine Zuordnung vom Anwendungs-State zum UI-State und das Framework aktualisiert die UI zur Laufzeit, wenn sich der Anwendungs-State ändert. Dieses Modell ist inspiriert durch Arbeiten von Meta für ihr eigenes React Framework, welches eine Neubewertung vieler traditioneller Designprinzipien beinhaltet [54].

3.1.4 React Native

React Native wurde 2013 von Facebook (heute Meta Platforms Inc.) ins Leben gerufen und zielt neben Android und iOS auf weitere Plattformen ab. 2015 wurde der erste ***Stable Build*** veröffentlicht. Mit React Native begann der Trend zu deklarativen UIs bei der Entwicklung von mobilen Apps [33].

Die Sprache: JavaScript / TypeScript

JavaScript ist eine Skriptsprache, die ursprünglich von Netscape unter dem Namen LiveScript entwickelt wurde. 1996 wurde JavaScript in Kooperation mit Sun Microsystems in der Version 1.0 veröffentlicht. Sie dient vor allem zur Erstellung interaktiver Webseiten und hat sich seitdem zu einer der dominierenden Sprachen im Web-Entwicklungsbereich entwickelt [55]. TypeScript, eingeführt von Microsoft im Jahr 2012, ist eine Obermenge von JavaScript und erweitert JavaScript unter anderem um statische Typisierung, um die Entwicklung von größeren Anwendungen zu unterstützen und Entwicklern dabei zu helfen, Fehler frühzeitig zu erkennen [56]. Neue React Native Projekte verwenden standardmäßig TypeScript, auch wenn es nach wie vor möglich ist, JavaScript einzusetzen [57]. Wenn also im Folgenden von JavaScript gesprochen wird, ist damit auch TypeScript als Möglichkeit eingeschlossen.

Funktionsweise und UI in React Native

React Native ähnelt in seiner Funktionsweise React, einer ebenfalls von Meta stammenden JavaScript Bibliothek, mit welcher UIs erstellt werden, die auf sog. *Components* basieren [58]. In React Native wird neben der Anwendungslogik auch das UI ausschließlich mit JavaScript erzeugt, HTML und CSS kommen nicht zum Einsatz. React Native Components werden zur Laufzeit zu Plattform-nativen UI Elementen (Views) gerendert. React Native ist somit den interpretierten Apps zuzuordnen.
Beim Öffnen einer App wird vom Main **Thread** ein JavaScript Thread gestartet, der das eigentliche React Native Programm ausführt. Die Anwendungslogik wird auf diesem Thread abgearbeitet. UI Komponenten werden durch die *Bridge* zu nativen Komponenten übersetzt [59]. Änderungen des State werden über die Bridge als serialisierte JSON Objekte gesendet. Updates für die UI werden gebündelt am Ende jeder Iteration des Event Loops an die native Seite geschickt. Die Bridge kann hier durchaus als Flaschenhals die Leistung negativ beeinflussen [60]. Dieser Nachteil bei der *Performance* wird in verschiedenen Publikationen thematisiert, so schreiben etwa Oliveira et al., dass React Native in fast allen getesteten Szenarien mangelhaft abschneidet [22].

3.1.5 .NET MAUI (ehemals Xamarin)

Microsoft beschreibt MAUI als Evolution von Xamarin.Forms (iOS und Android), erweitert um Desktop Szenarien für Windows (mit der WinUI 3 Bibliothek) und macOS (Mac Catalyst). Außerdem kann hiermit auch für Samsungs TizenOS entwickelt werden [61].

Die Sprache: C#

Bei .NET MAUI wird die Programmlogik in C# geschrieben, einer multiparadigmatischen, objektorientierten und statisch typisierten Programmiersprache. Die Entwicklung der Sprache begann im Jahr 2000, sie wurde etwas später von Microsoft zusammen mit dem .NET Framework vorgestellt [62].

Funktionsweise MAUI

Der Code eines solchen Projekts interagiert in erster Linie direkt mit der .NET MAUI API, die wiederum die nativen Plattform-APIs anspricht. Bei Bedarf können die Plattform-APIs jedoch auch direkt über die .NET MAUI API angesprochen werden. In diesem Fall muss für jede Zielplattform eine entsprechende Implementierung vorhanden sein. [63].

Für die Ausführung auf Android-Geräten wird der C#-Code in eine *intermediate language* (IL) kompiliert, die beim Start der Anwendung just-in-time (JIT) in nativen Assembler-Code kompiliert wird. Für iOS-Geräte wird der C#-Code vollständig AOT (ahead-of-time) in nativen ARM-Assemblercode kompiliert. [63].

UI in MAUI

Standardmäßig wird in MAUI die Markup-Sprache XAML für die Erstellung von UIs verwendet, es kann aber auch hier ein deklarativer Ansatz gewählt werden: Durch den Einsatz von C# Markup [64] zusammen mit dem aus der funktionalen Programmiersprache Elm bekannten **MVU**-Pattern [65] kann eine schnelle Aktualisierung der UI bei Änderungen des State erreicht werden [66], [67].

3.2 Code-Transpiler

Cross-Plattform-Frameworks arbeiten in der Regel mit einer Architektur, die es erlaubt, eine nicht-native Codebasis auf verschiedenen Targets auszuführen. Der Vollständigkeit halber soll hier noch ein weiteres Konzept Erwähnung finden, nämlich die Verwendung von **Transpiler**n. Dabei wird der Code für die Anwendungslogik mit dem für die Plattform empfohlenen **SDK** in der entsprechenden Sprache erstellt und anschließend in die Programmiersprache der anderen benötigten Plattformen übersetzt. Es liegt jedoch in der Natur dieser Technologie, dass eine solche Übersetzung die in den jeweiligen Sprachen verfügbaren Codekonstrukte nicht zu 100 % abdecken kann.

3.2.1 Transpiler von Kotlin nach Swift und vice versa

Schneider [68] vergleicht in ihrer Masterarbeit aus dem Jahr 2022 verschiedene Transpiler, die Kotlin nach Swift oder Swift nach Kotlin übersetzen und bezeichnet dieses Konzept als „aufstrebenden Ansatz für die plattformübergreifende Entwicklung für Android und Apple OS“ [68, S. 29]. Inzwischen sind die meisten dieser Werkzeuge nicht mehr aktuell: *Kotlift* übersetzt Kotlin zu Swift, ist aber seit März 2023 als deprecated markiert [69]. *SwiftKotlin* übersetzt Swift zu Kotlin, das Projekt wurde aber seit Mai 2020 nicht mehr aktualisiert [70]. *Gryphon* übersetzt Swift zu Kotlin [71], allerdings gab der Entwickler im April 2023 bekannt, dass er Gryphon nicht weiter pflegen kann. Gryphon ist in Swift geschrieben und muss lokal installiert werden, es unterstützt derzeit die Swift Versionen 5.2 bis 5.5, die veraltet sind. Dies bedeutet, dass Benutzer mit neueren Versionen von Xcode und Swift es nicht installieren und ausführen können [72].

Damit bleibt nur noch *SequalsK* (bidirektional), ein Projekt von Prof. Dominik Schultes, der an der Technischen Hochschule Mittelhessen lehrt. SequalsK ist als Webservice verfügbar, die letzte Aktualisierung erfolgte am 25.09.2023 [73]. Zusätzlich gibt es ein Plugin für die JetBrains IDEs IntelliJ IDEA und Android Studio [74]. Dieses greift auf denselben Server zu, welcher die eigentlichen Übersetzungen durchführt (D. Schultes, persönliche Kommunikation per E-Mail am 7.11.2023). SequalsK implementiert **Lexer** und **Parser** mit Hilfe des Parser Generator ANTLR4 [75], um aus dem eingegebenen Quelltext einen Syntaxbaum zu generieren. ANTLR benötigt dafür die Grammatik der jeweiligen Sprachen. JetBrains liefert für Kotlin eine solche Grammatik in ihrer offiziellen Dokumentation [76], während für Swift 3.0.1 eine unoffizielle Version auf GitHub zur Verfügung steht [77]. Schultes beschreibt, dass an letzterer Korrekturen vorgenommen werden mussten, bevor diese für SequalsK verwendet werden konnte [78]. Auf Rückfrage hat Prof. Schultes bestätigt, dass die aktuelle Version des Transpilers noch auf dieser Version beruht (D. Schultes, persönliche Kommunikation per E-Mail am 7.11.2023). Im

genannten GitHub Repository ist inzwischen auch eine Grammatik für Swift 5.4 zu finden.

Um den praktischen Nutzen der Transpiler beurteilen zu können, hat Schneider den Anteil der Code-Konstrukte analysiert, die von den jeweiligen Werkzeugen übersetzt werden können und kommt auf Werte von maximal 74 % der Swift-Konstrukte und maximal 78 % der Kotlin-Konstrukte [68, S. 23]. Für die nicht unterstützten Konstrukte wurde untersucht, wie häufig sie tatsächlich von Softwareentwicklern verwendet werden. Dazu wurde eine **MSR**-Studie durchgeführt, in der öffentlich zugängliche GitHub-Repositories ausgewertet wurden. Daraus wurde ein Vergleichswert für die notwendige manuelle Nacharbeit berechnet. Dieser mit W bezeichnete Vergleichswert konnte Werte zwischen 0 und 1 annehmen, wobei 1 keine manuelle Nacharbeit bedeutete. SequalsK erreichte bei der Übersetzung von Kotlin nach Swift einen Wert von 0,964, während bei der Übersetzung von Swift nach Kotlin nur ein Wert von 0,622 erreicht werden konnte. Somit wäre Kotlin der geeignete Ausgangspunkt für den Einsatz dieser Technologie [68, S. 79].

Mit dem *Swiftlin Converter* [79] steht ein weiterer Transpiler als Online-Dienst zur Verfügung. Laut der FAQ unterstützt Swiftlin derzeit Swift 5.8 und Kotlin 1.8.20. Neben der Möglichkeit, Codeblöcke direkt in Browserfenster konvertieren zu lassen, kann man auch ganze Dateien zur Konvertierung hochladen.

Die Übersetzung von Code mittels Transpilern lässt jedoch einige Aspekte unbeachtet wie der folgende kleine Test zeigt. Einfacher Swift-Code (siehe Listing 3.1) wurde mit SequalsK zu Kotlin übersetzt, siehe Listing 3.2.

```
class Converters {
    let constOunce = 31.1034768
    let sellingPriceFactor = 1.8

    func addSalesMarginToPrice(rate: Double) -> Double {
        return rate * sellingPriceFactor
    }

    func calculatePrice(rate:Double) -> Double {
        return round(((1 / rate) / constOunce) * 100) / 100.0
    }

    func convertDateFromUnixTimestamp(timestamp:Double) -> Date {
        return Date(timeIntervalSince1970: timestamp)
    }
}
```

Codefragment 3.1: Konvertierungslogik in Swift geschrieben

```kotlin
class Converters {
    val constOunce = 31.1034768
    val sellingPriceFactor = 1.8
    fun addSalesMarginToPrice(rate: Double): Double {
        return rate * sellingPriceFactor
    }

    fun calculatePrice(rate: Double): Double {
        return round(((1 / rate) / constOunce) * 100) / 100.0
    }

    fun convertDateFromUnixTimestamp(timestamp: Double): Date {
        return Date(timestamp)
    }
}
```

Codefragment 3.2: Konvertierungslogik zu Kotlin transpiliert (SequalsK)

Zunächst wurde auch Swiftlin zum Vergleich herangezogen. Bei einem zweiten Versuch mit nahezu unverändertem Code konnte der Online-Transpiler jedoch nicht mehr dazu gebracht werden, Kotlin-Code auszugeben.

Zum Vergleich wurde auch ChatGPT mit der Übersetzung des Codes beauftragt, siehe Listing 3.3.

```kotlin
import java.util.Date
import kotlin.math.round

class Converters {
    private val constOunce = 31.1034768
    private val sellingPriceFactor = 1.8

    fun addSalesMarginToPrice(rate: Double): Double {
        return rate * sellingPriceFactor
    }

    fun calculatePrice(rate: Double): Double {
        return round(((1 / rate) / constOunce) * 100) / 100.0
    }

    fun convertDateFromUnixTimestamp(timestamp: Double): Date {
        return Date((timestamp * 1000).toLong())
    }
}
```

Codefragment 3.3: Konvertierungslogik zu Kotlin transpiliert (ChatGPT)

Lediglich ChatGPT nahm bei der Konvertierung des Unix-Timestamps eine Multiplikation mit dem Faktor 1000 vor und übersetzte somit korrekt nach Kotlin. Es war auch das

einzige Tool, das den *.toLong()* Cast-Operator hinzufügte, da die Date-Funktion der java.util.Date Bibliothek nur einen Long-Wert akzeptiert.

Um einen Dezimalwert auf 2 Nachkommastellen zu runden, verwendet ChatGPT eine Methode aus der Kotlin Math Library und gibt auch an, welche Bibliotheken zu importieren sind.

Es ist jedoch zu beachten, dass ChatGPT nicht deterministisch arbeitet. Wenn man den Code mehrmals zur Übersetzung eingibt, erhält man immer wieder neue Versionen, die sich stets ein wenig unterscheiden.

Noch schwieriger wird es, wenn im Kotlin-Code Bibliotheken verwendet werden, um spezielle Funktionalitäten zu implementieren. Ein Transpiler kann nicht wissen, welche Bibliothek für die Zielsprache verfügbar ist und die entsprechende Funktionalität enthält. Ein Beispiel wäre die Bibliothek Ktor, die in Kotlin häufig verwendet wird, um einen HttpClient zu implementieren.

Man kann also festhalten, dass es derzeit keine fehlerfreie, automatisierbare Methode gibt, um Code zwischen Swift und Kotlin zu übersetzen. Eine Automatisierung ist eventuell bis zu einem gewissen Grad mit dem Plugin des SequalsK Transpilers möglich, jedoch nicht auf CI/CD-Ebene. Das Plugin war im Januar 2024 nicht mit den zu diesem Zeitpunkt aktuellen Versionen von Android Studio und IntelliJ IDEA kompatibel und konnte daher nicht weiter untersucht werden.

3.2.2 Transpiler bei cluetec

Wie in Kapitel 1.2 erwähnt, wurden bei cluetec im Jahr 2001 die ersten mobilen Applikationen entwickelt. Die Anwendungslogik (bei mQuest als *mobilecore* bezeichnet) wurde in Java 1.7 geschrieben, UI und Datenpersistierung, wie auch die Synchronisation der Daten mit dem Backend (im folgenden verkürzt als *Sync* bezeichnet), wurden über eigens definierte Schnittstellen angedockt. Somit war von Beginn an eine modulare Struktur gewählt worden, die es 2010 erlaubte, den mobilecore bei der Entwicklung der Android App wiederzuverwenden. Die UI, eine lokale Datenhaltung mit SQLite und die Logik für den Sync mussten neu programmiert werden.

Um 2012 wurde die Entscheidung getroffen, auch eine App für das iPhone zu entwickeln. Zu dieser Zeit umfasste das Apple Ökosystem für die Applikationsentwicklung Objective-C für die Logik, Cocoa für die UI und CoreData für die Persistenz. Für die Daten-Synchronisation über einen RESTful Webservice stand mit RestKit ein Objective-C Framework zur Verfügung. Es wurde nun nach einer Möglichkeit gesucht, zumindest den mobilecore nicht neu programmieren zu müssen. Google stand wohl vor einer ähnlichen Herausforderung und hat daher J2ObjC entwickelt und als Open-Source **CLI**-Tool zur

Verfügung gestellt [80]. J2ObjC ist ein Transpiler, der Java-Quellcode in Objective-C Code für die iOS-Plattform (iPhone/iPad) übersetzt.

So war es möglich geworden, den mobilecore als Java-Bibliothek in Android Studio und die übersetzte iOS-Bibliothek entsprechend in Xcode einzubinden. Die anderen oben genannten Applikationskomponenten werden plattformspezifisch, also jeweils nativ programmiert.

2020 wurde dann mit mQuest online eine Thin-Client Browser App vorgestellt. Diese wurde als **SPA** in *AngularJS* geschrieben und nutzt ebenfalls den mobilecore, allerdings ohne lokale Persistenz.

Im Jahr 2022 sah man sich mit der Situation konfrontiert, den mobilecore um eine Funktionalität erweitern zu müssen, die mit Java 1.7 nicht abgebildet werden konnte. In der Zwischenzeit war im Unternehmen Know-how in der Kotlin-Programmierung aufgebaut worden und die Idee eines App-Relaunches auf Basis von Kotlin Multiplatform stand im Raum. So entschied man sich, die fehlende Funktionalität in eine Kotlin-Bibliothek zu verpacken und als *mobilecorenative* zusätzlich einzubinden. Dies kann auch als eine Art Testballon betrachtet werden.

3.3 Literaturrecherche

Arbeiten zur Cross-Plattform-Entwicklung gibt es seit mehr als 10 Jahren, aber die Landschaft der Entwicklungs-Frameworks hat sich in den letzten Jahren stark verändert. Einige dieser Frameworks sind praktisch vom Markt verschwunden, jüngere sind gereift und konnten ihre anfänglichen Schwächen zu einem großen Teil beseitigen. Aus diesem Grund beschränkt sich die Betrachtung auf Veröffentlichungen der letzten 5 Jahre, insgesamt wurden 16 Arbeiten berücksichtigt.

Neben der theoretischen Betrachtung wird in vielen Arbeiten auch die Erstellung von Prototypen beschrieben, um einerseits die Funktionalität und User Experience des Frameworks beurteilen zu können und andererseits die Performance der Apps und den Zugriff auf Hardwarefunktionen zu testen. Viele Arbeiten erstellen jedoch ausschließlich Android Apps (mit Ausnahme von Mota [21], Nawrocki [20], Armagan [19], Evert [26] und Fayzullaev [25]), die erstellten Anwendungen sind sehr inhomogen und teilweise sehr einfach (so wird bei Suri et al. ein Tic Tac Toe Spiel programmiert, welches weder Zugriff auf Hardware-Komponenten wie Kamera oder Sensorik erfordert, noch besondere Anforderungen an die Leistungsfähigkeit stellt, was etwa die Verarbeitung hoher Datenmengen betrifft [81]). Auch die verwendeten Testmethoden sind stets unterschiedlich, einige verwenden dafür echte Telefon-Hardware, einige verwenden Simulatoren, einige lassen die Apps im Debug-Modus für Leistungstests laufen, obwohl dies die Ergebnisse

verfälscht. Man kann daher die Ergebnisse der bisher veröffentlichten Arbeiten nur bedingt miteinander vergleichen.

3.3.1 Arbeiten zu Cross-Plattform-Entwicklung

Zunächst fällt auf, dass sich von 12 der betrachteten Arbeiten (Artikel/Konferenzbeiträge und Abschlussarbeiten), die zwischen 2020 und 2023 veröffentlicht wurden, fast alle mit React Native und Flutter beschäftigen. Shwetha et al. untersuchen kein konkretes Framework [82] und Tom et al. vergleichen React Native mit Xamarin [83] (bemerkenswerterweise wurde hier nicht der Nachfolger .NET MAUI zum Vergleich herangezogen, obwohl Xamarin nur noch bis Mai 2024 Updates erhalten wird [84]). Die Autoren begründen die Wahl von React Native und Xamarin mit der Popularität von JavaScript und C# in der Stackoverflow Developer Survey 2019, wobei der Trend zeigt, dass C# leicht rückläufig ist und mittlerweile von TypeScript überholt wurde [31]. Ein ähnliches Bild zeigt sich bei GitHub [85]. Kotlin und Dart (Flutter) sind in diesen Auswertung nicht unter den Top 10 zu finden.

Kotlin Multiplattform (KMP) ist ein Newcomer im Feld der Cross-Plattform-Frameworks und findet daher bisher nur in wenigen Publikationen Erwähnung.

Fayzullaev vergleicht es u.a. mit Multi-OS Engine, welches heute wie in Kapitel 2.4 beschrieben wohl keine Rolle mehr spielt [25]. Daneben wird KMP in dieser Thesis noch mit Flutter verglichen und herausgestellt, dass KMP nur eine gemeinsam genutzte Anwendungslogik unterstützt, während bei Flutter auch das UI aus einer gemeinsamen Codebasis erzeugt wird[5].

In ihrer Masterarbeit vergleicht Evert KMP nicht mit anderen Cross-Plattform-Frameworks, sondern nur mit nativer Entwicklung [26]. Die Arbeit zielt auf drei Hauptaspekte ab: vom Benutzer wahrgenommene Performance, Anwendungsgröße und Entwicklerproduktivität. Die zu untersuchenden Metriken unfassen LOC (Anzahl der Codezeilen), Entwicklungsdauer, Anwendungsgröße und Startzeit der Anwendung. Für die Studie wird eine Beispielanwendung verwendet, die Funktionen wie Datenübertragung über das Internet, Parsing von Daten, GUI-Komponenten, mehrere Ansichten und zentrale Anwendungslogik enthält.

Suri et al. sprechen von *Kotlin* als Cross-Plattform-Framework, der Begriff *Kotlin Multiplatform* wird nicht erwähnt, die im Zuge der Arbeit entwickelten Apps wurden nur auf

5 In dieser Arbeit aus dem Jahr 2018 wird von Kotlin Native gesprochen, was nicht der von JetBrains heute verwendeten Nomenklatur entspricht. Während Kotlin Multiplatform das eigentliche plattformübergreifende Framework bezeichnet, steht Kotlin Native für den Teil, in dem Kotlin-Code in nativen Code übersetzt wird, um beispielsweise auf iOS ausgeführt zu werden [37].

einem Android Gerät getestet. Insofern ist der Unterschied zu einer nativen Android App Entwicklung auf Basis von Kotlin nicht so recht erkennbar. Verglichen wird hier React Native und Flutter [81].

Einen Ansatz für eine strukturierte und reproduzierbare Bewertungsmöglichkeit für den Einsatz von Cross-Plattform-Frameworks versuchen Tom et al. zu finden [83]. Ihre Arbeit baut auf einer früheren Publikation des Co-Autors Tim A. Majchrzak [86] auf und soll die Auswahl von Frameworks durch gewichtete Kriterien und entsprechende Checklisten ermöglichen. Allerdings bewerten die Autoren den Prozess der Evaluierung als sehr zeitaufwändig und stellen für zukünftige Arbeiten die Entwicklung eines Formulars in Aussicht, das bei der Bewertung der Kriterien helfen soll.

Alkhalifah (dessen Arbeit auf der von Majrashi et al. aufsetzt [87]) befasst sich mit der Usability von mobilen Apps, die mit Cross-Plattform-Frameworks erstellt wurden, und auch hier wird Flutter und React Native zum Vergleich herangezogen [88]. Während React Native bei der Benutzerfreundlichkeit und der Navigation glänzte, war Flutter in Bezug auf die Benutzerzufriedenheit, die Performance und die umfassende Dokumentation überlegen.

Shwetha et al. vergleichen die native Entwicklung mit der Entwicklung hybrider und Cross-Plattform-Anwendungen [82]. Dabei betrachten sie unter anderem die Kriterien Performance, Zugriff auf native Features, Wiederverwendbarkeit des Codes und die Verfügbarkeit von Drittanbieter-Bibliotheken. Sie entwickeln ein einfaches Flussdiagramm zur Entscheidungsfindung und werfen auch einen Blick auf nicht-technische Aspekte. Dazu gehören der Support durch die Community, die Entwicklungszeit und die Kosten für die Entwicklung und Wartung der Software, aber auch der Reifegrad der Frameworks und wie leicht es ist, Entwickler dafür zu finden.

Tabelle 3.1 gibt einen Überblick über die betrachteten Publikationen.

Publikation & Jahr	Untersuchte Cross-Plattform Frameworks	Schwerpunkte
Shwetha et al., 2023 [82]	-	Nativ, Hybrid, Cross-Plattform
Oliveira et al., 2023 [22]	Flutter, React Native, u.a.	Analyse des Ressourcenverbrauchs
Tom et al., 2023 [83]	React Native, Xamarin	Bewertungskriterien finden
Ahmad, 2023 [89]	Flutter, React Native, MAUI	Extensive Analyse, Securityaspekte
Suri et al., 2023 [81]	Flutter, React Native, Kotlin	Performance, Code Reusability
Nawrocki et al., 2021 [20]	Flutter, React Native, Xamarin	Performance, Developer Experience
Mota, 2021 [21]	Flutter, React Native	Performance
Alkhalifah, 2022 [88]	Flutter, React Native	Usability, Nutzerzufriedenheit
You; Hu, 2021 [90]	Flutter, React Nat., Xamarin	Arbeitsaufwand, UI Erstellung
Işitan; Koklu, 2020 [91]	Flutter, React Nat., Xamarin, u.a.	Performance
Armagan, 2020 [19]	Flutter, React Native	Vergleich mit nativer Entwicklung
Biørn-Hansen et al., 2020 [18]	Flutter, React Native	Performance, Ressourcen
Biørn-Hansen et al., 2019 [92]	-	Online Befragung von Entwicklern
Evert, 2019 [26]	Kotlin Multiplatform	Entwicklerproduktivität
Fayzullaev, 2018 [25]	Flutter, Kotlin Native, u.a.	Vergleich mit nativer Entwicklung
Wu, 2018 [17]	Flutter, React Native	Architektur Pattern, Routing

Tabelle 3.1: Liste verwandter Arbeiten

3.3.2 Performancebetrachtungen

Viele der Arbeiten befassen sich mit der Messung der App-Performance. Die Arbeit von Oliveira et al. legt den Schwerpunkt auf den erhöhten Ressourcenverbrauch beim Einsatz von Cross-Plattform-Frameworks, der letzendlich die Batterielaufzeit von mobilen Geräten negativ beeinflusst [22]. Zur Messung verwenden die Autoren die CLBG Benchmarks [93].

Ahmad verwendet in seiner Arbeit AppTim [94], um damit eine Test-App zu evaluieren [89]. Zum einen führt diese eine Schleife aus, welche einen zufällig generierten Text an eine Liste anhängt. Zum anderen ändert sie permanent die Hintergrundfarbe des Bildschirms, um ein Neuzeichnen der UI zu provozieren.

Ein Python-basiertes Tool namens *mobileperf* [95] wird von You und Hu eingesetzt, um die Performance einer zu Testzwecken entwickelten *Blockchain Coin Wallet* zu testen [90]. mobileperf kann nur für die Android Plattform verwendet werden.

Die weitaus meisten Forschenden verwenden die von Apple für iOS (Xcode Instruments, [96]) und von Google für Android (Android Profiler, [97]) bereitgestellten Tools.

Tabelle 3.2 gibt einen Überblick über die betrachteten Publikationen, die u.a. Performance-Kriterien betrachten.

Publikation & Jahr	**Untersuchte Features und Performance-Metriken**
Oliveira et al., 2023 [22]	CPU/Execution time, Memory, Energy
Ahmad, 2023 [89]	CPU, Memory, App size, HW access
Suri et al., 2023 [81]	CPU, Memory, App size
Nawrocki et al., 2021 [20]	CPU, RAM, App size, Launch time
Mota, 2021 [21]	CPU, Memory, FPS (Frames per second)
You; Hu, 2021 [90]	CPU, Memory, App size
Işitan; Koklu, 2020 [91]	CPU, Memory, App size, TTC, Energy, Network
Biørn-Hansen et al., 2020 [18]	CPU, TTC, Memory (PreRam, RAM)
Evert, 2019 [26]	Launch time, App size
Wu, 2028 [17]	FPS, Disk-I/O

Tabelle 3.2: Liste der Arbeiten mit Performance-Untersuchungen

Ein Vergleich der Performancetests ist, wie bereits erwähnt, kaum möglich, da die zugrundeliegenden Anwendungen sehr unterschiedlich sind. Gerade bei den Aussagen zur CPU-Last und Speicherauslastung von Flutter wird dies sehr deutlich, da sich diese zwischen den Arbeiten oft widersprechen. Die mit Flutter erstellten Apps waren in der Regel sehr kompakt, was die Größe des App-Bundles betrifft.

Zusammenfassend lässt sich sagen, dass Apps, die mit Cross-Plattform-Frameworks erstellt werden, eine etwas geringere Performance bzw. einen etwas höheren Ressourcenverbrauch aufweisen. Je nach Art der App kann dies jedoch vernachlässigbar sein, in diesem Fall überwiegen die Vorteile durch den geringeren Zeitaufwand für Entwicklung, Testen und Wartung einer gemeinsamen Codebasis.

4 Methoden

Hier wird dargelegt, mit welchen Methoden die Forschungsfrage beantwortet werden soll, wie also eine fundierte Entscheidung über die Auswahl eines Cross-Plattform-Frameworks getroffen werden kann.

Dazu werden zunächst Befragungen von firmeninternen Experten durchgeführt. Diese sollen einerseits aufzeigen, welche Kriterien im konkreten Fall wichtig sind.

Schließlich werden mit ausgewählten Frameworks Test-Apps gebaut. Diese sollen prototypisch jene Features beinhalten, die eine Beurteilung der Eignung des jeweiligen Frameworks für den konkreten Fall ermöglichen.

4.1 Befragung von Experten

Diese Befragungen sollen wertvolle Informationen über objektive und subjektive Kriterien liefern, die bei der Auswahl eines Cross-Plattform-Frameworks eine zentrale Rolle spielen. Darüber hinaus werden die Ergebnisse dazu beitragen, das Konzept und den Funktionsumfang des Prototypen zu definieren.

Bei der Auswahl der Methoden für die Expertenbefragung ist zunächst zu entscheiden, wie die Informationen gewonnen werden sollen. Quantitative Methoden wie Fragebögen liefern leicht vergleichbare Daten. Durch den Verzicht auf Freitextantworten kann es jedoch zu Informationsverlusten kommen, da nur zuvor recherchierte Themen berücksichtigt werden. Bei qualitativen Ansätzen, wie z.B. semi-strukturierten Interviews, erfordert die Auswertung besondere Sorgfalt, um die Vergleichbarkeit zu gewährleisten. Eine solche qualitative Inhaltsanalyse kann deutlich zeitaufwändiger sein, als die quantitative Auswertung von Fragebögen. Sie orientiert sich heute üblicherweise an den Verfahren, die von Mayring und Kuckartz hierfür entwickelt wurden [98].

Verschiedene Arbeiten[6] zeigen auf, dass mit der Kombination von quantitativen und qualitativen Methoden die Vorteile beider Ansätze kombiniert werden können und man

6 Die für diese Arbeit durchgeführte Literaturrecherche zu diesem Thema war nicht erschöpfend, und es kann weitere relevante Artikel zu diesem Thema geben, die nicht berücksichtigt wurden.

in der Folge eine umfassendere Antwort auf die Forschungsfrage erhalten kann ([99, S.4], [100]). Roch und Kuckartz sprechen in diesem Zusammenhang von *Mixed-Methods* [101, S.95-96], [102, S.18-21]. Dieser Ansatz wird auch für die vorliegende Arbeit gewählt.

Als Erhebungsform wird daher neben einem standardisierten Fragebogen das Experteninterview gewählt. Es wird, wie bei Misoch beschrieben, als Leitfadeninterview durchgeführt [103, S.120ff]. Der Leitfaden hat dabei eine strukturierende Funktion, von der im Leitfaden definieren Reihenfolge kann jedoch abgewichen werden, selbst ein Auslassen von Fragen ist möglich. Damit gehört das Leitfadeninterview zu den semi-strukturierten Erhebungsformen [103, S.65].

Die Interviews werden aufgezeichnet und zur leichteren Auswertung softwaregestützt transkribiert. Aus Datenschutzgründen wird hierfür eine Software verwendet, die die Transkription lokal auf dem Computer durchführt [104].

Die Auswertung dieser verschriftlichten Interviews erfolgt in mehreren Schritten[7]. Eine solche Auswertung wird immer anhand eines Kategoriensystems vorgenommen (synonym werden auch die Begriffe Codes bzw. Codierung verwendet). Die Codes können vorab definiert (deduktives Vorgehen), oder aber während der Arbeit am Text induktiv entwickelt werden. Die beiden Verfahren müssen sich jedoch nicht ausschließen [102, S.129]. So wird bei der vorliegenden Arbeit eine inhaltlich strukturierende qualitative Inhaltsanalyse nach Kuckartz durchgeführt [102, S.129ff]. Die einzelnen Schritte lassen sich wie folgt beschreiben:

1. Initiierende Textarbeit, Memos, Fallzusammenfassungen.
2. Bildung von Hauptkategorien.
3. Daten mit Hauptkategorien codieren (1. Codierprozess).
4. Induktiv Subkategorien bilden.
5. Daten mit Subkategorien codieren (2. Codierprozess).
6. Analysieren.
7. Ergebnisse verschriftlichen, Vorgehen dokumentieren.

Der bei Kuckartz genannte erste Schritt kann hier entfallen, da die Hauptkategorien deduktiv aus den in der Literaturrecherche gefundenen Kriterien entwickelt werden.

7 Die Techniken der qualitativen Inhaltsanalyse stammen ursprünglich aus den Kommunikationswissenschaften und werden in weiterentwickelter Form häufig in den Sozialwissenschaften eingesetzt [105, S.601] und [102, S.33]. Sie können aber prinzipiell auch in anderen Bereichen angewendet werden.

Die Codierungsvorgänge können durch Software unterstützt werden. Hierfür wurde das Open Source Tool TAMS (Text Analysis Markup System) ausgewählt [106]. Es erlaubt, entsprechende Textstellen mit einer Kategorie zu markieren, um anschließend Auswertungen auf Basis dieser Kategorien durchzuführen.

4.2 Prototypen

In die Evaluierung der Entwicklungsframeworks sollen auch praktische Erfahrungen einfließen. Zu diesem Zweck wird mit den beiden vorausgewählten Frameworks jeweils eine prototypische Testanwendung erstellt. Dies ermöglicht zum einen den Vergleich der jeweils benötigten Zeit. Zum anderen ermöglicht dies eine Gegenüberstellung der Ergebnisse hinsichtlich Qualität und Funktionalität. Es wird also jeweils geprüft, ob die App gut funktioniert, ob sie sich nativ anfühlt und ob alle Funktionen mit dem Framework realisiert werden können.

Der Funktionsumfang soll folgendes beinhalten:

- Texteingabe
- Multimedia-Aufnahme (Bild, Video mit Ton)
- Multimedia-Wiedergabe (Bild, Video mit Ton)
- Lokale Persistenz der Daten
- Auslesen von GPS-Daten
- NFC Schreib-/Lesezugriff
- Erstellen und Scannen eines QR-Codes
- Abrufen von Daten von einer REST-API über http und JSON-Deserialisierung
- Mehrsprachigkeit der UI (DE/EN)

Um all diese Funktionen in der App unterzubringen, wird als Showcase eine Notiz-App für Musikliebhaber entwickelt. Zu einem Lied kann eine Notiz angelegt werden, die neben Titel und Interpret weitere Informationen enthalten kann. So kann hier auch ein Foto und ein Video hinzugefügt werden. Alle Notizen werden auf der Startseite in einer Listendarstellung angezeigt.

Die Detailansicht zeigt das Foto und das Video, falls vorhanden. Über einen Button „Gerade gehört“ wird die GPS-Position mit dem aktuellen Zeitstempel gespeichert und angezeigt.

Über einen weiteren Button können Informationen zum Musikstück aus dem Online-Dienst Discogs [107] abgerufen werden. Ein Teil der abgerufenen Informationen wird nach dem Abruf in Feldern angezeigt, die je nach Verfügbarkeit der Informationen eingeblendet werden.

Die Notiz (Titel, Interpret und Notiztext) kann entweder über einen QR-Code oder einen NFC-Tag exportiert und auch wieder importiert werden.

Abbildung 4.1 zeigt die Wireframe-Darstellung der UI (erstellt mit Figma [108]).

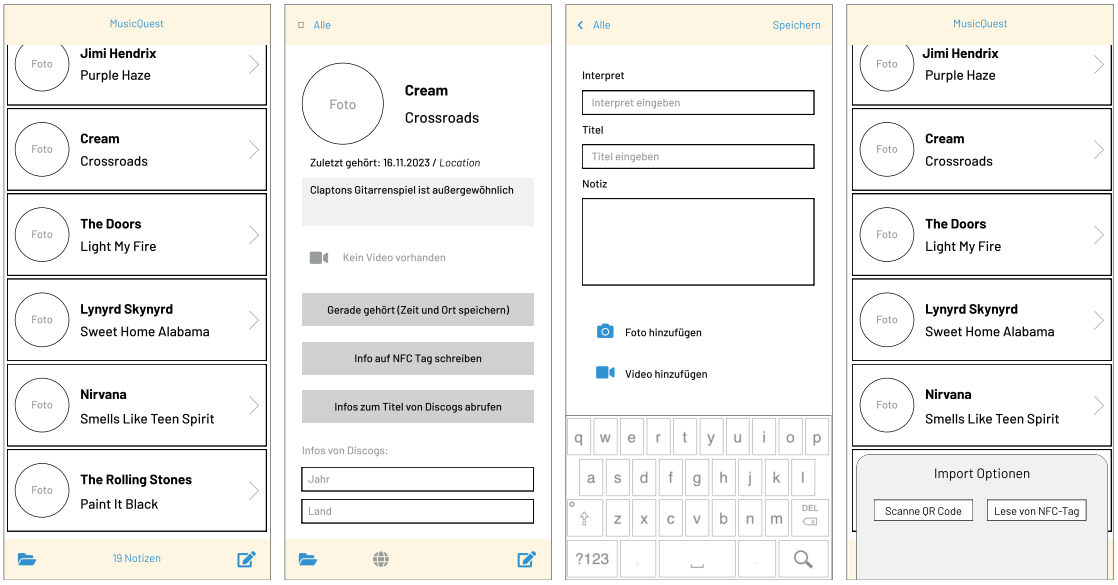

Abbildung 4.1: Wireframe der Prototypen-App

5 Ergebnisse

In diesem Kapitel werden die Untersuchungsergebnisse diskutiert. Hierzu zählen die Resultate der Expertenbefragung, welche sowohl Kriterien für die Entscheidungsfindung als auch deren jeweilige Gewichtungen umfassen.

Darüber hinaus werden die Erfahrungen bei der Entwicklung der Prototypen analysiert und anhand der zuvor festgelegten Kriterien bewertet. Die Resultate der Fragebögen werden grafisch aufbereitet.

Basierend auf diesen Erkenntnissen wird eine Entscheidungsmatrix erstellt. Aus ihr ergibt sich eine Empfehlung für den spezifischen Fall mQuest. Die Matrix selbst findet sich im Kapitel 6.1.

5.1 Experteninterviews

Die Interviews mit den internen Experten wurden mithilfe der in Kapitel 4.1 beschriebenen Methoden ausgewertet. Die ermittelten Kategorien werden jeweils in einer Mindmap visualisiert. Darauffolgend werden die Aussagen der Experten in Bezug auf die im Rahmen der Inhaltsanalyse identifizierten Haupt- und Subkategorien zusammengefasst.

5.1.1 mQuest-App

Dieser Teil befasst sich mit den Aussagen der Experten zum aktuellen Stand der mQuest-App und zu strategischen Entwicklungen des mQuest-Ökosystems. Dabei werden Erweiterungen und Verbesserungen der mQuest-App diskutiert, die von neuen Features bis hin zu Optimierungen in den Bereichen User Interface, Usability, App-Architektur und Performance reichen. Eine Übersicht über die betreffenden Themen ist in Abbildung 5.1 gegeben.

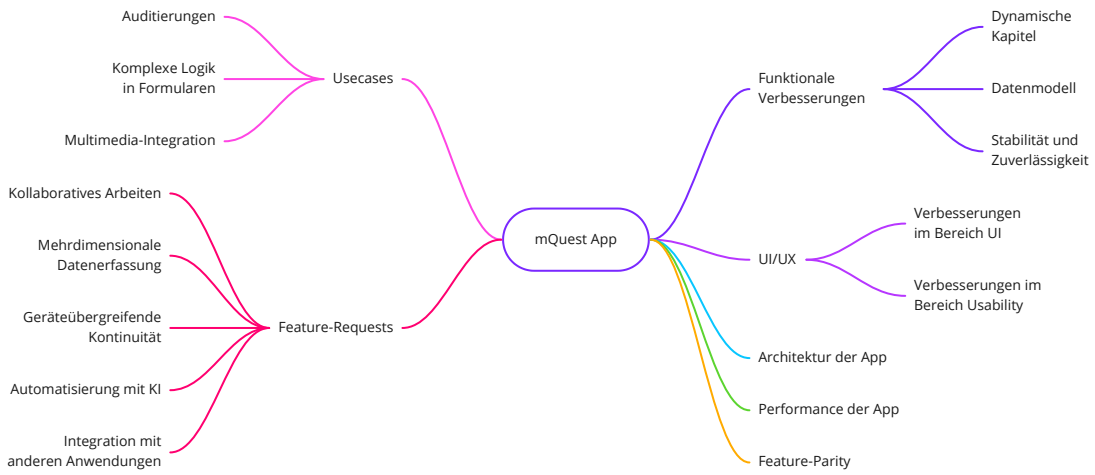

Abbildung 5.1: Mindmap zu den mQuest-App Themen

Anwendungsfälle für die mQuest-App

Die Kernkompetenz des mQuest-Ökosystems liegt in der Abbildung komplexer Anwendungsfälle in digitalen Formularen. Die mQuest-App spielt dabei eine zentrale Rolle, da sie als primäres Werkzeug zur Datenerfassung dient. Sie ermöglicht das Ausfüllen von digitalen Formularen, die aus mehreren Datensätzen bestehen und an unterschiedliche Szenarien angepasst werden können. Dies deckt eine Vielzahl von Anwendungsfällen ab, unter anderem:

1. Auditierungen: Hier wird ein Zuwachs an Funktionalität erwartet, um dem wachsenden Bedarf an detaillierten und komplexen Audits und Protokollen gerecht zu werden.

2. Komplexe Logik in Formularen: mQuest kann auch sehr komplexe Logik verarbeiten, wobei die Antworten auf bestimmte Fragen den weiteren Verlauf bestimmen können.

3. Multimedia-Integration: Die App integriert multimediale Elemente wie Fotoaufnahmen und NFC-Scanner und erweitert damit die Möglichkeiten der traditionellen Datenerfassung. Darüber hinaus ist es denkbar, die mQuest-App in Zukunft um neue Funktionalitäten wie Augmented Reality zu erweitern.

Feature-Request: Kollaboratives Arbeiten

Kollaboratives Arbeiten innerhalb der mQuest-App wird von Kunden zunehmend nachgefragt. Mehrere Auditoren sollen gleichzeitig an einem Audit arbeiten können. Dies erfordert eine Funktionalität, die es einem Auditor erlaubt, die Arbeit eines anderen fortzusetzen oder gemeinsam an einem Formular zu arbeiten. Eine Herausforderung besteht darin, trotz der Notwendigkeit, offline arbeiten zu können, eine Synchronisation zwischen mehreren Nutzern zu ermöglichen. Lokale Daten müssen konsistent gehalten werden, um sie dann zu synchronisieren, wenn eine Netz-Verbindung verfügbar ist. Dabei muss die Datenintegrität stets gewährleistet bleiben. Dies könnte durch fortgeschrittene Synchronisationsmechanismen erreicht werden, die Strategien zur Datenkonfliktlösung beinhalten.

Feature-Request: Mehrdimensionale Datenerfassung

mQuest soll so weiterentwickelt werden, dass Fragen zu einem logischen Datensatz zusammengefasst werden können. Diese Funktion würde es dem Nutzer ermöglichen, mehrere Einzelfragen, die zusammen einen größeren Sachverhalt abbilden, in einen zusammenhängenden Kontext zu stellen. Beispielsweise könnte die übergeordnete Frage „Wer hat Produkt X gekauft?“ durch die aggregierten Antworten auf Einzelfragen wie Vorname, Nachname, Alter und Beruf beantwortet werden, wodurch ein kohärenter Datensatz entsteht.

Weitere funktionale Verbesserungen

1. Dynamische Kapitel: Es besteht die Notwendigkeit, dynamische Kapitel besser in die App zu integrieren, um Workarounds zu vermeiden.

2. Datenmodell: Es wird erwartet, dass das Thema Audits in den nächsten Jahren noch weiter an Bedeutung gewinnen wird. Daraus könnte sich die Notwendigkeit ergeben, das Datenmodell anzupassen und zu erweitern, um spezifischen Anforderungen gerecht zu werden.

3. Geräteübergreifende Kontinuität: Eine nahtlose Benutzererfahrung über verschiedene Geräte hinweg, ähnlich der Handoff-Funktionalität von Apple. Nutzer sollen ihre Arbeit auf einem Gerät beginnen und auf einem anderen fortsetzen können.

4. Automatisierung, ggf. durch den Einsatz von KI: Die App soll intuitive Interaktionen ermöglichen, so dass Nutzer beispielsweise nur die Probleme erfassen, die sie sehen, und die App automatisch die Zuordnung zu den relevanten Kategorien vornimmt.

5. Integration mit anderen Anwendungen: Beispielsweise wird eine Integration mit Kalenderanwendungen als sinnvolle Ergänzung angesehen.

6. Stabilität und Zuverlässigkeit: Als Auszeichnungsmerkmal der mQuest-App wird auf Stabilität und Beständigkeit weiterhin großen Wert gelegt, dies muss bei allen Neuerungen gewährleistet bleiben.

Verbesserungen im Bereich UI

Die UI einer App spielt eine entscheidende Rolle für das Nutzererlebnis und ist ein wichtiger Faktor für die Kundenzufriedenheit. Die Herausforderung besteht darin, die UI modern und ansprechend zu gestalten und gleichzeitig die native Performance und das Look and Feel beizubehalten, um die Nutzer nicht durch ungewohnte Bedienungsmuster zu irritieren. Die UI sollte daher plattformspezifisch gestaltet werden. Langfristig muss eine Balance zwischen plattformübergreifender Entwicklung und plattformspezifischer Anpassung gefunden werden.

Verbesserungen im Bereich Usability

Das Ziel ist die Entwicklung einer UI, die unaufdringlich und benutzerfreundlich ist. Ideal ist eine User Experience, die so fließend, natürlich und intuitiv gestaltet ist, dass die Komplexität der Anwendung für den Anwender unbemerkt bleibt. Ein modernes Design und eine intuitive Bedienung erhöhen die Freude an der Verwendung der App. Dies fördert die Eigenständigkeit der Nutzer und motiviert sie, mehr Projekte mit mQuest umzusetzen. Eine gute Usability ermöglicht es dem Nutzer ohne Ablenkung und mit einem geringen Aufwand an Klicks und wenig Scrollen seine Daten schnell und effizient einzugeben.

Architektur der App

In der derzeitigen Anwendungsarchitektur sind Persistenz und Synchronisation plattformspezifisch implementiert, was zu einer Verdoppelung des Codes und damit des Testaufwands

führt. Es ist zu prüfen, inwieweit eine Zentralisierung möglich ist, um den Testaufwand zu reduzieren und die Wartung zu vereinfachen.

In jedem Fall wird sich die Implementierung grundlegend ändern, insbesondere unter Berücksichtigung moderner Entwicklungen im Bereich der asynchronen und parallelen Verarbeitung, die in der Vergangenheit nicht zur Verfügung standen. Anwendungslogik und UI sollten auf jeden Fall getrennte Module bleiben, eine schwache Kopplung ist hier anzustreben.

Die bisherige Architektur der Anwendungslogik mit ihren spezifischen Implementierungen von Datenstrukturen und Navigation würde bei einem Relaunch anders aufgebaut.

Performance der App

Die Wahl der Entwicklungstechnologie darf sich nicht negativ auf die Geschwindigkeit der Anwendung auswirken, insbesondere bei komplexen Formularen und mehrsprachigen Inhalten. Anwender erwarten eine schnelle und responsive Benutzeroberfläche. Insbesondere für Geschäftskunden, die mit sehr komplexen Formularen arbeiten, ist Effizienz entscheidend. Um die Performance zu maximieren, könnte die UI idealerweise nativ gestaltet werden. Bisherige Versuche mit JavaScript-basierten Frameworks führten zu keinen zufriedenstellenden Ergebnissen.

Feature-Parity

Beim Thema Feature-Parity geht es um die Herausforderung, gleiche Funktionalitäten auf verschiedenen Plattformen anzubieten. Bisher gibt es Unterschiede, z.B. ist das Auslesen von NFC-E-Tickets nur auf Android möglich. In Zukunft wird angestrebt, Funktionen für beide Plattformen gleichzeitig zu entwickeln, um den Erwartungen der Nutzer gerecht zu werden. Dies war in der Vergangenheit aus Kostengründen nicht immer der Fall. Zudem fehlt mQuest-Online derzeit die Offline-Funktionalität.

5.1.2 Organisatorische Themen

In den folgenden Abschnitten werden die Aussagen zu den Herausforderungen eines App-Relaunches zusammengefasst. Diese reichen von der Neuausrichtung der Entwicklungsprozesse über die Adaption technologischer Innovationen bis hin zur effizienten Teamorganisation und einem möglichen Migrationspfad. Es wird beleuchtet, wie diese Aspekte zusammenwirken, um sowohl die Entwicklererfahrung zu verbessern als auch langfristige Kosteneinsparungen zu erzielen. Abbildung 5.2 zeigt eine Übersicht über die wichtigsten Stichworte.

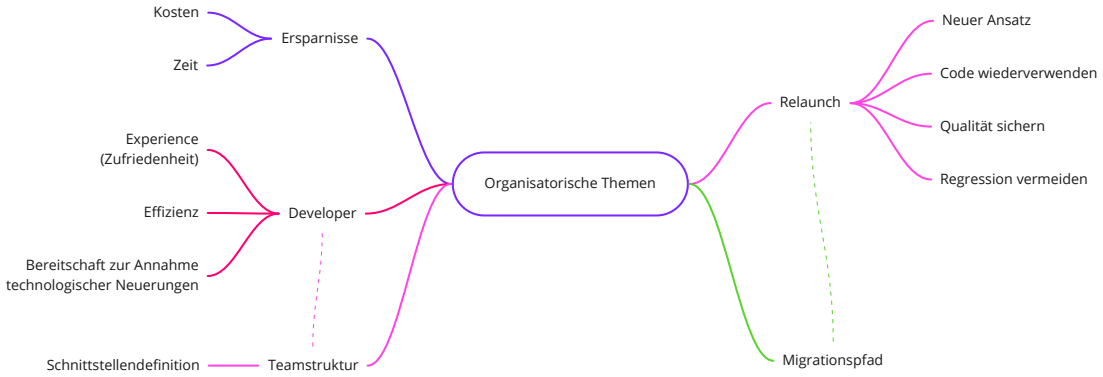

Abbildung 5.2: Mindmap zu organisatorischen Themen

Relaunch

Beim Relaunch der App wünschen sich vor allem die Entwickler einen komplett neuen Ansatz. Dabei ist es aber auch wichtig, die Qualitätssicherung kontinuierlich zu integrieren. Ziel ist es, eine dynamischere und zukunftsfähige Anwendung zu entwickeln, die neue Anwendungsfälle ohne Workarounds unterstützt. Es besteht jedoch die Gefahr, dass der Verlust des nativen Look and Feels bei einem Cross-Plattform-Ansatz die Benutzererfahrung beeinträchtigt. Regressionsfehler können auftreten, wenn Funktionen in neuen Technologien nachgebildet werden, ohne die ursprünglichen Absichten und Implementierungen zu verstehen. Darüber hinaus kann die Abhängigkeit von Convenience-APIs und externen Bibliotheken die Flexibilität und Wartbarkeit der Anwendung einschränken. Eine plattformübergreifende Lösung mag effizient erscheinen, aber plattformspezifische Eigenheiten können dennoch individuelle Anpassungen erfordern. Eine wohlüberlegte

Entscheidung für neue Technologien ist unerlässlich, da diese langfristig gepflegt und gewartet werden müssen, um Stabilität und Datensicherheit zu gewährleisten.

Developer-Experience

Für die Auswahl von Technologien ist auch die Zufriedenheit und Effizienz der Entwickler entscheidend. Mehrfachentwicklungen von Anwendungslogik für verschiedene Plattformen werden als mühsam betrachtet, wohingegen das Erstellen der Benutzeroberfläche als kreativer Prozess gesehen wird. Die Entwickler bevorzugen moderne Technologien, die auf vorhandenem Wissen aufbauen und legen Wert auf eine umfangreiche Standardbibliothek, aber auch auf die Verfügbarkeit gut gepflegter Pakete. Insbesondere wird Kotlin als moderne und elegante Sprache geschätzt. Effizientes Toolmanagement, gute Dokumentation und Community-Support sind wichtig für eine effiziente Entwicklungsarbeit. Die Möglichkeit, die neuesten APIs zu verwenden und einfach auf native Funktionen zuzugreifen, wird geschätzt, während veraltete Techniken weniger beliebt sind.

Bereitschaft zur Annahme technologischer Neuerungen

Das Unternehmen zeigt eine starke Offenheit für Innovationen, wobei das bestehende Wissen in Sprachen wie Java, Kotlin und Swift durch Erfahrungen einiger Mitarbeiter mit Flutter und Dart ergänzt wird. Neue Technologien sollten für einen Großteil der Belegschaft erlernbar sein und auf vorhandenen Kompetenzen aufbauen oder zumindest einen schnellen Einarbeitungsprozess ermöglichen. Einfachheit und Benutzerfreundlichkeit sind dabei entscheidend. Zu spezielle oder komplexe Technologien werden eher kritisch betrachtet. Generell ist die Bereitschaft hoch, sich neuen Herausforderungen zu stellen und das Unternehmen hat bereits bewiesen, dass es fähig ist, sich neuen technologischen Trends anzupassen.

Ersparnisse in Kosten und Zeit

Die Entwicklung plattformspezifischer Anwendungen führt zu wiederholtem Arbeitsaufwand und steigert die Kosten, da Funktionen mehrmals entwickelt und gewartet werden müssen. Um diese Ineffizienz zu verringern, ist ein Wechsel zu einer einheitlichen Entwicklungsplattform notwendig, die es erlaubt, einmal entwickelte Features auf allen Systemen zu nutzen und zu warten. Frühere Cross-Plattform-Tools auf der Basis von Webtechnologien waren zu langsam und schwerfällig in der Integration. Moderne, kompilierte Sprachen wie Kotlin oder Rust versprechen eine schnellere Implementierung und einfache Aktualisierung mit neuen Features, wodurch Ressourcen effektiver genutzt werden können.

Teamstruktur

Eine ideale Organisationsstruktur würde verschiedene Entwickler für verschiedene Aufgaben vorsehen. Beispielsweise könnten Entwickler an der Anwendungslogik arbeiten, während UI-Experten die Benutzeroberfläche erstellen. Für ein kleines Unternehmen mit begrenzten Ressourcen ist der Cross-Plattform-Ansatz in dieser Hinsicht attraktiv. Die Schnittstellendefinition ist von zentraler Bedeutung, um eine nahtlose Zusammenarbeit zu ermöglichen. Ziel ist es, Technologien zu verwenden, die bereits von einem Großteil des Teams beherrscht werden oder bei der eine Einarbeitung schnell erfolgen kann, um die Abhängigkeit von spezialisiertem Plattformwissen zu minimieren.

Migrationspfad

Es ist angedacht, mit einer Basisversion zu beginnen, die die wichtigsten Funktionen enthält und parallel zum bestehenden Produkt iterativ weiterentwickelt wird. Die UI könnte zunächst nativ entwickelt werden, um sie später eventuell mit Technologien wie Compose Multiplatform auf eine einheitliche Codebasis zu migrieren. Es wird als besonders wichtig erachtet, sich von alten Methoden zu lösen und mit einem frischen Ansatz zu planen, anstatt sich auf die Wiederverwendung bestehender Code-Elemente zu konzentrieren.

5.1.3 Kernaspekte der Softwareentwicklung

Unter dieser Kategorie wurden Themen wie die effiziente Wiederverwendung von Code und das Management von Abhängigkeiten im Kontext der mQuest-Softwarearchitektur abgefragt. In den folgenden Abschnitten werden die Gedanken zu den Herausforderungen und Lösungsansätzen dargelegt. Außerdem wird der Hardwarezugriff durch plattformübergreifende Frameworks und die Integration moderner Technologien diskutiert. Die Relevanz von technologischen Altlasten für die Wartbarkeit wird aufgezeigt und Teststrategien erläutert. Darüber hinaus wird der Einfluss aktueller Technologietrends auf die Entwicklung von Betriebssystemen und Benutzerschnittstellen beleuchtet. Abbildung 5.3 zeigt die Zusammenhänge.

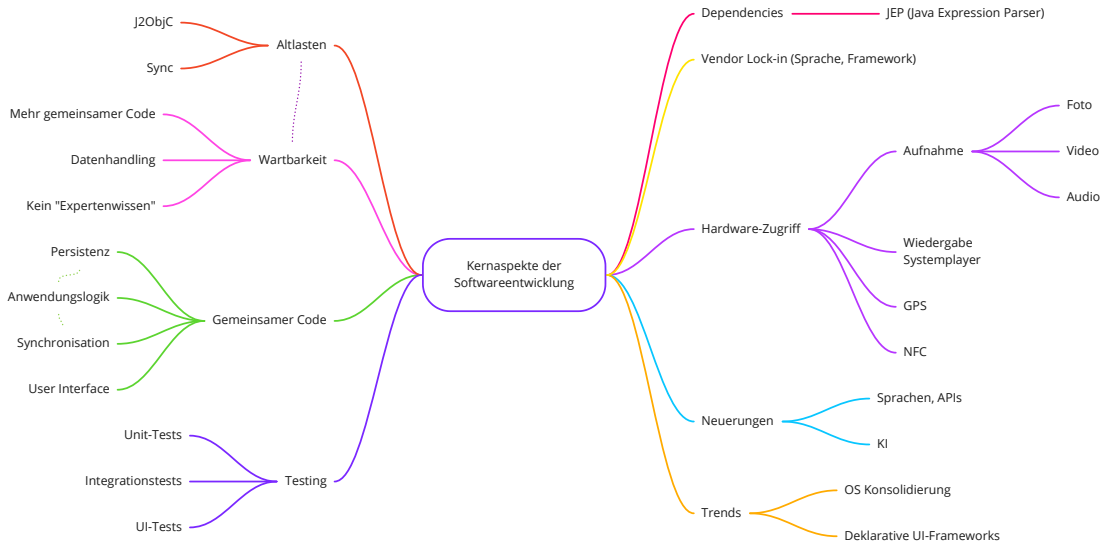

Abbildung 5.3: Mindmap zu Kernaspekten der Softwareentwicklung

Gemeinsamer Code

Es wird die Entwicklung eines Grundmoduls angestrebt, das zentrale Funktionen wie Synchronisation, Persistenz und Befragungslogik unabhängig von der UI umfasst. Die Wiederverwendung von existierendem Code kann in Betracht gezogen werden. Die Notwendigkeit, den Testaufwand zu reduzieren, ist ebenfalls eine wichtige Überlegung. Es wird betont, dass Persistenz und Synchronisation möglicherweise plattformunabhängig

implementiert werden können, wenn für die Persistenz eine Technologie wie SQLite mit einem geeigneten Wrapper für verschiedene Plattformen verwendet wird. Die Anwendungslogik sollte in jedem Fall mit einer gemeinsamen Codebasis realisiert werden. Die Möglichkeit, die UI plattformübergreifend zu entwickeln, wird in Betracht gezogen, die Auswirkungen auf die Qualität der Usability müssen jedoch noch evaluiert werden.

Dependencies und spezielle Features

Zwei Features sollen an dieser Stelle erwähnt werden, da sie für eine mobile App eher ungewöhnlich sind und eine Neuimplementierung bei einem Relaunch der App wahrscheinlich eine Herausforderung darstellen wird.

Zum einen verwendet mQuest die Bibliothek *Java Expression Parser* (JEP), die weit über ihre ursprüngliche Funktion hinaus erweitert wurde. JEP wird zur Interpretation und Transformation von benutzerdefinierten Skripten verwendet und kann nicht direkt in plattformübergreifende Frameworks wie Flutter oder KMP eingebunden werden. Die Prozesse rund um JEP sind komplex und fehleranfällig, insbesondere bei der Transformation und Synchronisation von Daten.

Darüber hinaus können Kunden Fließtexte in Fragebögen mit ausgewählten HTML-Tags manipulieren sowie Bilder und Grafiken einbinden. Diese HTML-Tags müssen in der App interpretiert werden. Da in iOS keine Webviews mehr innerhalb von Apps eingebunden werden können, muss hierfür ein eigener Parser/Interpreter entwickelt werden.

Vendor Lock-in

Bei der Migration zu einem neuen Entwicklungs-Framework, gerade auch bei einem plattformübergreifenden Ansatz, müssen auch die neuen Abhängigkeiten berücksichtigt werden, die durch die Wahl des Frameworks und der Programmiersprache entstehen.

Hardware-Zugriff

Die Anforderungen hinsichtlich des Hardwarezugriffs umfassen eine Vielzahl von Funktionalitäten, die aufgrund der unterschiedlichen Fragetypen in der mQuest-Anwendung erforderlich sind. Zu den grundlegenden Funktionen gehören das Aufnehmen von Fotos und Videos, die Nutzung des Mikrofons, die Wiedergabe von Dateien über den Systemplayer, GPS-Dienste und der Zugriff auf die NFC-Schnittstelle. Die Bibliotheken der Cross-Plattform-Frameworks sollten einen Zugriff auf die nativen Funktionalitäten bieten, um z.B. die Standardkamera mit den Grundfunktionalitäten nutzen zu können. Dies

ist zumindest dann zu beachten, wenn eine Codebasis (Logik und UI) für alle Targets verwendet wird.

Technologische Neuerungen

Für einen technologischen Neustart ist es entscheidend, moderne Technologien und Programmiersprachen zu integrieren, um den aktuellen und zukünftigen Anforderungen gerecht zu werden. Es wird eine technische Lösung benötigt, die schnell auf neue Funktionalitäten reagieren kann. Zukunftsfähigkeit bedeutet hier, nicht nur auf Abwärtskompatibilität zu setzen, sondern einen Technologiestack zu wählen, der mit aktuellen Erkenntnissen und Trends mitwächst und gut unterstützt wird. Es wird anerkannt, dass Cross-Plattform-Technologien sich in den letzten Jahren verbessert haben. Eine Integration von KI-Funktionen wird als zukunftsweisend angesehen.

Technologische Altlasten

Technische Schulden sind eine Herausforderung, die sich aus überholten Praktiken und veralteten Technologien ergibt, die sich im Laufe der Jahre angesammelt haben. Besonders betroffen sind Bereiche wie die Synchronisationsschicht, deren Komplexität Änderungen sehr schwierig macht und das Risiko von Nebeneffekten birgt. Dies führt zu ineffizienten Workarounds. Die derzeitige starke Kopplung an die UI und die Verwendung veralteter Technologien wie J2ObjC schränken die Entwicklung ein, da nur ältere Java-Versionen verwendet werden können und die Übersetzung nach Objective-C zu ungewollten Typkonvertierungen und potenziellen Fehlern führt. Diese Einschränkungen erschweren auch die Fehlerbehandlung und machen die Anwendung fehleranfällig und schwer zu warten. Die Notwendigkeit, die App mit modernen Technologien und Praktiken zu aktualisieren, wird immer dringlicher. So verlangt etwa Google regelmäßige Updates des Ziel-SDKs, um die App im Play Store zu halten.

Wartbarkeit

Anpassungen sollen in Zukunft leichter möglich sein. Das Entwicklungssystem soll nicht nur von wenigen Experten, sondern von jedem Entwickler erlernt werden können. Dies trägt zur Nachhaltigkeit der Software bei. Verbesserungspotential wird auch beim Datenhandling gesehen. Es soll einfacher werden, Daten durch das System zu bewegen, ohne komplexe Überlegungen anstellen zu müssen, wie sie von einem Punkt zum anderen gelangen. Eine Vereinheitlichung der Codebasis führt ebenfalls zu einer Vereinfachung der Wartung, da Änderungen und Erweiterungen effizienter umgesetzt werden können.

Testing

Die Testabdeckung ist in der aktuellen Situation mit ca. einem Drittel zu gering. UI- und Integrationstests fehlen. Es besteht Einigkeit darüber, dass in Zukunft eine umfassendere Teststrategie verfolgt werden sollte, um die Qualität und Zuverlässigkeit der Software zu gewährleisten. Plattformspezifische Tests sollten durch CI/CD-Mechanismen automatisiert durchgeführt werden. Die Bedeutung von UI-Tests wird zwar anerkannt, es wird aber darauf hingewiesen, dass diese derzeit nicht im Vordergrund stehen. Neben Unit-Tests liegt der Fokus eher auf Integrationstests, die beispielsweise den Einsatz von Docker-Containern zum Mocken der Interaktion mit einem Server beinhalten können.

Trends

Technologische Trends deuten darauf hin, dass Unternehmen ihre Betriebssysteme zusammenführen wollen. Apple arbeitet an der Integration von iOS, macOS und iPadOS. Google verfolgt ähnliche Ambitionen, um eine einheitlichere Erfahrung auf verschiedenen Android-Geräten zu ermöglichen. Microsoft zeigt Interesse an einer Rückkehr in den Mobilsektor, insbesondere im Zusammenhang mit der zunehmenden Verbreitung von ARM-Prozessoren, die die Grenzen zwischen den Plattformen weiter verwischen könnten. Zu beachten ist auch die Entwicklung von Compose Multiplatform, das in Zukunft die native UI-Schicht ersetzen und damit die Entwicklungseffizienz erhöhen könnte.

5.1.4 Frameworkbezogene Themen

Dieser Abschnitt befasst sich mit den Aussagen zur Popularität und dem Reifegrad von Cross-Plattform-Frameworks, der Qualität ihrer Dokumentation und den spezifischen Herausforderungen bei ihrer Verwendung. Es wird diskutiert, wie die Wahl des Frameworks die Entwicklung auf verschiedenen Plattformen beeinflusst. Die Zusammenhänge sind in Abbildung 5.4 visualisiert.

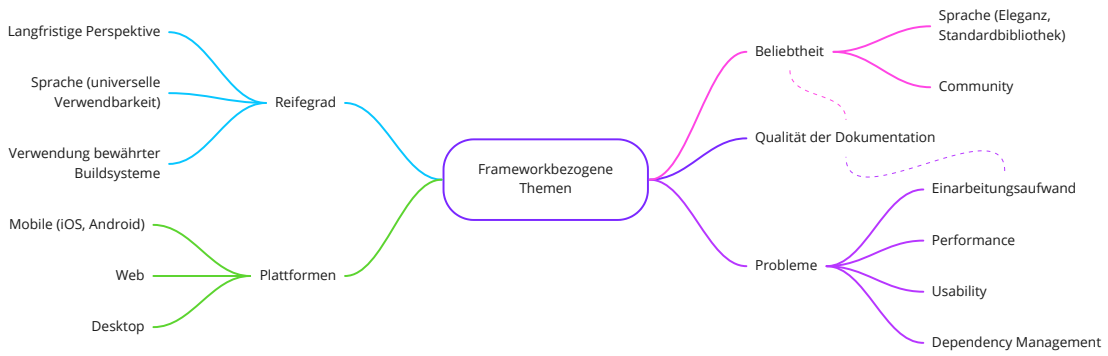

Abbildung 5.4: Mindmap zu frameworkbezogenen Themen

Beliebtheit von Entwicklungs-Frameworks

Die Popularität eines Frameworks hängt eng mit der Sprache zusammen, auf der es basiert. Kotlin erfreut sich großer Beliebtheit, da es als elegant, mächtig und ausgereift gilt. Die Stärke von Kotlin liegt nicht nur in seiner Effizienz für die Android-Entwicklung, es spielt auch eine große Rolle in serverseitigen Anwendungen, wie etwa im Spring Framework, wo es als eine moderne Alternative zu Java vollständig integriert ist. Die reichhaltige Standardbibliothek von Kotlin und die Unterstützung durch eine aktive und weitreichende Community machen die Sprache attraktiv. Hinzu kommt, dass Kotlin gut dokumentiert ist und in Verbindung mit Build-Systemen wie Gradle eine flexible Handhabung ermöglicht.

Reifegrad von Entwicklungs-Frameworks

Wichtig bei der Auswahl eines Frameworks ist die langfristige Perspektive. Ein bewährtes Framework sollte bereits etabliert sein und kontinuierlich verbessert werden, um auch in Zukunft relevant zu bleiben. Googles Empfehlung, Kotlin für die Android-Entwicklung

zu verwenden, hat zur Popularität der Sprache beigetragen, während Dart als neuer und weniger erprobt gilt. Ein ausgereiftes Framework verwendet bewährte Buildsysteme wie Gradle und bietet gute Möglichkeiten zur Integration nativer Systemfunktionen.

Qualität der Dokumentation

Eine gute Dokumentation ist für das Erlernen und die effektive Nutzung eines Frameworks unerlässlich. Die bisherigen Erfahrungen zeigen, dass die Qualität sehr unterschiedlich ist. Eine schlechte Dokumentation führt zu einer erhöhten Abhängigkeit von Suchmaschinen, um Lösungen für Probleme zu finden, die in der offiziellen Dokumentation nicht behandelt werden. Stack Overflow und qualitativ hochwertige Blog-Posts bieten eine reichhaltige Informationsquelle für die iOS-Entwicklung, die dichter und qualitativ hochwertiger zu sein scheint als jene für Android. Jetpack Compose wird als gut dokumentiert und intuitiv für die UI-Entwicklung für Android angesehen. Das Vorhandensein von Codebeispielen in der Dokumentation wird positiv bewertet. Abschließend wird betont, dass eine solide und zuverlässige Dokumentation für den Erfolg und die Zufriedenheit bei der Entwicklung unerlässlich ist.

Probleme beim Einsatz von Cross-Plattform-Frameworks

Cross-Plattform-Frameworks wie Flutter haben gegenüber nativen Apps oft Nachteile in Bezug auf Performance und User Experience. Die UI-Performance kann schwach sein und das Dependency Management kann zu Versionskonflikten führen. Qualitativ hochwertige Widgets, die zuverlässig gewartet und aktualisiert werden, sind selten. Wenn eine Sprache noch sehr jung ist und nur in einem Nischenbereich eingesetzt wird, bedeutet dies einen höheren Einarbeitungsaufwand für die Entwickler. Eine lückenhafte Dokumentation kann die Entwicklung zusätzlich erschweren. Ein Framework sollte immer mit den aktuellen Betriebssystemversionen Schritt halten.

Plattformen

Eine Offline-fähige Desktop-Anwendung und die bestehende Web-Anwendung könnten in Zukunft auf derselben Code-Basis basieren. Dies könnte die Reichweite auf Desktop-Betriebssysteme wie Windows, macOS und Linux ausdehnen. mQuest-Online wurde u.a. als Reaktion auf Kundenbedürfnisse für Windows-Geräte entwickelt und könnte durchaus in eine Cross-Plattform-Strategie integriert werden.

5.2 Auswertung der Fragebögen

Der quantitative Charakter der Ergebnisse erlaubt eine grafische Darstellung. An den Befragungen haben 12 Personen teilgenommen. Die Auswertung der Antworten erfolgte mit dem Open Source Tool PSPP [109] und Microsoft Excel. Bei einem Teil der Fragen wurde eine 5-Punkte-Likert-Skala als Antwortschema verwendet. Für die grafische Auswertung dieser Fragen wurde ein frei zugängliches Online-Tool [110] verwendet.

Frage 1: Probleme mit der bisherigen App-Architektur

Gibt es in der bisherigen App-Architektur Probleme, die aus dem Prinzip „eine Code-Basis – mehrere Targets“ resultieren? Die Antwortmöglichkeiten sind aus Abbildung 5.5 zu entnehmen. Demnach sahen die meisten Befragten Probleme beim Debugging, gefolgt von mangelhafter Dokumentation und Inkonsistenzen in der UI. In den Freitextantworten wurden außerdem die folgenden Punkte genannt: Mehraufwand durch unpassende Typen im übersetzten Mobilecore-Code, vor allem beim Null-Handling. J2ObjC wird als limitierender Faktor angesehen. Persistenz und Sync müssen plattformspezifisch implementiert werden und es gibt funktionale Unterschiede zwischen den Plattformen.

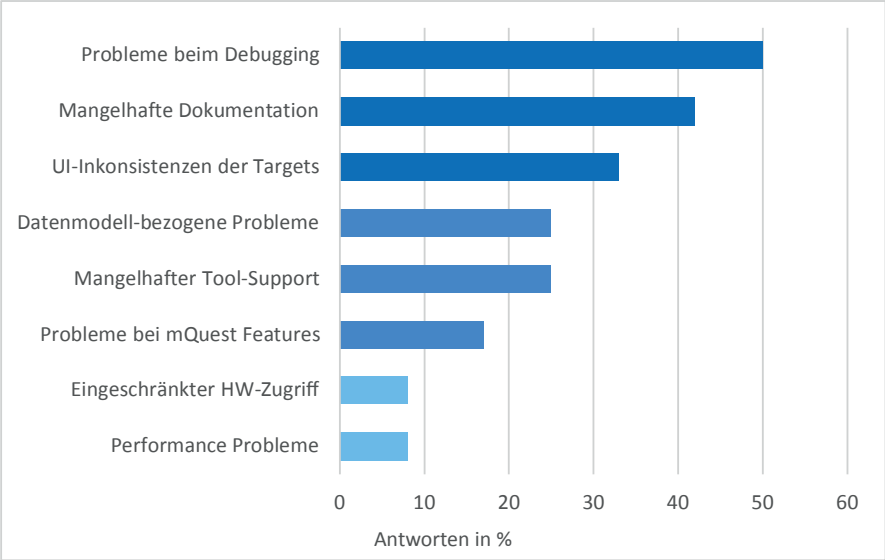

Abbildung 5.5: Frage 1 - Probleme mit der bisherigen App-Architektur

Frage 2: Lizenzmodell des Entwicklungs-Frameworks

Welche Rolle spielt das Lizenzmodell des Entwicklungs-Frameworks? Hier konnte zwischen folgenden Optionen gewählt werden:

- Nur Open Source
- Nur SW hinter der ein großer, renommierter Konzern steht, auch wenn Lizenzkosten entstehen
- Egal, beides möglich

Abbildung 5.6 zeigt, dass das Open-Source-Modell von einigen Personen unterstützt wird, während die Mehrheit hierzu kein explizite Meinung hat. Niemand sprach sich explizit für ein geschlossenes System eines renommierter Konzerns aus.

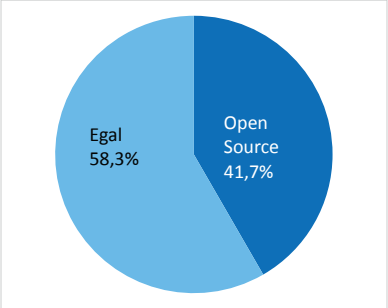

Abbildung 5.6: Frage 2 - Bevorzugtes Lizenzmodell des Entwicklungs-Frameworks

Frage 3: Bedeutung der Community

Wie wichtig wird die Bedeutung einer großen Community hinter einem plattformübergreifenden Framework eingeschätzt? Hierzu wurde eine 5-Punkt Likert-Skala verwendet. (1 bedeutet unwichtig, 5 bedeutet sehr wichtig). Abbildung 5.7 zeigt, dass die Community für alle Befragten eine essentielle Bedeutung hat.

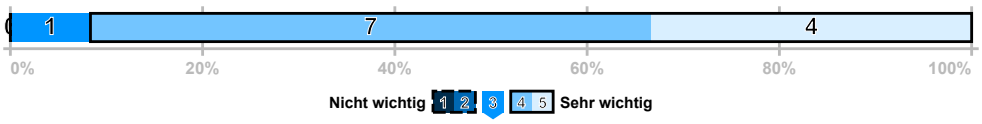

Abbildung 5.7: Frage 3 - Bedeutung der Community

Frage 4: UI-Konzept der neuen App

Auf welchem Konzept soll die Entwicklung des UI der neuen App basieren? Als Antwortmöglichkeiten standen zur Verfügung:

- Custom Design (auf beiden Plattformen möglichst ähnlich, einem firmeneigenen Corporate Design folgend)
- Nativ, je nach Plattform
- Egal

Abbildung 5.8 bestätigt, was sich bereits in den Experteninterviews andeutete: Es besteht eine starke Präferenz für eine native UI.

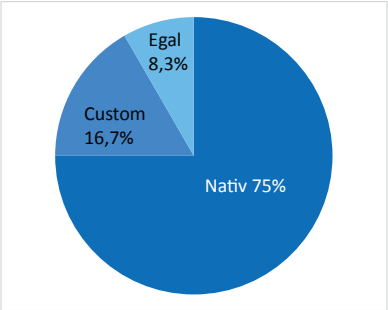

Abbildung 5.8: Frage 4 - UI-Konzept der neuen App

Frage 5: Bedeutung der Usability

Wie wichtig wird das Thema UX/Usability bei mobilen Apps generell eingeschätzt? Die Relevanz sollte auf einer Skala von 1 bis 5 bewertet werden. Das Thema ist für die Befragten von zentraler Bedeutung, wie die Auswertung dieser Frage in Abbildung 5.9 zeigt. Mehr als die Hälfte der Befragten haben hierfür den höchsten Wert angegeben, werden die beiden höchsten Skalenwerte zusammengenommen, ergibt sich eine Zustimmung von 83%.

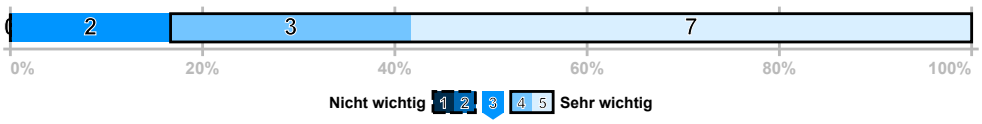

Abbildung 5.9: Frage 5 - Bedeutung der Usability

Frage 6: Einheitliche User Experience

Wie wichtig ist es aus strategischer Sicht für das Unternehmen, eine einheitliche User Experience über verschiedene Plattformen hinweg zu haben? Wie die Abbildung 5.10 veranschaulicht, hat dies für die Mehrheit der Befragten einen hohen Stellenwert.

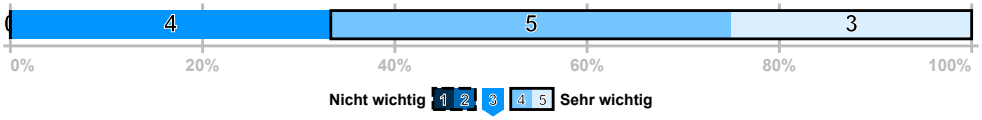

Abbildung 5.10: Frage 6 - Einheitliche User Experience

Frage 7: Performance der UI

Zur Usability gehört auch die Leistungsfähigkeit (Performance) der UI. Hier wurde nach der Einschätzung zu diesem Aspektes gefragt. Die Abbildung 5.11 lässt erkennen, dass dies den Befragten besonders wichtig erscheint.

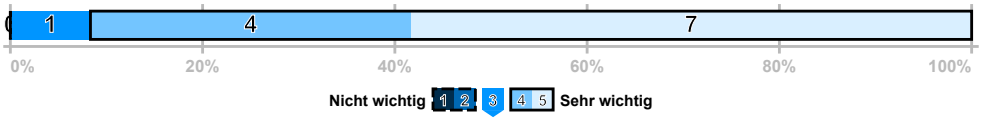

Abbildung 5.11: Frage 7 - Performance der UI

Frage 8: Bewertung plattformübergreifender Frameworks

Welche Kriterien haben bei der Bewertung eines plattformübergreifenden Frameworks das größte Gewicht? Es sollten die 3 wichtigsten ausgewählt werden. An erster Stelle steht der Reifegrad des Frameworks, gefolgt von der Qualität der Werkzeuge, die zur Effizienz des Entwicklers beiträgt, und der Qualität der Dokumentation. Siehe Abbildung 5.12 bezüglich weiterer Punkte. Zwei weitere Kriterien, die zur Auswahl standen, erhielten keine Stimmen: Dependency-Management und Hersteller-Support.

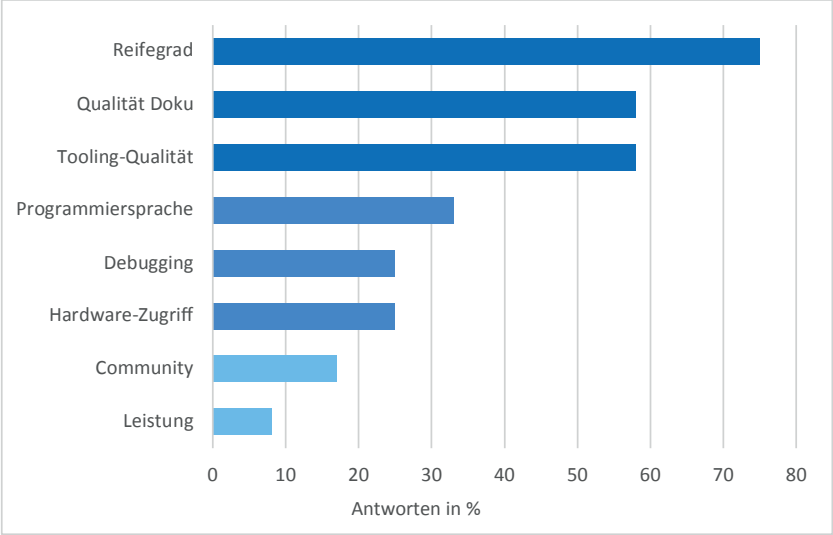

Abbildung 5.12: Frage 8 - Bewertung plattformübergreifender Frameworks

Frage 9: Erfahrungen mit Cross-Plattform Frameworks

Hier wurde gefragt: Hast Du bereits Erfahrungen mit einem Cross-Plattform Framework sammeln können? Wenn ja, mit welchem? Aus Abbildung 5.13 geht hervor, dass KMP und Flutter hier gleichauf liegen. React Native und andere JavaScript-basierte Frameworks kommen zusammen auf den selben Wert. Mit .NET MAUI oder Xamarin hat sich bei cluetec bisher noch niemand beschäftigt.

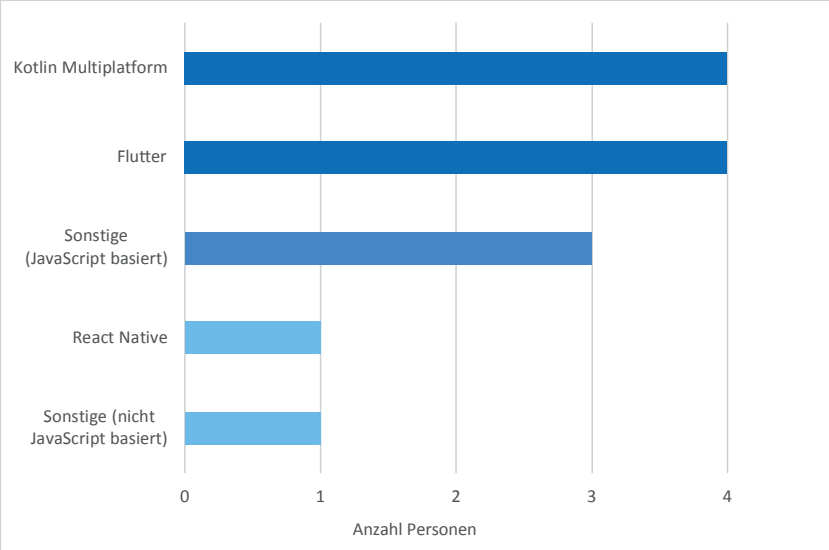

Abbildung 5.13: Frage 9 - Erfahrungen mit Cross-Plattform Frameworks

Frage 10: Bevorzugte Programmiersprachen

Mit welchen der folgenden Programmiersprachen/Technologien arbeitest Du am liebsten? Bei cluetec ist viel Erfahrung im Bereich Kotlin/Java vorhanden und wird auch gerne genutzt, wie Abbildung 5.14 zeigt. Zur Auswahl standen außerdem C# (.NET), XML/XAML und C/C++, diese erhielten jedoch keine Stimmen. Ein Blick auf die Zustimmungswerte für Swift/SwiftUI und Kotlin/Jetpack-Compose bestätigt, dass Entwickler gerne mit aktuellen Technologien arbeiten.

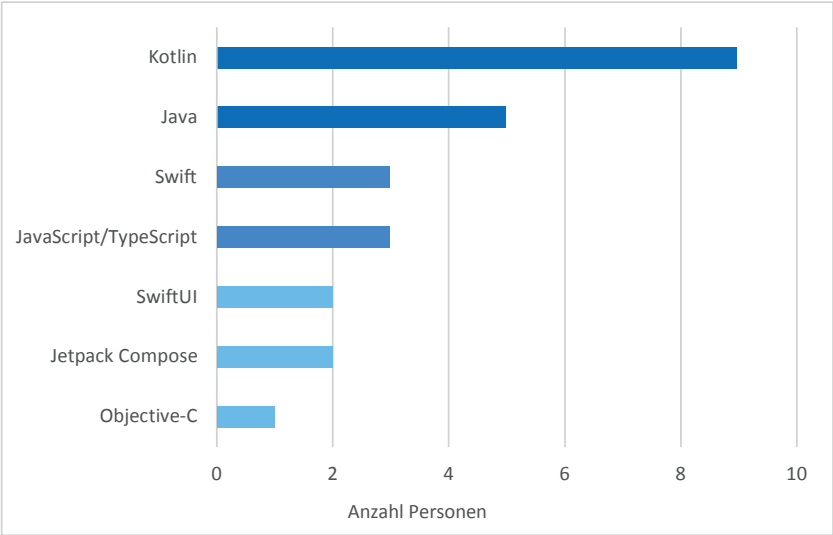

Abbildung 5.14: Frage 10 - Bevorzugte Programmiersprachen

Frage 11: Anforderungen an eine Programmiersprache

Welche Kriterien haben für Dich bei einer Programmiersprache das größte Gewicht? Auch hier sollten die 3 Wichtigsten ausgewählt werden. Die Umfrage ergab, dass zuverlässig gewartete Pakete und Bibliotheken die höchste Priorität haben. Die Sprache sollte zudem effizient sein, d.h. der Code sollte einfach zu programmieren, aber auch einfach zu warten sein. Eine mächtige und umfangreiche Standardbibliothek ist ebenfalls wichtig. Die weiteren Punkte sind aus Abbildung 5.15 zu ersehen.

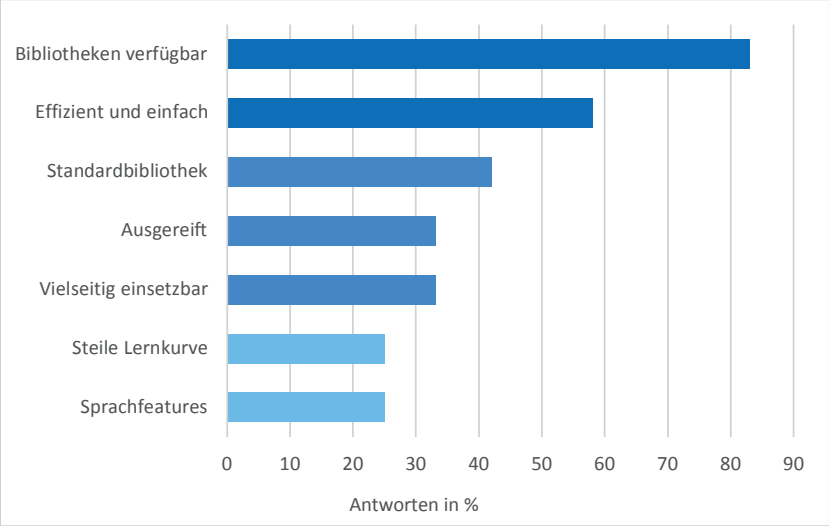

Abbildung 5.15: Frage 11 - Anforderungen an eine Programmiersprache

Frage 12: Transpiler als Alternative

Wäre aus Deiner Sicht auch eine Lösung denkbar, die (wie bisher bei cluetec) einen Transpiler beinhaltet? Hierfür gibt es kaum Zustimmung, wobei ein großer Teil der Befragten sich nicht festlegen möchte, wie aus Abbildung 5.16 zu erkennen ist.

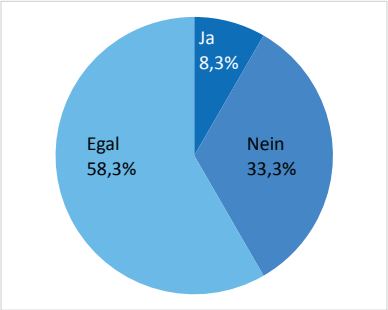

Abbildung 5.16: Frage 12 - Transpiler als Alternative

Frage 13: Zukünftige Teststrategie

Welche Tests müssen Deiner Ansicht nach auf jeden Fall in eine zukünftige Teststrategie mit einbezogen werden? Es konnte aus folgenden Antworten gewählt werden:

- Unit-Tests alleine sind ausreichend
- Unit-Tests und Integrationstests
- Zusätzlich zu Unit-Tests und Integrationstests sollten auch UI-Tests vorgesehen werden

Die Mehrheit der Befragten ist dafür, alle Tests (Unittests, UI-Tests und Integrationstests) durchzuführen, 5 von 7 Personen sind der Meinung, dass Unit- und Integrationstests ausreichen, siehe Abbildung 5.17.

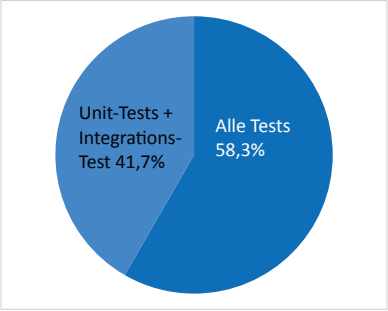

Abbildung 5.17: Frage 13 - Zukünftige Teststrategie

Frage 14: Neue Technologien

Welche neuen Technologien sollten Deiner Meinung nach in die Entwicklung der mQuest App einfließen? Nach dem Spitzenreiter Künstliche Intelligenz werden deklarative UI-Programmierung (siehe Kapitel 3.1.1) und VR-/AR-Technologien hoch bewertet, siehe Abbildung 5.18. Zur Auswahl stand zudem noch No-Code-Development, dies erhielt jedoch keine Stimme.

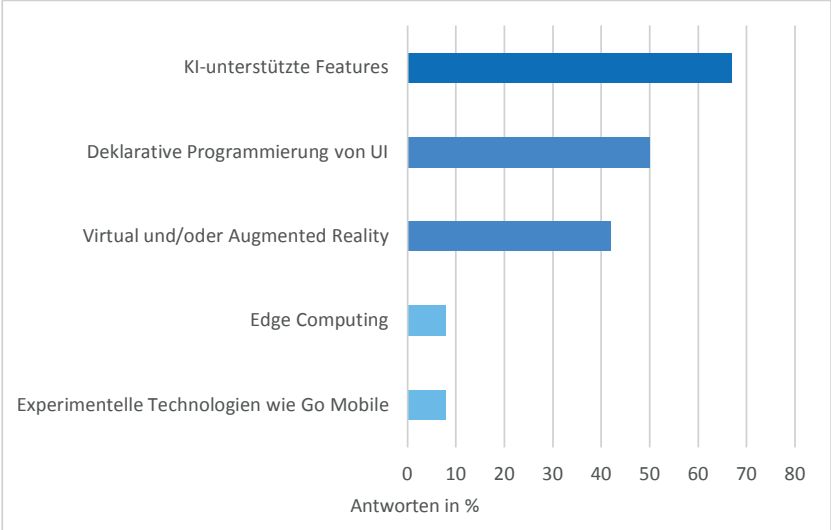

Abbildung 5.18: Frage 14 - Neue Technologien

Frage 15: Bevorzugte Entwicklungsumgebung

Welches sind Deine bevorzugten IDEs? Abbildung 5.19 zeigt, dass eine klare Präferenz für die IDEs der Fa. JetBrains besteht. Visual Studio von Microsoft erhielt keine Stimme.

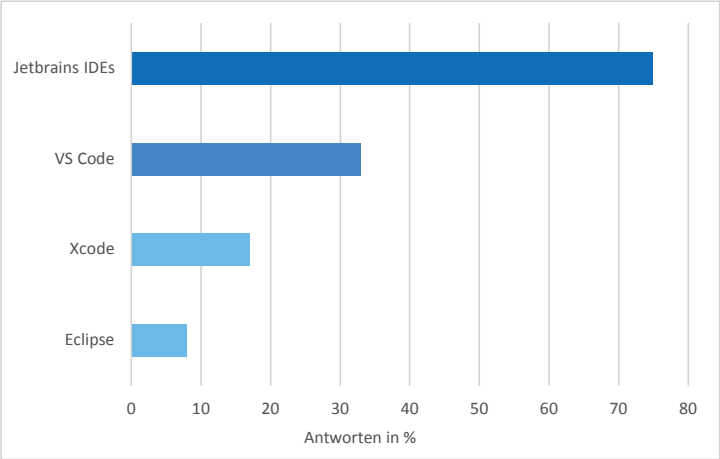

Abbildung 5.19: Frage 15 - Bevorzugte Entwicklungsumgebung

Frage 16: Feature Parity zwischen Plattformen

Die Frage: Wie wichtig ist das Thema Feature Parity zwischen den Plattformen? wurde mehrheitlich als wichtig eingestuft, siehe Abbildung 5.20.

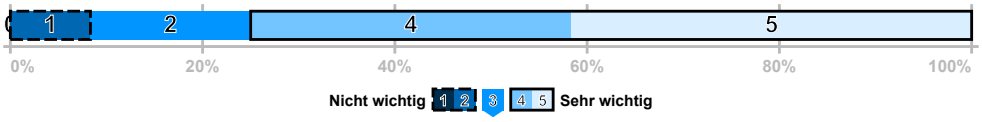

Abbildung 5.20: Frage 16 - Feature Parity

Frage 17: IT-Security

Welche Rolle spielt das Thema IT-Security? Dieser Frage wurde eine hohe Bedeutung beigemessen, wie aus Abbildung 5.21 ersichtlich ist.

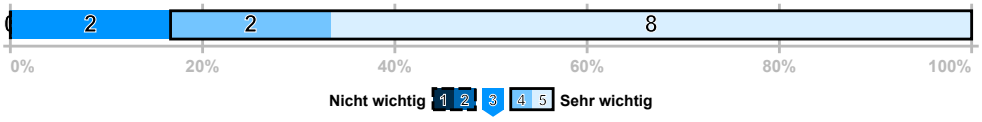

Abbildung 5.21: Frage 17 - IT-Security

Frage 18: Barrierefreiheit

Zuletzt wurde gefragt: Welche Rolle spielt Barrierefreiheit (Accessability)? Abbildung 5.22 zeigt, dass die Antworten zu dieser Frage überwiegend im neutralen Bereich liegen.

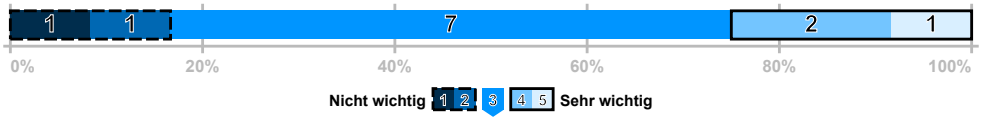

Abbildung 5.22: Frage 18 - Barrierefreiheit

In diesem Zusammenhang wurde auch gefragt, ob mQuest Audit vom Barrierefreiheitsstärkungsgesetz betroffen ist. Hierzu konnte jedoch keiner der Befragten eine Aussage treffen.

5.3 Evaluation der Prototypen-Entwicklung

Der Entwicklungsprozess der Prototypen wird nun anhand der definierten Kriterien bewertet. Die jeweiligen Punkte sind in der Entscheidungsmatrix im Kapitel 6.1 zu finden. Im Folgenden werden einige wichtige Kriterien und deren Bewertung in Bezug auf die jeweiligen Frameworks erläutert.

5.3.1 Programmiersprache

Die Entwicklung von Dart und Kotlin begann etwa zur gleichen Zeit und die Eigenschaften sind bis zu einem gewissen Grad vergleichbar. Beide Sprachen sind stark typisiert, typsicher und nullsicher.

Kotlin Coroutines Kotlin bietet umfassende Unterstützung für Coroutines, ein mächtiges Feature zur Verwaltung von Hintergrundaufgaben und asynchronem Code. Coroutinen sind nicht an einen Thread gebunden, sie können unterbrochen und auf einem anderen Thread fortgesetzt werden [111].

Asynchrone Operationen in Dart Dart bietet ebenfalls Unterstützung für asynchrone Operationen über spezielle APIs (z.B. Futures, Streams) und über Isolates. Letztere sind unabhängige Ausführungseinheiten, die gleichzeitig in einem separaten Speicherbereich ausgeführt werden. Sie eignen sich daher zur Parallelverarbeitung und zur besseren Ausnutzung von Mehrkernsystemen. Im Gegensatz zu Threads teilen sich Isolates keinen Speicher und vermeiden so Probleme mit gleichzeitigen Zugriffen [112].

Kotlin Scope Functions In Kotlin gibt es verschiedene Scope-Funktionen (let, apply, run, with, also) zur Ausführung von Code im Kontext eines Objekts, zur Unterstützung der Objektinitialisierung, zur bedingten Ausführung und für zusätzliche Effekte [113].

Kaskadennotation und Lambda-Funktionen in Dart Das Konzept der Scope-Funktionen ist in Dart nicht direkt enthalten. Die Kaskadennotation und die Lambda-Funktionen von Dart können jedoch verwendet werden, um ähnliche Ergebnisse zu erzielen. Die Kaskadennotation ermöglicht die Anwendung einer Folge von Operationen auf ein und dasselbe Objekt [114]. Lambda-Funktionen kapseln als anonyme Funktionen einen Codeblock [115]. Diese Funktionen in Dart können bis zu einem gewissen Grad das Verhalten von Kotlins Scope-Funktionen imitieren und ermöglichen einen prägnanteren und besser lesbaren Code.

Data Classes in Kotlin Dies sind spezielle Klassen zum Speichern von Daten. Data Classes verfügen unter anderem über eine vom Compiler abgeleitete copy()-Funktion, die das Erstellen neuer Instanzen mit der Möglichkeit des teilweisen Überschreibens einiger ihrer Eigenschaften vereinfacht [116].

Records in Dart In Dart bieten Records eine einfache Möglichkeit, unveränderliche datenhaltende Klassen zu deklarieren. Für diese gibt es jedoch keinen Copy Constructor [117].

Pattern Matching Kotlin bietet eine aussagekräftige und prägnante Möglichkeit, konditionale Logik zu handhaben, insbesondere mit dem when-Ausdruck [118]. Dart hat eine ähnliche Funktionalität und verwendet dafür das switch-case Konstrukt [119].

Zusammenfassend kann gesagt werden, dass Kotlin im Vergleich zu Dart Stärken vor allem im Bereich der Coroutinen und der Scope Functions hat.

5.3.2 Dependency-Management

KMP verwendet Gradle für das Dependency Management, das Tool ist auch für die Build-Automatisierung und die Konfiguration eines Projekts zuständig. Es automatisiert das Kompilieren, Paketieren und Testen von Code und verwaltet Abhängigkeiten, um den Entwicklungsprozess zu vereinfachen.

Gradle ist flexibel und mächtig für verschiedene Build-Szenarien, aber es ist komplex und kann überwältigend zu konfigurieren sein, vor allem für Entwickler, die noch wenig Erfahrung mit dem Tool haben. Aufgrund der umfangreicheren Prozesse kommt es oft zu längeren Buildzeiten. Die Pakete kommen aus verschiedenen Quellen und teilweise muss die Paketversion mit den Versionen von Kotlin und Compose koordiniert werden, da es sonst zu Konflikten kommt.

In Flutter wird der Befehl `flutter pub get` verwendet, um alle Abhängigkeiten eines Projekts herunterzuladen und zu installieren. Das Tool ist einfach zu bedienen. Es ist speziell für Flutter Projekte optimiert und garantiert Kompatibilität und hohe Performance. Unter pub.dev stehen die verfügbaren Pakete in einem offiziellen Repository zur Verfügung [120].

5.3.3 Tooling

Für die Entwicklung von Flutter Projekten gibt es verschiedene Möglichkeiten. Als IDE kann Visual Studio Code mit einer speziellen Flutter Extension verwendet werden. Diese Erweiterung bietet Unterstützung für effektives Editieren, Refactoring, Ausführen

und Debuggen. Die Unterstützung für die Programmiersprache Dart wird automatisch mitinstalliert. Darüber hinaus können auch die IDEs der Firma JetBrains verwendet werden, z.B. Android Studio oder IntelliJ IDEA. Auch für diese gibt es ein Flutter Plugin, welches eine spezielle Unterstützung für das Erstellen und Deployen von Flutter Apps und die Dart Sprachunterstützung mitbringt. Prinzipiell kann eine Flutter App aber auch in jedem anderen Texteditor geschrieben werden, da die Flutter CLI eine vollständige Unterstützung für das Kompilieren und Debuggen auf der Kommandozeile bietet.

JetBrains gibt an, dass KMP-Projekte in jeder IDE erstellt werden können, die Kotlin unterstützt, empfiehlt aber in erster Linie ihre Produkte Fleet und Android Studio. Für letzteres gibt es ein Plugin.

Sowohl für Flutter als auch für KMP gilt: Wenn man die App auch für Apple-Plattformen bereitstellen möchte, benötigt man zusätzlich eine Xcode-Installation auf dem Entwicklungsrechner.

Bei der Entwicklung mit Flutter kann ein Open-Source-Paket namens *Widgetbook* eingesetzt werden [121]. Mit Widgetbook kann man seine eigenen Widgets in einer zentralen Widget-Bibliothek katalogisieren. Außerdem können Widgets mit Figma synchronisiert und mit Teammitgliedern geteilt werden, um Frontend-Reviews zu vereinfachen. Dies kann in einen CI/CD-Prozess eingebunden werden.

Auch für KMP gibt es zahlreiche Tools und Erweiterungen, die die Arbeit erleichtern und die Effizienz steigern können. Einen guten Überblick bietet das Git Repository *Awesome Kotlin Multiplatform* [122].

5.3.4 Performance, Effizienz

Wie im Kapitel 5.3.2 schon ausgeführt, hat KMP einige Defizite bei der Performance und der Effizienz. Gradle Tasks dauern generell recht lange, und durch das Bauen der Bibliotheken für die Frontend-Projekte fallen hier zusätzliche Aufwände an. JetBrains arbeitet an einem Tool namens *Amper*, das zwar im Hintergrund Gradle verwendet, aber zukünftig zumindest die Konfiguration eines Projektes durch eine simplere Syntax stark vereinfachen soll [123].

Stateful Hot Reload und Hot Restart sind herausragende Entwicklungsfunktionen in Flutter. Gerade der Hot Reload ermöglicht es Entwicklern, die Auswirkungen von Codeänderungen sofort im Simulator oder auf einem physischen Gerät zu sehen, ohne dass der State der UI zurückgesetzt wird. Befindet man sich beispielsweise in einem Zustand, der eine Navigation zu einer Unterseite und eine Dateneingabe erfordert, muss man diese Schritte nach einem Hot Reload nicht erneut ausführen.

In KMP-Projekten gibt es bei der UI-Entwicklung mit Android Studio und Xcode kein Äquivalent zu Stateful Hot Reload. Änderungen erfordern typischerweise ein Rebuild und ein erneutes Laden auf das Gerät oder den Simulator, was mehr Zeit in Anspruch nehmen kann. Außerdem geht der State der UI verloren. Beide Entwicklungsumgebungen verfügen jedoch mittlerweile über sehr ausgefeilte Vorschaufunktionen, die das Entwickeln der UI ebenfalls erleichtern.

5.3.5 Reifegrad des Frameworks

Die Arbeiten an den beiden Frameworks Flutter und KMP begannen im selben Jahr. Die erste Erwähnung von Flutter findet sich im Mai 2017 auf GitHub.com [124]. Im November desselben Jahres wird die Version 1.2 von Kotlin veröffentlicht, in den Release Notes wird Multiplattform als experimentelles Feature erwähnt [125]. Die Entwicklung der zugrundeliegenden Sprachen Dart und Kotlin begann, ebenfalls fast zeitgleich, im Jahr 2011 [40], [47]. Danach verlief die Entwicklung bis zur Veröffentlichung einer Stable-Version jedoch nicht mehr parallel.

Im Dezember 2018 verkündete Google auf der Flutter Live in London [126] mit der ersten *stable* Version 1.0 die „General availability“ von Flutter [127]. Flutter kann heute als äußerst erfolgreich bezeichnet werden. Zum Beginn des Jahres 2023 gab es mehr als 700.000 Apps im Google Play Store, die mit Flutter erstellt wurden [128]. Die Gesamtzahl der Apps im Play Store betrug zu diesem Zeitpunkt ca. 2,7 Millionen [129]. Damit hat Flutter einen Anteil von ca. 26%.

Bei der Entwicklungsarbeit mit Flutter fällt auf, dass sich das Framework seit mehreren Jahren in einem freigegebenen Status befindet. Die Dokumentation ist größtenteils gut gepflegt und aktuell. Die angebotenen Widgets und Pakete laufen stabil.

KMP wurde von JetBrains während der Abfassung dieser Arbeit, am 1. November 2023 für „Stable and Production-Ready“ erklärt [130]. Dies bedeutet, dass KMP bis kurz vor der Bekanntgabe noch signifikanten Änderungen unterlag, was verschiedene Schwierigkeiten mit sich bringt. Die Dokumentation ist teilweise nicht aktuell. Codebeispiele aus Quellen, die nur wenige Monate alt sind, funktionieren möglicherweise aufgrund von *breaking changes* nicht mehr.

Der Kern von KMP ist seit Kotlin 1.9.20 Stable. JetBrains wird diese Teile nur nach strengen Kompatibilitätsregeln weiterentwickeln [130]. Allerdings erfordern viele Drittanbieter-Bibliotheken immer noch ein *Opt-in* für experimentelle Features. Solche Bibliotheken können sich in Zukunft so gravierend ändern, dass darauf basierender Code nicht mehr funktioniert. Dies gilt auch für Bibliotheken aus dem Compose Multiplatform Projekt, das noch keinen Stable Status erreicht hat.

5.3.6 Dokumentation

Für die Dokumentation gibt es bei Flutter eine zentrale Anlaufstelle [131] mit vielen Bereichen, u.a. einem Katalog aller verfügbaren Widgets, der API-Dokumentation und Code-Beispielen. Auf YouTube gibt es viele offizielle Erklärvideos, wie z.B. die Serie „Flutter Widget of the Week“ mit mehr als 150 kurzen Videos (ca. 2 Minuten), mit deren Hilfe man sich schnell über die Eigenschaften und die Art der Nutzung informieren kann. In der Videobeschreibung befindet sich ein Link, der wiederum direkt zur entsprechenden API-Beschreibung führt. Diese enthält stets Code-Beispiele, die eine schnelle Orientierung ermöglichen.

Die Dokumentation zu KMP ist auf der offiziellen Kotlin-Webseite zu finden [132]. Im Gegensatz zu einem geschlossenen Ökosystem wie Flutter erfordert die Entwicklung eines KMP-Projekts jedoch das Einholen von Informationen aus verschiedenen Quellen. An vorderster Stelle steht dabei die offizielle Kotlin-Sprachdokumentation [133]. Zusätzlich wird man häufig auf Bibliotheken stoßen, die spezifische Ausprägungen für die Entwicklung in Kotlin bieten. Ein Beispiel hierfür ist die No-SQL Datenbank Realm, deren Dokumentation unter [134] verfügbar ist.

Bei der Entwicklung der UIs in den jeweiligen nativen Frameworks (z.B. SwiftUI für iOS) muss auch auf diese Dokumentation zurückgegriffen werden. Die Dokumentation dieser beiden UI-Frameworks ist im Allgemeinen nicht so zugänglich wie die Sprachdokumentation von Kotlin oder die Flutter-Dokumentation. Sie enthalten zum Beispiel weniger umfangreiche Codebeispiele. Daher ist man in der Einarbeitungsphase stärker auf Quellen wie einschlägige Entwicklerforen und Erklärvideos angewiesen.

5.3.7 Modularität

Wenn ein neues KMP-Projekt erstellt wird, hat es eine integrierte Projektstruktur. Das bedeutet, dass die Ordnerstruktur einen Ordner für den gemeinsamen Code enthält, in dem es einen Unterordner commonMain und jeweils einen Ordner für die Zielplattform gibt, also iosMain und androidMain. Zusätzlich gibt es auf der obersten Ebene einen Ordner für die iOS App und einen für die Android App. Diese Struktur würde sich in Git in einem sogenannten Mono-Repository widerspiegeln.

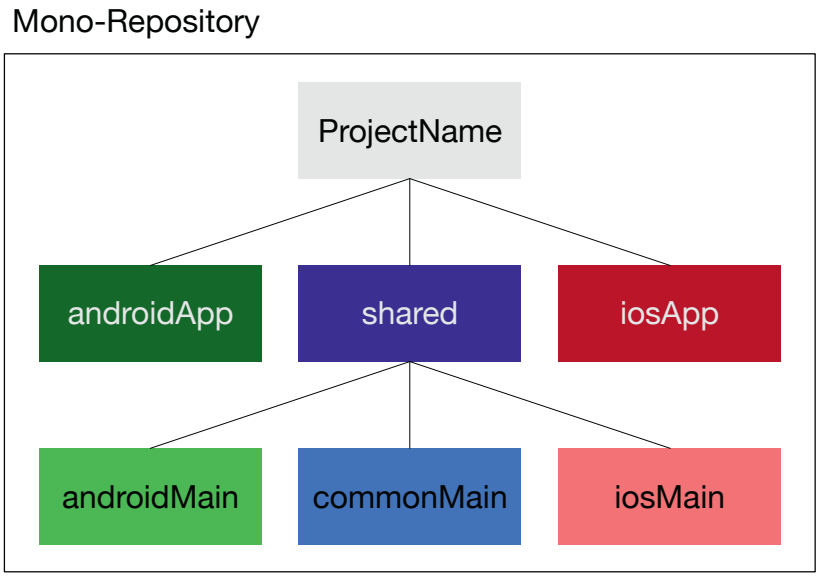

Abbildung 5.23: Mono-Repository

Wird hingegen ein Multi-Repository-Ansatz bevorzugt, kann für die gemeinsame Geschäftslogik eine KMP-Bibliothek erstellt werden. Diese wird dann in das jeweilige plattformspezifische Projekt (Xcode bzw. Android) integriert. Für Android als AAR Library (z.B. über Maven Central oder Local) und für iOS als XCFramework. Dieser Ansatz erleichtert das kollaborative Arbeiten.

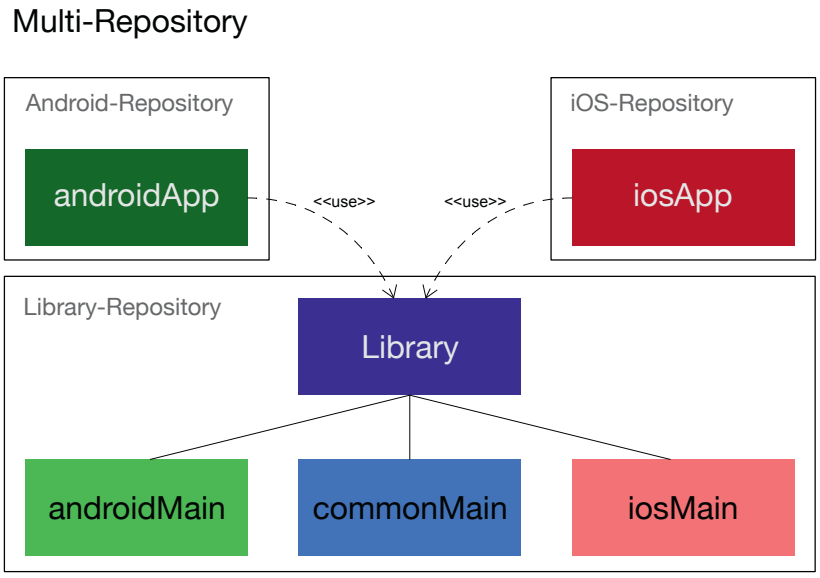

Abbildung 5.24: Multi-Repository

Flutter besitzt keine Maven-Integration. Möchte man die Logik auslagern, so kann dies über ein Paket (Package) realisiert werden. Pakete werden normalerweise auf pub.dev [120] veröffentlicht. Soll eine proprietäre Geschäftslogik nicht veröffentlicht werden, kann das Package auch in einem firmeneigenen Git Repository abgelegt und über die `pubspec.yaml` referenziert werden [135].

5.3.8 Code-Wiederverwendbarkeit

Flutter verfolgt einen klassischen Cross-Plattform-Ansatz. Aus einer gemeinsamen Codebasis können Anwendungen für alle Zielplattformen generiert werden. Allerdings muss bei Flutter mit zusätzlichem Aufwand gerechnet werden, wenn die UI nativ aussehen soll, siehe Kapitel 5.3.10.

Zum heutigen Zeitpunkt wird bei KMP die UI jeweils nativ programmiert[8]. Hierdurch ist die Menge des gemeinsam genutztem Codes etwas geringer. Zentraler Aspekt des gemeinsamen Codes ist ein Mechanismus, der mit *expect-actual* bezeichnet wird. In manchen

8 JetBrains arbeitet bereits an Compose Multiplatform, um auch die UI aus gemeinsam genutztem Code zu generieren (siehe Kapitel 3.1.2). Allerdings befindet sich das Projekt noch in einem frühen Stadium, die bisherigen Demoprojekte zeigen eine UI ohne plattformspezifische Ausprägung.

Fällen müssen Funktionalitäten plattformspezifisch implementiert werden. Für solche Funktionalitäten werden in KMP Interfaces geschrieben, die mit dem Schlüsselwort expect gekennzeichnet sind. KMP weiß, dass es für die im Projekt verwendeten Plattformen immer eine plattformspezifische Implementierung geben muss, die mit actual gekennzeichnet ist. Diese wird beim Kompilieren automatisch für die jeweiligen Zielplattformen verwendet [136, S. 15].

Unter Berücksichtigung dieser Aspekte lohnt sich ein KMP-Projekt umso mehr, je komplexer und umfangreicher die Programmlogik ist.

Die Entscheidung, welche Codeteile dem gemeinsamen Bereich (commonMain) zugeordnet werden, ist nicht trivial. Wenn ein erheblicher Teil des Codes dennoch eine plattformspezifische Implementierung über den expect-actual-Mechanismus erfordert, kann dies die erwartete Zeitersparnis reduzieren. Insbesondere für die Erstellung von actual-Implementierungen auf iOS kann spezifisches Fachwissen oder eine Einarbeitung in Objective-C erforderlich sein.

5.3.9 Weitere Plattformen

Die beiden betrachteten Entwicklungs-Frameworks können neben Android und iOS noch weitere Plattformen bedienen, siehe [137] und [138].

KMP läuft auf Desktop- und Serversystemen (Windows, macOS, Linux) mittels einer **JVM**. Für Webanwendungen unterstützt KMP zwei Technologien: Einerseits kann Kotlin-Code in JavaScript übersetzt werden, was bereits als Stable eingestuft wurde [139], andererseits befindet sich die Unterstützung für Wasm noch im Alpha-Stadium.

Beim Projekt „Compose Multiplatform UI Framework“ ist zu beachten, dass die Unterstützung für Android und Desktop bereits Stable ist, während sich iOS und Wasm noch im Alpha-Stadium befinden [137].

Flutter unterstützt auch Desktop-Anwendungen und bietet vollen Zugriff auf die zugrundeliegenden Win32-, Cocoa- oder UNIX-Plattform-APIs [140]. Für das Web arbeitet Flutter mit einer Kombination aus DOM, Canvas und WebAssembly und zielt damit vor allem auf Progressive Web Apps (**PWA**) und Single Page Applications (**SPA**). Flutter gibt außerdem an, auch auf Embedded Devices [138] lauffähig zu sein.

5.3.10 Plattformspezifische UI

Dies ist bei KMP-Projekten inhärent gegeben.

Plattformspezifische UI in Flutter

In einem Flutter-Projekt kann die UI aus ein und derselben Codebasis erstellt werden. Wird dieser Ansatz konsequent verfolgt, sieht das Ergebnis auf jeder Zielplattform nahezu gleich aus. Für eine plattformspezifische UI stehen für Android die Material Widgets [141] und für iOS die Cupertino Widgets [142] zur Verfügung. Bei neuen Projekten ist Material Design voreingestellt, das CupertinoApp Widget muss manuell erstellt werden. Mit Hilfe der Platform-Klasse kann ermittelt werden, auf welchem Gerät die App ausgeführt wird. Davon abhängig kann das entsprechende Widget aufgerufen werden, siehe Code-Listing 5.1.

```
1  void main() {
2      runApp(Platform.isIOS
3      ? const MQAppCupertino()
4      : const MQAppMaterial());
5  }
6
7  class MQAppMaterial extends StatelessWidget {
8      const MQAppMaterial({super.key});
9
10     @override
11     Widget build(BuildContext context) {
12         return MaterialApp(...);
13     }
14 }
15
16 class MQAppCupertino extends StatelessWidget {
17     const MQAppCupertino({super.key});
18
19     @override
20     Widget build(BuildContext context) {
21         return CupertinoApp(...);
22     }
23 }
```

Codefragment 5.1: Aufruf plattformspezifischer UI-Widgets

Dies setzt sich in den darunterliegenden Widgets fort, immer wenn ein Widget in einer plattformspezifischen Implementierung vorhanden ist, muss diese Plattformabfrage stattfinden. Dies betrifft sehr viele UI-Elemente, von Buttons über Context Menus bis hin zu Page Transitions.

Nicht alle Designelemente, die aus der nativen iOS-Entwicklung bekannt sind, stehen als Cupertino Widgets zur Verfügung. Ein Beispiel: Es gibt zwar das *CupertinoTabScaffold*, mit dem eine klassische Tabbed Navigation erstellt werden kann, um jedoch anstelle der Tab-Leiste eine Leiste mit Buttons zum Starten von Aktionen zu erhalten, wie sie sich z.B. in Apples Notizen und Mail Apps finden, muss ein eigenes Widget erstellt werden, siehe Abbildung 5.25.

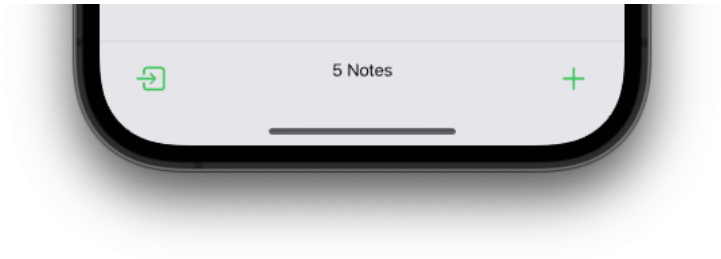

Abbildung 5.25: Selbsterstelltes BottomBar-Widget für iOS

Für die native iOS-Entwicklung steht mit SFSymbols eine große Anzahl von Icons zur Verwendung in Xcode zur Verfügung. In Flutter steht mit CupertinoIcons eine ähnliche, aber nicht so umfangreiche Icon-Bibliothek zur Verfügung [143]. Für die Verwendung in Android/Material Apps empfiehlt sich die Verwendung der Material Icons [144].

Flutter ermöglicht das Erstellen eigener Widgets, indem mehrere existierende Widgets in einer Klasse zusammengefasst werden. So können beispielsweise Widgets entwickelt werden, die sowohl in Android UIs als auch in iOS UIs verwendet werden können. Die plattformspezifische Anpassung erfolgt direkt innerhalb des Widgets, wobei die notwendigen Assets für beide UI-Designs zur Verfügung gestellt werden. Code-Listing 5.2 zeigt dies am Beispiel eines benutzerdefinierten Buttons, der sowohl ein Icon als auch Text enthält.

```
1   class ButtonIconText extends StatelessWidget {
2     final IconData cupertinoIcon;
3     final IconData materialIcon;
4     final String buttonText;
5     final Color? backgroundColor;
6     final Color? textColor;
7     final VoidCallback? onPressed;
8
9     const ButtonIconText({super.key, // weitere Properties});
10
11    @override
12    Widget build(BuildContext context) {
13      return Platform.isIOS
```

```
14            ? CupertinoButton.filled(
15                onPressed: onPressed,
16                padding: const EdgeInsets.all(16),
17                child: Row(
18                  mainAxisAlignment: MainAxisAlignment.start,
19                  children: [
20                    Icon(
21                      cupertinoIcon,
22                      color: textColor,
23                    ),
24                    const SizedBox(width: 16),
25                    Text(
26                      buttonText,
27                      style: TextStyle(
28                        color: textColor,
29                      ),
30                    ),
31                  ],
32                ),
33              )
34            : ElevatedButton(
35                // Material Design Implementierung
36              );
37      }
38    }
```

Codefragment 5.2: Flutter: Custom Widget

Plattformspezifisches UI Verhalten in Flutter

Die UI von Android- und iOS-Geräten unterscheidet sich nicht nur im Aussehen der Bedienelemente. Es gibt auch Unterschiede im Verhalten, z.B. bei der Textbearbeitung und beim Scrollen. Diese Elemente würden als „falsch“ empfunden, wenn sie sich anders verhalten würden. Aus diesem Grund hat Flutter automatische Plattformanpassungen entwickelt [145]. Diese betreffen unter anderem die Navigation und die damit verbundenen Übergänge sowie das Scrollverhalten. Auch die nur unter iOS verfügbare Funktion, durch Tippen auf die Statusleiste an den Anfang einer Liste zu scrollen, wurde implementiert. Durch die Verwendung der App-Pakete Material oder Cupertino werden zudem Typografie und Ikonografie entsprechend ausgewählt und das haptische Feedback angepasst.

5.3.11 Plattformspezifischen Code integrieren

In seltenen Fällen kann es auch in Cross-Plattform-Projekten notwendig sein, nativen Code zu integrieren. In einem KMP Projekt ist diese Funktionalität einerseits durch

den expect-actual Mechanismus im gemeinsamen Code gegeben. Andererseits kann diese Funktionalität auch in das native App-Projekt integriert werden, auch wenn dadurch die Trennung zwischen Logik und UI möglicherweise nicht mehr vollständig aufrechterhalten werden kann.

In einem Flutter Projekt stehen dafür die *Platform Channels* zur Verfügung [146]. Diese Channels dienen dem Informationsaustausch zwischen dem Dart Code und dem jeweiligen nativen Code. Es gibt Channels für Java, Kotlin, Objective-C, Swift, C und C++. Das Beispiel in der Flutter Dokumentation zeigt die Abfrage des Batteriezustands. Andy Julow beschreibt einen weiteren Fall, in dem eine plattformspezifische Implementierung erforderlich ist, nämlich für den Zugriff auf die Sensorwerte des Barometers, das heute in modernen Smartphones üblicherweise verbaut ist [147].

5.3.12 Ressourcen, Lokalisierung

Es ist wünschenswert, Ressourcen wie lokalisierte Texte, Grafiken und Bilder, aber auch Fonts und Farbdefinitionen nur einmal pflegen zu müssen.

Unter Lokalisierung versteht man bei der Entwicklung mobiler Apps den Prozess der Anpassung einer App an die sprachlichen, kulturellen und sonstigen Anforderungen eines bestimmten Zielmarktes oder -gebiets. Dazu gehört vor allem die Bereitstellung der verwendeten Texte in verschiedenen Sprachen.

In den Prototypen sollten die Texte in Deutsch und Englisch hinterlegt werden. Beim Bootstrappen von Projekten kann die Mehrsprachigkeit jedoch nicht direkt angelegt werden, hierfür sind weitere Schritte notwendig.

Für Flutter sind die folgenden Schritte durchzuführen:

- Pakete als Dependencies in pubspec.xml einfügen (`flutter_localizations` und `intl`)
- `flutter_localizations` importieren
- localizationsDelegates und supportedLocales spezifizieren
- Sprachdateien (ARB = App Resource Bundle) anlegen
- Aus den ARB-Dateien werden Dart-Code-Dateien generiert
- Diese Dart-Dateien importieren
- Nun können die AppLocalizations überall verwendet werden

In den ARB-Dateien kann mit Platzhaltern für Variablen gearbeitet werden und es können unterschiedliche Wörter für Singular und Plural definiert werden. Es können auch Auswahlmöglichkeiten erstellt werden, beispielsweise um Begriffe geschlechtergerecht zu formulieren [148].

In einem KMP-Projekt ist es empfehlenswert, Ressourcen im gemeinsamen Code vorzuhalten. KMP bietet hierfür keinen eigenen Mechanismus, es muss auf externe Bibliotheken zurückgegriffen werden. Ein sehr bekannter Anbieter von KMP-Bibliotheken, IceRock [149], bietet hierfür Moko-Resources [150] an. Das Verfahren für die Bereitstellung von lokalisierten Texten umfasst laut Dokumentation die folgenden Schritte:

- Dependency Setup in Gradle
- Setup der gewünschten Sprachen in Xcode
- Erstellung von XML Dateien mit den übersetzen Texten
- Ausführen des `generateMRcommonMain` Gradle-Task
- Manuelle Erstellung von Gettern für die `Strings` in commonMain (optional, reduziert aber die Komplexität der Aufrufe im UI-Code)
- Verwendung der Text-Ressource in der nativen UI
  Bsp. iOS: `let string = getMyString().localized()`

Die Bibliothek bietet auch Unterstützung für Farben, Schriften, Dateien und grafische Ressourcen.

Die aktuelle Version von Moko-Resources ist jedoch nicht kompatibel mit der neuesten Version von Kotlin (d.h. mit der Version, die bei KMP als stable veröffentlicht wurde), und die Alpha-Version, die diese Kompatibilität herstellen soll, konnte nicht zum Laufen gebracht werden. Im Prototyp wurde daher *Libres* verwendet [151]. Libres kommt mit einem etwas einfacheren Setup aus, unterstützt aber keine Farben, Schriften und Dateien. Es ermöglicht die gemeinsame Nutzung von Strings und grafischen Ressourcen, wobei letzteres die Verwendung von CocoaPods [152] im Xcode-Projekt voraussetzt.

5.3.13 Testing

In Flutter gibt es 3 Arten von Tests: Unit Tests, Widgettests und Integrationstests. Widgettests sind UI-Tests und eine Spezialität von Flutter. Sie erlauben es, die UI auf einer granularen Ebene zu testen, indem die Funktion einzelner Widgets überprüft wird. Diese Tests sind jedoch nur für Widgets sinnvoll, die bei Interaktion ihren Zustand ändern, z.B. ein Testfeld, das die Eingabe validiert und daraufhin die Farbe wechselt.

Integrationstests sind naturgemäß aufwendig, aber in Flutter mit überschaubarem Aufwand zu erstellen. Diese Tests können mit dem Befehl `flutter drive` im Simulator oder auf realen Geräten durchgeführt werden. Dies hat den Vorteil, dass z.B. Datenbanken nicht gemockt oder gefaked werden müssen und der Test dadurch umso realistischer ist.

Unit-Tests für KMP-Projekte lassen sich in zwei Kategorien einteilen: Zum einen die Tests für den gemeinsamen Code, die auf jeder Plattform mit jedem unterstützten Framework ausgeführt werden können. Zum anderen die Tests für den plattformspezifischen Code, für die plattformspezifische Testframeworks verwendet werden. Wenn also der expect-actual-Mechanismus verwendet wurde, um plattformspezifische Funktionen zu programmieren, können auch separate Tests erforderlich sein.

Für UI-Tests und Integrationstests muss dann auf die jeweiligen Testframeworks und Methoden zurückgegriffen werden, die für die Zielplattformen bereitstehen.

Für CI/CD Workflows gilt: KMP ist etwas aufwändiger in der Einrichtung, da neben den Pipelines für die Apps auch eine für den gemeinsamen Code aufgesetzt werden muss. Dies ist jedoch ein einmaliger Aufwand.

5.3.14 Debugging

Für das Debugging stehen in den IDEs die üblichen Werkzeuge wie Breakpoints und schrittweise Ausführung zur Verfügung. In Flutter gibt es zusätzlich einen Widget Inspector, um ein spezifisches UI-Debugging durchzuführen. Damit können Inkonsistenzen aufgedeckt werden, die durch die Verschachtelung von Widgets entstehen können, wie z.B. Überläufe durch zu breit oder zu hoch gewählte Child-Elemente.

Das Debugging in Flutter bezieht sich hauptsächlich auf den Dart-Code, der für alle Plattformen gleich ist. Daher ist die Erfahrung beim Debuggen auf verschiedenen Plattformen ziemlich einheitlich.

In KMP kann das Debugging komplexer sein, da es nicht nur den gemeinsamen Kotlin-Code, sondern auch die plattformspezifischen Implementierungen umfasst. Dies kann unterschiedliche Debugging-Ansätze oder die Verwendung mehrerer Werkzeuge erfordern.

5.3.15 Resultat

Die erstellten Apps konnten alle Anforderungen erfüllen. Da bei KMP die UI in den jeweiligen nativen Frameworks entwickelt wurden, hat der Nutzer eine vollständig native User Experience. Aber auch bei Flutter war es möglich, unter Verwendung von plattformspezifischen Widgets einen nativen Eindruck zu erhalten.

Die erforderlichen Features konnten ebenfalls mit beiden Frameworks vollständig realisiert werden.

5.3.16 Deployment

Die mQuest App wird auch mit kundenspezifischem Branding angeboten. Dies erfordert einen flexiblen Deployment-Prozess, der Android Product Flavors und Xcode Build Configurations verwendet. Bei KMP können diese Technologien beibehalten werden.

Für Flutter stehen verschiedene Optionen zur Verfügung. Im Flutter Framework gibt es auch Flavors [153]. Laut dem Podcast-Team der „programmier.bar“ stößt man damit aber schnell an seine Grenzen [154, 0:25:50]. Ihre in Flutter geschriebene App „4 Bilder 1 Wort“ wird in 8 Sprachen für 2 Plattformen veröffentlicht, so dass bei jedem Build-Prozess 16 App-Varianten entstehen[9]. Dazu verwenden sie nach eigenen Angaben ein Dart-Paket namens dart_define [155].

In der oben genannten Podcast-Episode wird auch das Kommandozeilen-Tool Fasttrack erwähnt [154, 0:28:17]. Dieses Tool vereinfacht die Erstellung von App-Versionen, die Einstellung der Release-Eigenschaften und die Veröffentlichung im Google Play Store oder Apple App Store [156].

9 In den meisten Fällen wird man nicht für jede Sprache eine eigene Build-Variante und damit ein eigenes Binary verwenden, sondern die Möglichkeiten der Lokalisierung nutzen (wie in Kapitel 5.3.12 beschrieben). In diesem Fall wurde diese Entscheidung getroffen, die Gründe dafür werden im Podcast nicht explizit genannt.

6 Schlussbetrachtung

Zum Abschluss der Arbeit sollen die diskutierten Konzepte, Technologien und Herausforderungen reflektiert werden. Die Entwicklung mobiler Apps mit plattformübergreifenden Technologien stellt eine Herausforderung dar, wenn das Ergebnis eine möglichst native Benutzererfahrung bieten soll. Der Ansatz bietet aber auch erhebliche Vorteile. Diese Schlussbetrachtung fasst die wichtigsten Erkenntnisse zusammen, bewertet die diskutierten Ansätze und Technologien und gibt einen Ausblick auf strategische Varianten zur Gestaltung des Relaunchs, auch im Hinblick darauf, welches Framework gewählt wird.

6.1 Entscheidungsmatrix

Die Tabelle 6.1 enthält die Bewertung der beiden Frameworks. Die Kriterien stammen aus der Expertenbefragung. Den Kriterien wurde eine Gewichtung (Gew.) zugewiesen. Die Gewichtung ergab sich aus der Auswertung der Interviews und Fragebögen.

Die Punkte wurden auf einer Skala von 0 (nicht erfüllt/nicht vorhanden) bis 5 (voll erfüllt/vorhanden) vergeben. Die Ergebnisse der gewichteten Punkte sind unterhalb der Tabelle aufsummiert.

			Flutter		**KMP**	
Kriterium	**Unterkriterium**	**Gew.**	**Pkt.**	**Pkt. gew.**	**Pkt.**	**Pkt. gew.**
Framework	Dependency-Management (5.3.2)	0,5	4	2	2	1
	Tooling (5.3.3)	0,9	5	4,5	4	3,6
	Performance, Effizienz (5.3.4)	0,5	5	2,5	3	1,5
	Reifegrad (5.3.5)	1	5	5	4	4
	Qualität der Dokumentation (5.3.6)	0,8	5	4	3	2,4
	Code-Wiederverwendbarkeit (5.3.8)	0,5	4	2	2	1
	Testing (5.3.13)	1	4	4	3	3
	Debugging (5.3.14)	0,9	5	4,5	3	2,7
	Beliebtheit, Community	0,9	5	4,5	3	2,7
	Open Source (Ja = 5 P.)	0,4	5	2	5	2

Fortsetzung auf der nächsten Seite

			Flutter		KMP	
Kriterium	**Unterkriterium**	**Gew.**	**Pkt.**	**Pkt. gew.**	**Pkt.**	**Pkt. gew.**
Sprache	Features (5.3.1)	0,4	4	1,6	5	2
	Standardbibliothek (Umfang)	0,8	4	3,2	5	4
	Verbreitung, universelle Verwendbarkeit	0,6	3	1,8	5	3
	Effizienz	1	4	4	5	5
	Lernkurve	0,6	4	2,4	4	2,4
	Reife, Popularrität	1	4	4	5	5
	Verfügbarkeit von Erweiterungen	1	5	5	4	4
Plattformen	Android	1	5	5	5	5
	iOS	1	5	5	4	4
	Web (5.3.9)	0,9	5	4,5	3	2,7
	Desktop Windows (5.3.9)	0,9	5	4,5	5	4,5
	Desktop Mac (5.3.9)	0,5	5	2,5	5	2,5
	Desktop Linux (5.3.9)	0,5	5	2,5	5	2,5
	Embedded (5.3.9)	0,1	5	0,5	0	0
	VR/AR	0,3	0	0	0	0
Resultat	Feature Parity	0,8	5	4	5	4
	User Experience, Usability	0,9	5	4,5	5	4,5
	Native User Experience (5.3.10)	0,7	4	2,8	5	3,5
	Performance	1	4	4	5	5
	Wartbarkeit des Codes	0,5	4	2	5	2,5
	Architektur	0,7	4	2,8	4	2,8
	Schnittstellen	0,5	3	1,5	4	2
HW-Zugriff	Audio-Video-Aufnahme	1	5	5	5	5
	Audio-Video-Wiedergabe	1	5	5	5	5
	GPS (Location Services)	1	5	5	5	5
	QR Code, lesen u. schreiben	1	5	5	5	5
	NFC, lesen u. schreiben	1	5	5	5	5
Orga	Teamstruktur, verteiltes Arbeiten	1	3	3	5	5
	Wissensredundanz im Unternehmen	1	2	2	4	4
	Kostenersparnis	0,8	4	3,2	3	2,4
Summe				136,3		131,2

Tabelle 6.1: Entscheidungsmatrix (Fortgesetzt)

6.2 Fazit

Betrachtet man das Ergebnis der Entscheidungsmatrix, so gewinnt Flutter mit einem knappen Vorsprung. Dies liegt vor allem daran, dass sich Flutter schon wesentlich länger in einem produktiv nutzbaren Zustand befindet. Das merkt man nicht nur an der Stabilität und daran, dass Flutter sehr ausgereift wirkt, sondern auch an der Anzahl der gut funktionierenden und gut dokumentierten Pakete, die zur Verfügung stehen.

Kotlin Multiplatform hingegen hat diesen Status erst vor kurzem erreicht. Es wird noch einige Zeit dauern, bis auch die Tools und Bibliotheken auf einem entsprechenden Stand sind (siehe z.B. die Versionierungsthematik bei Moko-Resources, Kapitel 5.3.12). Dieser Nachteil wird jedoch von Tag zu Tag geringer.

Aus zeitlicher und finanzieller Sicht kann man also durchaus zu Flutter raten. Einziger Nachteil ist die geringere Modularität, die bei der Wahl einer Migrationsstrategie für die neue Version berücksichtigt werden muss (siehe Kapitel 6.3).

Wenn man dann noch bedenkt, dass bei cluetec ein fundiertes Wissen im Bereich Kotlin vorhanden ist, und es bei mQuest einige spezielle Themen gibt (siehe Kapitel 5.1.3), bei denen gerade die Mächtigkeit der Sprache Kotlin von Vorteil sein kann, so schwindet der kleine Vorteil von Flutter schnell. Auf jeden Fall sollte die Wahl des Frameworks im Zusammenhang mit der Wahl des Entwicklungskonzeptes/Migrationspfades gesehen werden, was im folgenden Abschnitt näher beleuchtet wird.

6.3 Ausblick

Abschließend wird diskutiert, wie ein Migrationspfad beim Relaunch der mQuest App aussehen könnte und welche Punkte dabei zu beachten sind. Die komplette Neuentwicklung einer Software dieser Komplexität bringt einige Herausforderungen mit sich. Zum einen wurde in den Interviews deutlich, dass keine der bestehenden Komponenten sinnvoll weiterverwendet werden kann, auch nicht die Logik des mobilecore. Immerhin blickt man auf eine über 20 Jahre gewachsene Softwarestruktur zurück. Zum anderen wurde deutlich, dass es auch schwierig ist, die Entwicklung der neuen Version zu sehr in die Länge zu ziehen. Denn auch die bestehende Lösung muss während dieser Zeit gewartet werden. Weiterentwicklungen bei den Ziel-Betriebssystemen Android und iOS können immer wieder zu Anpassungsbedarf führen. Wichtig ist in jedem Fall auch ein engmaschiges Testen der Neuentwicklung, um Regressionsfehler zu vermeiden - die neue Software darf auf keinen Fall schlechter sein als die bisherige.

Folgende Szenarien stehen für die Entwicklung zur Verfügung:

Aufteilen Um schneller zum Ziel zu gelangen, würde man sich zunächst auf die Entwicklung der iOS-Version konzentrieren, da hier das größere Nutzerpotential liegt. Insbesondere bei der Nutzung von KMP würde dies eine deutliche Entlastung bedeuten, da zunächst nur zwei anstatt drei Teilprojekte bearbeitet werden müssten. Aber auch für Flutter wäre es eine Option, zunächst eine einheitliche UI zu erstellen und diese später für Android und iOS auszudifferenzieren.

MVP In diesem Szenario würde zunächst ein Produkt mit möglichst geringem Funktionsumfang entwickelt. Der Funktionsumfang wäre jedoch so zu definieren, dass ein möglichst großer Teil der Kunden zufrieden gestellt werden kann. Das Produkt würde dann sukzessive zu einer vollumfänglichen Alternative zum bestehenden Produkt ausgebaut, bis dieses vollständig ersetzt werden kann. Für diesen Ansatz sind beide Frameworks gleichermaßen geeignet.

Extern vergeben Zumindest ein Teil der Entwicklungsleistung könnte extern vergeben werden. Das am stärksten spezialisierte Fachwissen steckt im mobilecore, dieser sollte also eher im eigenen Haus entwickelt werden. Aber auch die UI ist bei mQuest recht anspruchsvoll, hier kann durch eine Vergabe an externe Entwickler eine Verkürzung der Gesamtentwicklungszeit erreicht werden, wenn die Entwicklung durch eine saubere Definition der Schnittstellen parallel erfolgen kann. Für diese Variante eignet sich KMP aufgrund der inhärenten Modularität bei der Wahl eines Multi-Repository-Ansatzes besonders gut (siehe Kapitel 5.3.7).

Abschließend sei noch einmal erwähnt, dass für KMP-Projekte zukünftig mit Compose Multiplatform eine Technologie zur Verfügung stehen wird, die es erlaubt, die UI aus einer gemeinsamen Codebasis zu erstellen (siehe Kapitel 3.1.2). Dies bringt weitere zeitliche und damit finanzielle Vorteile bei der Entwicklung und auch bei der Wartung des Produktes.

Literaturverzeichnis

Online-Quellen

1. CHATGPT, 2023. *ChatGPT*. [online]. [Zugriff am: 11. Okt. 2023]. Verfügbar unter: https://chat.openai.com.
2. DEEPL, 2023. *DeepL*. [online]. [Zugriff am: 11. Okt. 2023]. Verfügbar unter: https://www.deepl.com/.
3. MOZILLA, 2023. *WebAssembly*. [online]. [Zugriff am: 11. Okt. 2023]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/WebAssembly.
4. CLUETEC GMBH, 2023. *cluetec - Unternehmen*. [online]. [Zugriff am: 13. Sep. 2023]. Verfügbar unter: https://www.cluetec.de/unternehmen/cluetec/.
5. LUKE WROBLEWSKI, 2009. *Mobile First*. [online]. [Zugriff am: 3. Nov. 2009]. Verfügbar unter: https://www.lukew.com/ff/entry.asp?933.
6. FAZ, 2010. *Eric Schmidt: Googles Devise heißt jetzt „Mobile first“*. [online]. [Zugriff am: 17. Feb. 2010]. Verfügbar unter: https://blogs.faz.net/netzwirtschaft-blog/2010/02/17/googles-devise-heisst-ab-jetzt-mobile-first-1496/.
7. BRITANNICA, 2023. *Palm OS*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://www.britannica.com/technology/Palm-OS.
8. SYMBIAN FOUNDATION, 2023. *All About the Symbian OS*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://symbianfoundation.org/all-about-the-symbian-os/.
9. WIKIPEDIA, 2023. *Windows Mobile*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://en.wikipedia.org/wiki/Windows_Mobile.
10. HISTORY-COMPUTER, 2023. *BlackBerry: Complete Guide — History, Products, Founding, and More*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://history-computer.com/blackberry-history/.
11. MACWORLD, 2023. *iOS versions: Every version of iOS from the oldest to the newest*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://www.macworld.com/article/1659017/ios-versions-list.html.

12. BRITANNICA, 2023. *Android*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://www.britannica.com/technology/Android-operating-system.

13. WIKIPEDIA, 2023. *Windows Phone*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://en.wikipedia.org/wiki/Windows_Phone.

14. WIKIPEDIA (CC BY-SA 3.0), 2023. *Worldwide Smartphone Sales*. [online]. [Zugriff am: 21. Sep. 2023]. Verfügbar unter: https://commons.wikimedia.org/wiki/File:World_Wide_Smartphone_Sales.png.

16. WIKIPEDIA, 2023. *Titanium SDK - Architecture*. [online]. [Zugriff am: 21. Sep. 2023]. Verfügbar unter: https://en.wikipedia.org/wiki/Titanium_SDK#Architecture.

23. STACK OVERFLOW, 2023. *Stack Overflow Trends*. [online]. [Zugriff am: 21. Sep. 2023]. Verfügbar unter: https://insights.stackoverflow.com/trends/.

24. JETBRAINS, 2023. *General Development Trends*. [online]. [Zugriff am: 26. Jan. 2024]. Verfügbar unter: https://www.jetbrains.com/lp/devecosystem-2023/development/.

28. MULTI-OS ENGINE, 2023. *Multi-OS Engine Blog*. [online]. [Zugriff am: 26. Sep. 2023]. Verfügbar unter: https://multi-os-engine.org/blog/2022-09-29-move-website-to-github/.

29. EMBARCADERO, 2023. *RAD Studio 11.3*. [online]. [Zugriff am: 22. Sep. 2023]. Verfügbar unter: https://www.embarcadero.com/products/rad-studio/.

30. YOUTUBE, 2023. *What's New in RAD Studio and Delphi XE4*. [online]. [Zugriff am: 22. Sep. 2023]. Verfügbar unter: https://www.youtube.com/watch?v=RS1rFaRWIzA.

31. STACK OVERFLOW, 2023. *2023 Developer Survey : Technology*. [online]. [Zugriff am: 4. Okt. 2023]. Verfügbar unter: https://survey.stackoverflow.co/2023/#technology.

32. IEEE SPECTRUM, 2023. *The Top Programming Languages 2023*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: https://spectrum.ieee.org/the-top-programming-languages-2023.

33. INCREMENT.COM, 2023. *The new shiny*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://increment.com/mobile/the-shift-to-declarative-ui/.

35. APPLE, 2023. *Managing data flow between views*. [online]. [Zugriff am: 29. Sep. 2023]. Verfügbar unter: https://developer.apple.com/tutorials/app-dev-training/managing-data-flow-between-views.

36. APPLE, 2023. *SwiftUI*. [online]. [Zugriff am: 26. Sep. 2023]. Verfügbar unter: https://developer.apple.com/xcode/swiftui/.

37. JETBRAINS, 2023. *Update on the Name of Kotlin Multiplatform*. [online]. [Zugriff am: 26. Sep. 2023]. Verfügbar unter: `https://blog.jetbrains.com/kotlin/2023/07/update-on-the-name-of-kotlin-multiplatform/`.

39. DMITRY JEMEROV, 2023. *Dmitry Jemerov*. [online]. [Zugriff am: 12. Okt. 2023]. Verfügbar unter: `https://www.yole.page/`.

40. JETBRAINS, 2023. *Why JetBrains needs Kotlin*. [online]. [Zugriff am: 26. Sep. 2023]. Verfügbar unter: `https://blog.jetbrains.com/kotlin/2011/08/why-jetbrains-needs-kotlin/`.

42. ANDROID DEVELOPERS, 2023. *Android's Kotlin-first approach*. [online]. [Zugriff am: 12. Okt. 2023]. Verfügbar unter: `https://developer.android.com/kotlin/first`.

43. KOTLIN, 2023. *Kotlin Multiplatform*. [online]. [Zugriff am: 26. Sep. 2023]. Verfügbar unter: `https://kotlinlang.org/docs/native-overview.html`.

44. JETBRAINS, 2023. *Compose Multiplatform*. [online]. [Zugriff am: 26. Sep. 2023]. Verfügbar unter: `https://www.jetbrains.com/lp/compose-multiplatform/`.

45. FLUTTER, 2023. *Beautiful apps for every screen*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: `https://flutter.dev/multi-platform`.

47. GOOGLE CODE BLOG, 2023. *Dart: a language for structured web programming*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: `http://googlecode.blogspot.com/2011/10/dart-language-for-structured-web.html`.

48. GOOGLE, 2023. *Dart*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: `https://dart.dev/`.

49. DART, 2024. *Command-line and server apps*. [online]. [Zugriff am: 18. Jan. 2024]. Verfügbar unter: `https://dart.dev/server`.

50. ANGULARDART, 2024. *AngularDart*. [online]. [Zugriff am: 18. Jan. 2024]. Verfügbar unter: `https://angulardart.xyz/`.

51. FLUTTER DOCS, 2023. *Flutter architectural overview*. [online]. [Zugriff am: 29. Sep. 2023]. Verfügbar unter: `https://docs.flutter.dev/resources/architectural-overview`.

52. FLUTTER DOCS, 2023. *Hot reload - Previous state is combined with new code*. [online]. [Zugriff am: 11. Okt. 2023]. Verfügbar unter: `https://docs.flutter.dev/tools/hot-reload#previous-state-is-combined-with-new-code`.

54. FLUTTER DOCS, 2023. *Reactive user interfaces*. [online]. [Zugriff am: 29. Sep. 2023]. Verfügbar unter: `https://docs.flutter.dev/resources/architectural-overview#reactive-user-interfaces`.

56. WIKIPEDIA, 2023. *TypeScript*. [online]. [Zugriff am: 10. Nov. 2023]. Verfügbar unter: `https://de.wikipedia.org/wiki/TypeScript`.

57. META OPEN SOURCE, 2023. *Using TypeScript*. [online]. [Zugriff am: 28. Sep. 2023]. Verfügbar unter: https://reactnative.dev/docs/typescript.

58. META OPEN SOURCE, 2023. *Core Components and Native Components*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://reactnative.dev/docs/intro-react-native-components.

59. AKVEO INC., 2023. *How does React Native work?* [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://www.akveo.com/blog/how-does-react-native-work.

60. DEV, 2023. *C# Markup*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://dev.to/wjimmycook/how-the-react-native-bridge-works-and-how-it-will-change-in-the-near-future-4ekc.

61. MICROSOFT, 2023. *Supported platforms for .NET MAUI apps*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://learn.microsoft.com/en-us/dotnet/maui/supported-platforms.

62. WIKIPEDIA, 2024. *C Sharp (programming language)*. [online]. [Zugriff am: 28. Jan. 2024]. Verfügbar unter: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

63. MICROSOFT, 2023. *What is .NET MAUI?* [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui.

64. MICROSOFT, 2023. *C# Markup*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/markup/markup.

65. EVAN CZAPLICKI, 2023. *The Elm Architecture*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://guide.elm-lang.org/architecture/.

66. MICROSOFT, 2023. *Introducing .NET Multi-platform App UI*. [online]. [Zugriff am: 27. Sep. 2023]. Verfügbar unter: https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/#mvu.

67. GITHUB, 2023. *MauiReactor*. [online]. [Zugriff am: 29. Sep. 2023]. Verfügbar unter: https://github.com/adospace/reactorui-maui.

69. KOTLIFT, 2023. *Kotlift*. [online]. [Zugriff am: 30. Okt. 2023]. Verfügbar unter: https://github.com/studoverse/Kotlift.

70. GITHUB, 2023. *SwiftKotlin*. [online]. [Zugriff am: 30. Okt. 2023]. Verfügbar unter: https://github.com/angelolloqui/SwiftKotlin/releases.

71. GRYPHON, 2023. *The Swift to Kotlin translator*. [online]. [Zugriff am: 30. Okt. 2023]. Verfügbar unter: https://vinivendra.github.io/Gryphon/.

72. GITHUB/GRYPHON, 2023. *Issue: Support for Swift 5.8 and Xcode 14.3*. [online]. [Zugriff am: 6. Nov. 2023]. Verfügbar unter: `https://github.com/vinivendra/Gryphon/issues/127`.

73. DOMINIK SCHULTES, TECHNISCHE HOCHSCHULE MITTELHESSEN, 2023. *SequalsK - A Bidirectional Transpiler*. [online]. [Zugriff am: 30. Okt. 2023]. Verfügbar unter: `https://transpile.iem.thm.de/`.

74. JETBRAINS, DOMINIK SCHULTES, 2023. *SequalsK*. [online]. [Zugriff am: 30. Okt. 2023]. Verfügbar unter: `https://plugins.jetbrains.com/plugin/17097-sequalsk`.

75. TERENCE PARR, 2023. *ANTLR*. [online]. [Zugriff am: 9. Nov. 2023]. Verfügbar unter: `https://www.antlr.org/`.

76. JETBRAINS, 2023. *Kotlin Docs - Grammar*. [online]. [Zugriff am: 9. Nov. 2023]. Verfügbar unter: `https://kotlinlang.org/docs/reference/grammar.html`.

77. GITHUB, 2023. *antlr/grammars-v4/swift/swift3*. [online]. [Zugriff am: 9. Nov. 2023]. Verfügbar unter: `https://github.com/antlr/grammars-v4/tree/master/swift/swift3`.

79. SWIFTLIN, 2023. *Swiftlin Converter*. [online]. [Zugriff am: 23. Okt. 2023]. Verfügbar unter: `https://swift2kotlin.com/converter/code/`.

80. GOOGLE DEVELOPERS, 2023. *What J2ObjC is*. [online]. [Zugriff am: 21. Sep. 2023]. Verfügbar unter: `https://developers.google.com/j2objc`.

84. MICROSOFT, 2023. *Xamarin Blog : Fall 2022 Updates for Xamarin, Hello iOS 16 and Android 13*. [online]. [Zugriff am: 4. Okt. 2023]. Verfügbar unter: `https://devblogs.microsoft.com/xamarin/xamarin-ios-16-android-13-updates/`.

85. GITHUB, 2023. *The top programming languages*. [online]. [Zugriff am: 6. Okt. 2023]. Verfügbar unter: `https://octoverse.github.com/2022/top-programming-languages`.

93. DEBIAN GITLAB, 2023. *benchmarksgame*. [online]. [Zugriff am: 4. Okt. 2023]. Verfügbar unter: `https://salsa.debian.org/benchmarksgame-team/benchmarksgame`.

94. APPTIM, 2023. *Efficiently test and fine-tune the performance of your mobile app with AI-driven solutions*. [online]. [Zugriff am: 4. Okt. 2023]. Verfügbar unter: `https://salsa.debian.org/benchmarksgame-team/benchmarksgame`.

95. GITHUB, 2023. *mobileperf*. [online]. [Zugriff am: 4. Okt. 2023]. Verfügbar unter: `https://github.com/alibaba/mobileperf`.

96. APPLE, 2023. *Performance and metrics*. [online]. [Zugriff am: 4. Okt. 2023]. Verfügbar unter: `https://developer.apple.com/documentation/xcode/performance-and-metrics`.

97. ANDROID DEVELOPERS, 2023. *Profile your app performance*. [online]. [Zugriff am: 4. Okt. 2023]. Verfügbar unter: `https://developer.android.com/studio/profile`.

98. AUDIOTRANSKRIPTION DR. DRESING & PEHL GMBH, 2023. *Qualitative Inhaltsanalyse – nach Mayring oder nach Kuckartz?* [online]. [Zugriff am: 23. Okt. 2023]. Verfügbar unter: `https://www.audiotranskription.de/qualitative-inhaltsanalyse-nach-mayring-oder-nach-kuckartz/`.

104. MACWHISPER, 2023. *Private, high-quality transcription*. [online]. [Zugriff am: 20. Okt. 2023]. Verfügbar unter: `https://goodsnooze.gumroad.com/l/macwhisper`.

106. MATTHEW WEINSTEIN, 2023. *TAMS Analyzer for Macintosh OS X*. [online]. [Zugriff am: 20. Okt. 2023]. Verfügbar unter: `https://tamsys.sourceforge.io/osxtams/`.

107. DISCOGS, 2024. *Discogs API*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: `https://www.discogs.com/developers/`.

108. FIGMA, 2023. *Firgma Design*. [online]. [Zugriff am: 16. Nov. 2023]. Verfügbar unter: `https://www.figma.com/design-overview/`.

109. SYSTEM, GNU Operating, 2023. *GNU PSPP*. [online]. [Zugriff am: 5. Dez. 2023]. Verfügbar unter: `https://www.gnu.org/software/pspp/`.

110. MAURER, Max-Emanuel, 2013. *likertplot.com - Plot Likert Scales*. [online]. [Zugriff am: 5. Dez. 2023]. Verfügbar unter: `http://www.likertplot.com/`.

111. KOTLIN, 2024. *Coroutines basics*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://kotlinlang.org/docs/coroutines-basics.html`.

112. DART, 2024. *Concurrency in Dart*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://dart.dev/language/concurrency`.

113. KOTLIN, 2024. *Scope functions*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://kotlinlang.org/docs/scope-functions.html`.

114. DART, 2024. *Cascade notation*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://dart.dev/language/operators#cascade-notation`.

115. DART, 2024. *Anonymous functions*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://dart.dev/language/functions#anonymous-functions`.

116. KOTLIN, 2024. *Data Classes*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://kotlinlang.org/docs/data-classes.html`.

117. DART, 2024. *Records*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://dart.dev/language/records`.

118. KOTLIN, 2024. *Patterns*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: `https://kotlinlang.org/docs/control-flow.html#when-expression`.

119. DART, 2024. *Patterns*. [online]. [Zugriff am: 13. Jan. 2024]. Verfügbar unter: https://dart.dev/language/patterns.

120. PUB.DEV, 2024. *The official package repository for Dart and Flutter apps*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://pub.dev/.

121. WIDGETBOOK, 2024. *The collaboration platform for Flutter*. [online]. [Zugriff am: 18. Jan. 2024]. Verfügbar unter: https://www.widgetbook.io/.

122. GITHUB, 2024. *Awesome Kotlin Multiplatform*. [online]. [Zugriff am: 20. Jan. 2024]. Verfügbar unter: https://github.com/terrakok/kmp-awesome.

123. JETBRAINS, 2024. *Amper – Improving the Build Tooling User Experience*. [online]. [Zugriff am: 25. Jan. 2024]. Verfügbar unter: https://blog.jetbrains.com/blog/2023/11/09/amper-improving-the-build-tooling-user-experience/.

124. GITHUB.COM - FLUTTER, 2024. *v0.0.6: Rev alpha branch version to 0.0.6, flutter 0.0.26 (#10010)*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://github.com/flutter/flutter/releases/tag/v0.0.6.

125. KOTLIN, 2024. *What's new in Kotlin 1.2*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://kotlinlang.org/docs/whatsnew12.html#table-of-contents.

126. GOOGLE, 2024. *Flutter Live 2018*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://developers.google.com/events/flutter-live.

127. MEDIUM, 2024. *Announcing Flutter 1.12: What a year!* [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://medium.com/flutter/announcing-flutter-1-12-what-a-year-22c256ba525d.

128. FLUTTER, 2024. *Flutter 2023 Strategy*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://flutter.dev/go/strategy-2023.

129. STATISTA, 2024. *Number of available applications in the Google Play Store from December 2009 to June 2023*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

130. JETBRAINS, 2023. *Kotlin Multiplatform Is Stable and Production-Ready*. [online]. [Zugriff am: 1. Nov. 2023]. Verfügbar unter: https://blog.jetbrains.com/kotlin/2023/11/kotlin-multiplatform-stable/.

131. FLUTTER DOCS, 2024. *Flutter Documentation*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://docs.flutter.dev/.

132. KOTLIN, 2024. *Kotlin Multiplatform Documentation*. [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: https://kotlinlang.org/docs/multiplatform.html.

133. KOTLIN, 2024. *Kotlin Docs*. [online]. [Zugriff am: 8. Jan. 2024]. Verfügbar unter: https://kotlinlang.org/docs/home.html.

134. REALM KOTLIN, 2024. *Fast, powerful data storage for Kotlin and KMM apps.* [online]. [Zugriff am: 5. Jan. 2024]. Verfügbar unter: `https://realm.io/realm-kotlin/`.

135. STACKOVERFLOW, 2024. *How to use my dart packages private and not show on pub dart lang?* [online]. [Zugriff am: 16. Jan. 2024]. Verfügbar unter: `https://stackoverflow.com/questions/54143695/how-to-use-my-dart-packages-private-and-not-show-on-pub-dart-lang/68023976#68023976`.

137. KOTLIN MULTIPLATFORM DEVELOPMENT, 2024. *Stability of supported platforms.* [online]. [Zugriff am: 16. Jan. 2024]. Verfügbar unter: `https://www.jetbrains.com/help/kotlin-multiplatform-dev/supported-platforms.html#compose-multiplatform-ui-framework-stability-levels`.

138. FLUTTER, 2024. *Beautiful apps for every screen.* [online]. [Zugriff am: 16. Jan. 2024]. Verfügbar unter: `https://flutter.dev/multi-platform`.

139. KOTLIN, 2024. *Kotlin for JavaScript.* [online]. [Zugriff am: 16. Jan. 2024]. Verfügbar unter: `https://kotlinlang.org/docs/js-overview.html`.

140. FLUTTER, 2024. *Flutter on Desktop.* [online]. [Zugriff am: 16. Jan. 2024]. Verfügbar unter: `https://flutter.dev/multi-platform/desktop`.

141. FLUTTER DOCS, 2023. *Material Components widgets.* [online]. [Zugriff am: 22. Nov. 2023]. Verfügbar unter: `https://docs.flutter.dev/ui/widgets/material`.

142. FLUTTER DOCS, 2023. *Cupertino (iOS-style) widgets.* [online]. [Zugriff am: 22. Nov. 2023]. Verfügbar unter: `https://docs.flutter.dev/ui/widgets/cupertino`.

143. FLUTTER, 2023. *Flutter SDK.* [online]. [Zugriff am: 23. Nov. 2023]. Verfügbar unter: `https://api.flutter.dev/flutter/cupertino/CupertinoIcons-class.html`.

144. GOOGLE FONTS, 2023. *Material Icons.* [online]. [Zugriff am: 23. Nov. 2023]. Verfügbar unter: `https://fonts.google.com/icons`.

145. FLUTTER, 2024. *Automatic platform adaptations.* [online]. [Zugriff am: 20. Jan. 2024]. Verfügbar unter: `https://docs.flutter.dev/platform-integration/platform-adaptations`.

146. FLUTTER, 2024. *Writing custom platform-specific code.* [online]. [Zugriff am: 20. Jan. 2024]. Verfügbar unter: `https://docs.flutter.dev/platform-integration/platform-channels?tab=type-mappings-kotlin-tab`.

147. MEDIUM, 2024. *Using Native Code with Flutter.* [online]. [Zugriff am: 20. Jan. 2024]. Verfügbar unter: `https://andyjulow.medium.com/using-native-code-with-flutter-21645d104ed0`.

148. FLUTTER DOCS, 2023. *Internationalizing Flutter apps*. [online]. [Zugriff am: 20. Nov. 2023]. Verfügbar unter: `https://docs.flutter.dev/ui/accessibility-and-internationalization/internationalization`.

149. ICEROCK, 2024. *We are Kotlin Multiplatform experts*. [online]. [Zugriff am: 19. Jan. 2024]. Verfügbar unter: `https://icerockdev.com/`.

150. GITHUB, 2024. *Moko Resources*. [online]. [Zugriff am: 19. Jan. 2024]. Verfügbar unter: `https://github.com/icerockdev/moko-resources`.

151. GITHUB, 2024. *Libres*. [online]. [Zugriff am: 22. Jan. 2024]. Verfügbar unter: `https://github.com/Skeptick/libres`.

152. COCOAPODS, 2024. *<CocoaPods>*. [online]. [Zugriff am: 22. Jan. 2024]. Verfügbar unter: `https://cocoapods.org/`.

153. FLUTTER, 2024. *Create flavors of a Flutter app*. [online]. [Zugriff am: 18. Jan. 2024]. Verfügbar unter: `https://docs.flutter.dev/deployment/flavors`.

154. PROGRAMMIER.BAR - DER PODCAST FÜR APP- UND WEBENTWICKLUNG, 2024. *Deep Dive 128 - Full Stack Dart*. [online]. [Zugriff am: 18. Jan. 2024]. Verfügbar unter: `https://www.programmier.bar/podcast/deep-dive-128-full-stack-dart`.

155. PUB.DEV, 2024. *dart_define*. [online]. [Zugriff am: 18. Jan. 2024]. Verfügbar unter: `https://pub.dev/documentation/dart_define/latest/`.

156. PUB.DEV, 2024. *Fasttrack*. [online]. [Zugriff am: 18. Jan. 2024]. Verfügbar unter: `https://pub.dev/packages/fasttrack`.

Thesis-Quellen

17. WU, Wenhao, 2018. *React Native vs Flutter, Cross-platforms mobile application frameworks* [Bachelorarbeit]. Helsinki: Metropolia Ammattikorkeakoulu. Verfügbar unter: `https://urn.fi/URN:NBN:fi:amk-201805158156`.

19. ARMAGAN, Ferdi, 2020. *Vergleich von nativer App- und Cross-Platform-Entwicklung (Facebook React Native und Google Flutter)* [Bachelorarbeit]. Hannover: Hochschule Hannover - University of Applied Sciences. Verfügbar unter: `https://serwiss.bib.hs-hannover.de/frontdoor/index/index/docId/1630`.

25. FAYZULLAEV, Jakhongir, 2018. *Native-like Cross-Platform Mobile Development : Multi-OS Engine & Kotlin Native vs Flutter* [Bachelorarbeit]. Mikkeli: South-Eastern Finland University of Applied Sciences. Verfügbar unter: `https://urn.fi/URN:NBN:fi:amk-2018053011229`.

26. EVERT, Anna-Karin, 2019. *Cross-Platform Smartphone Application Development with Kotlin Multiplatform : Possible Impacts on Development Productivity, Application Size and Startup Time* [Masterarbeit]. Stockholm: KTH, School of Electrical Engineering und Computer Science (EECS). Verfügbar unter: `http://www.diva-portal.org/smash/get/diva2:1368323/FULLTEXT01.pdf`.

27. SKANTZ, Anna, 2023. *Performance Evaluation of Kotlin Multiplatform Mobile and Native iOS Development in Swift* [Bachelorarbeit]. Stockholm: KTH Royal Institute of Technology. Verfügbar unter: `https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-335119`.

68. SCHNEIDER, Larissa, 2022. *A Construct-Based Evaluation of Kotlin-to-Swift and Swift-to-Kotlin Transpilers* [Masterarbeit]. Gießen: Technische Hochschule Mittelhessen. Verfügbar unter: `https://publikationsserver.thm.de/xmlui/handle/123456789/246`.

89. AHMAD, Marghoob, 2023. *Analysis of Cross Platform Mobile Application Development Frameworks* [Bachelorarbeit]. Riga: Riga Technical University. Verfügbar unter: `https://www.researchgate.net/publication/372679769_ANALYSIS_OF_CROSS_PLATFORM_MOBILE_APPLICATION_DEVELOPMENT_FRAMEWORKS`.

Artikel-Quellen

15. XANTHOPOULOS, S. und S. XINOGALOS, 2013. *A comparative analysis of cross-platform development approaches for mobile applications.* In: Proceedings of the 6th Balkan Conference in Informatics [online]. Verfügbar unter: `https://www.researchgate.net/profile/Stelios-Xinogalos/publication/258010031_A_Comparative_Analysis_of_Cross-platform_Development_Approaches_for_Mobile_Applications/links/0046353a3192808faa000000/A-Comparative-Analysis-of-Cross-platform-Development-Approaches-for-Mobile-Applications.pdf`.

18. BIØRN-HANSEN, Andreas; Christoph RIEGER; Tor-Morten GRØNLI; Tim A. MAJCHRZAK und Gheorghita GHINEA, 2020. *An empirical investigation of performance overhead in cross-platform mobile development frameworks.* In: Empir. Softw. Eng. [online]. Verfügbar unter: `https://doi.org/10.1007/s10664-020-09827-6`.

20. NAWROCKI, Piotr; Krzysztof WRONA; Mateusz MARCZAK und Bartlomiej SNIEZYNSKI, 2021. *A Comparison of Native and Cross-Platform Frameworks for Mobile Applications.* In: Computer [online]. Verfügbar unter: `https://doi.org/10.1109/MC.2020.2983893`.

21. MOTA, Dany, 2021. *An Approach to Asses the Performance of Mobile Applications: A Case Study of Multiplatform Development Frameworks*. In: Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering [online]. Verfügbar unter: `https://www.scitepress.org/Papers/2021/104974/104974.pdf`.

22. OLIVEIRA, Wellington; Bernardo MORAES; Fernando CASTOR und Joao Paulo FERNANDES, 2023. *Analyzing the Resource Usage Overhead of Mobile App Development Frameworks*. In: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering [online]. Verfügbar unter: `https://dl.acm.org/doi/10.1145/3593434.3593487`.

55. KIM, J. und E. LEE, 2022. *An empirical study on rule violation history of JavaScript code blocks on Stack Overflow*. In: International Journal of Software Engineering and Knowledge Engineering [online]. Verfügbar unter: `https://doi.org/10.1142/s0218194022500310`.

78. SCHULTES, Dominik, 2021. *SequalsK—A Bidirectional Swift-Kotlin-Transpiler*. In: 2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft) [online]. Verfügbar unter: `https://ieeexplore.ieee.org/document/9460946`.

81. SURI, Bhawna; Shweta TANEJA; Isha BHANOT; Himanshi SHARMA und Aanchal RAJ, 2023. *Cross-Platform Empirical Analysis of Mobile Application Development Frameworks: Kotlin, React Native and Flutter*. In: Proceedings of the 4th International Conference on Information Management & Machine Intelligence [online]. Verfügbar unter: `https://doi.org/10.1145/3590837.3590897`.

83. TOM, Ali El; Cristian BOGDAN; Tim A. MAJCHRZAK und Tor-Morten GRØNLI, 2023. *Criteria Based Evaluation of Cross-Platform Development Frameworks*. In: 56th Hawaii International Conference on System Sciences, HICSS 2023, Maui, Hawaii, USA, January 3-6, 2023 [online]. Verfügbar unter: `https://hdl.handle.net/10125/103475`.

86. RIEGER, Christoph und Tim A. MAJCHRZAK, 2019. *Towards the definitive evaluation framework for cross-platform app development approaches*. In: Journal of Systems and Software [online]. Verfügbar unter: `https://www.sciencedirect.com/science/article/pii/S0164121219300743`.

87. MAJRASHI, Khalid; Margaret HAMILTON; Alexandra UITDENBOGERD und Shiroq AL-MEGREN, 2020. *Cross-Platform Usability Model Evaluation*. In: Multimodal Technologies and Interaction [online]. Verfügbar unter: `https://doi.org/10.3390/mti4040080`.

88. ALKHALIFAH, Ali, 2022. *Predicting Mobile Cross-Platform Adaptation Using a Hybrid Sem–ANN Approach.* In: Computer Systems Science and Engineering [online]. Verfügbar unter: https://www.techscience.com/csse/v42n2/46126.

90. YOU, Dongliang und Minjie HU, 2021. *A Comparative Study of Cross-platform Mobile Application Development* [online]. Verfügbar unter: https://www.researchgate.net/publication/357898491_A_Comparative_Study_of_Cross-platform_Mobile_Application_Development.

91. IŞITAN, Mehmet und Murat KOKLU, 2020. *Comparison and Evaluation of Cross Platform Mobile Application Development Tools.* In: International Journal of Applied Mathematics Electronics and Computers [online]. Verfügbar unter: https://dergipark.org.tr/en/pub/ijamec/issue/57538/832673.

92. BIØRN-HANSEN, Andreas; Tor-Morten GRØNLI; Gheorghita GHINEA und Sahel ALOUNEH, 2019. *An Empirical Study of Cross-Platform Mobile Development in Industry.* In: Wireless Communications and Mobile Computing [online]. Verfügbar unter: https://www.hindawi.com/journals/wcmc/2019/5743892/.

99. WEBER, Susanne Theresia, 2021. *Die Inhaltsanalyse nach Mayring als Auswertungsmethode für wissenschaftliche Interviews* [online]. Verfügbar unter: https://res.cloudinary.com/iubh/image/upload/v1622450290/Presse%20und%20Forschung/Discussion%20Papers/Business%20and%20Management/DP_BUS_6_2021_Weber_Wernitz_fin_fb5u47.pdf.

100. ONWUEGBUZIE, Anthony J., 2000. *On Becoming a Bi-Researcher: The Importance of Combining Quantitative and Qualitative Research Methodologies.* [online]. Verfügbar unter: https://eric.ed.gov/?id=ED448201.

101. ROCH, Svenja, 2017. *Der Mixed-Methods-Ansatz* [online]. Verfügbar unter: https://www.researchgate.net/publication/320877887_Der_Mixed-Methods-Ansatz.

Buch-Quellen

34. HOMEM, Luis Manuel Cabrita Pais. 2013. *Topics in Programming Languages: A Philosophical Analysis Through the Case of Prolog.* Oxford: Chartridge Books Oxford. ISBN 1909287725.

38. KOHLS, Christian; Alexander DOBRYNIN und Florian LEONHARD. 2020. *Programmieren lernen mit Kotlin.* München: Carl Hanser Verlag GmbH & Co. KG. ISBN 3446467025.

41. KÜNNETH, Thomas. 2021. *Android 11 : das Praxisbuch für App-Entwickler.* [6., aktualisierte Auflage]. Bonn: Rheinwerk Verlag. ISBN 9783836270045.

46. KOPEC, David. 2014. *Dart for Absolute Beginners*. Berkeley, CA: Apress. ISBN 978-1-4302-6481-1.

53. GOLL, Joachim. 2013. *Architektur- und Entwurfsmuster der Softwaretechnik : Mit lauffähigen Beispielen in Java*. Wiesbaden: Springer Vieweg. ISBN 9783834824325.

82. SHWETHA, A.N.; R. SUMATHI und C.P. PRABODH. 2023. *Mobile Application Development: Practice and Experience : 12th Industry Symposium in Conjunction with 18th ICDCIT 2022*. 1st ed. 2023. Singapore: Springer Nature Singapore. ISBN 9789811968938.

102. KUCKARTZ, Udo. 2022. *Qualitative Inhaltsanalyse. Methoden, Praxis, Computerunterstützung : Grundlagentexte Methoden*. 5. Auflage. Weinheim: Beltz Juventa. ISBN 9783779955337.

103. MISOCH, Sabina. 2015. *Qualitative Interviews*. Berlin: Walter de Gruyter GmbH. ISBN 9783110354614.

105. MEY, Günter. 2010. *Handbuch Qualitative Forschung in der Psychologie*. Wiesbaden: VS Verlag für Sozialwissenschaften. ISBN 9783531920528.

136. NAGY, Róbert. 2022. *Simplifying Application Development with Kotlin Multiplatform Mobile*. Birmingham: Packt Publishing. ISBN 9781801812580.