JURNAL INFORMATIKA, Vol. 9 No. 2 Oktober 2022, Halaman 159~171
ISSN: 2355-6579 | E-ISSN**:** 2528-2247

Studi Komparasi *Maintainability* Antara Aplikasi yang Dikembangkan dengan *Framework* Flutter dan React Native

Muhammad Deta Aditya[1], Meredita Susanty[2]

[1,2] Universitas Pertamina
Jl Teuku Nyak Arief, Simprug, Kebayoran Lama, Jakarta Selatan 12220, Indonesia

e-mail: [1]muhammaddetaaditya@gmail.com, [2]meredita.susanty@universitaspertamina.ac.id

Informasi Artikel	Diterima: 21-05-2022	Direvisi: 06-10-2022	Disetujui: 09-10-2022

Abstrak

Setiap tahunnya, jumlah pengguna aplikasi seluler terus bertambah. Hal ini menyebabkan pengembang aplikasi seluler dituntut untuk mengembangkan produknya dengan cepat. *Framework* antarmuka pengguna lintas *platform* Flutter dan React Native menjadi sangat populer karena kemampuannya untuk dikompilasi ke berbagai jenis sistem operasi perangkat seluler. Namun, di antara kedua *framework* tersebut, belum diketahui manakah yang memiliki *maintainability* yang lebih baik. Tahap perawatan perangkat lunak merupakan tahap yang memakan paling banyak waktu, tenaga, dan usaha. Sehingga, penting untuk mengetahui *framework* mana yang basis kodenya lebih mudah dirawat. Pada penelitian ini, aplikasi media komunikasi untuk taman kanak-kanak dengan kompleksitas *use-case* yang beragam digunakan sebagai bahan untuk membandingkan *maintainability* Flutter dan React Native. Basis kode yang dikembangkan masing-masing diukur menggunakan *A Code Quality Metrics Model for React-Based Web Applications*. Hasil penelitian menunjukkan bahwa basis kode yang ditulis menggunakan Flutter memiliki *maintainability* lebih tinggi dibandingkan React Native. Faktor yang mempengaruhinya adalah bahasa pemrograman yang digunakannya, yaitu Dart untuk Flutter dan JavaScript untuk React Native, serta cara penulisan komponen yang berbeda antara Flutter dan React Native.

Kata Kunci: Flutter, React Native, *Maintainability*, Studi Komparasi, Aplikasi Mobile, Kualitas Kode

Abstract

Mobile phone users are increasing every year. This demands mobile application developers to speed up their development time. Flutter and React Native, which are cross-platform user interface framework, become more popular due to their ability to compile to any mobile operating system. But, it is not yet known which of the two frameworks have the better maintainability. Maintenance is a step in software development process that consumes most of the development effort. So, it is important to know which framework produces the more maintainable code base. In this research, a communication media application for kindergarten with diverse use case complexity is used for comparing maintainability between Flutter and React Native. Each code base are measured using A Code Quality Metrics Model for React-Based Web Applications. The result is code base that are developed using Flutter has more maintainability than React Native. The influencing factors are programming languages used, which are Dart for Flutter and JavaScript for React, as well as how components in each frameworks are written.

Keywords*: Flutter, React Native, Maintainability, Comparative Study, Mobile Application, Code Quality*

1. Pendahuluan

Jumlah pengguna aplikasi seluler terus meningkat setiap tahun. Terdapat 178.1 miliar unduhan aplikasi seluler di tahun 2017 dan 194 miliar pada 2018 (J.Clement, 2019). Dari banyaknya pengguna aplikasi seluler, 72.92% adalah pengguna ponsel pintar dengan sistem operasi Android, 26.53% pengguna iOS, dan 0.55% pengguna sistem operasi lainnya (S.O'Dea, 2020). Dominasi dari kedua sistem operasi tersebut menuntut pengembang aplikasi untuk mengembangkan dua aplikasi seluler yang sama untuk berjalan di Android dan iOS. Hal ini mengakibatkan biaya pengembangan menjadi dua kali lipat per aplikasi yang dikembangkan. Solusi untuk masalah tersebut adalah pendekatan pengembangan aplikasi seluler yang lintas *platform*, yang mengizinkan pengembang untuk menuliskan satu basis kode

saja untuk aplikasi pada *platform* yang berbeda-beda (Rahul Raj & Tolety, 2012).

Terdapat beberapa *framework* untuk mengembangkan aplikasi seluler yang menerapkan pendekatan lintas *platform*. Pada awal tahun 2021, Flutter dan React Native adalah *framework* open source yang paling diminati oleh pengembang dan memiliki kontributor terbanyak, mengalahkan *framework* Xamarin dan Ionic. Jumlah star, fork, dan kontributor repository utama masing-masing *framework* adalah 113.000, 16.100, 801 untuk Flutter (*GitHub - Flutter/Flutter: Flutter Makes It Easy and Fast to Build Beautiful Apps for Mobile and Beyond*, n.d.), 93.400, 20.500, 2.258 untuk React Native, 5.100, 1.900 (*GitHub - Facebook/React-Native: A Framework for Building Native Applications Using React*, n.d.), 316 untuk Xamarin (*GitHub - Xamarin/Xamarin.Forms: Xamarin.Forms Official Home*, n.d.), dan 43.000, 13.300, 400 untuk Ionic (*GitHub - Ionic-Team/Ionic-Framework: A Powerful Cross-Platform UI Toolkit for Building Native-Quality IOS, Android, and Progressive Web Apps with HTML, CSS, and JavaScript.*, n.d.). Flutter dan React Native memiliki kesamaan dalam paradigma pemrograman dan blok pembangun, yakni deklaratif dan berbasis komponen (*Components and Props – React*, n.d.; *Introduction to Declarative UI - Flutter*, n.d.; *Introduction to Widgets - Flutter*, n.d.). Namun, keduanya memiliki pendekatan yang berbeda dalam mengelola state (*Differentiate between Ephemeral State and App State | Flutter*, n.d.; *State and Lifecycle – React*, n.d.). Bahasa pemrogramman yang digunakan pada Flutter dan React Native berbeda, yaitu Dart untuk Flutter (*FAQ - Flutter*, n.d.), dan JavaScript untuk React Native (*JavaScript Environment · React Native*, n.d.). Perbedaan tersebut diasumsikan menyebabkan kode yang ditulis menggunakan Flutter dan React Native masing-masing memiliki kualitas yang berbeda.

Penelitian yang membahas perbandingan Flutter dan React Native pada segi *performance* sudah pernah dilakukan sebelumnya (Jagiełło, 2019; Wu, 2018). Namun, belum ada penelitian yang membahas *maintainability* pada kedua *framework* tersebut. *Maintainability* adalah salah satu aspek kualitas perangkat lunak bersama dengan *functional suitability*, *performance efficiency*, *compatibility*, *usability*, *reliability*, *security*, dan *portability* (*ISO - ISO/IEC 25010:2011 - Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models*, n.d.). *Maintainability* adalah hal yang perlu diperhatikan pada tahap *software evolution*, yaitu saat aplikasi perlu dirawat untuk mengurangi kesalahan (*corrective*), beradaptasi pada pembaharuan sistem (*adaptive*), disesuaikan dengan kebutuhan baru (*perfective*), dan mencegah terjadinya kesalahan baru (*preventive*) (Sommerville, 2016). Semakin tinggi *maintainability* suatu perangkat lunak, semakin mudah aplikasi untuk dirawat, yang berarti biaya, waktu, dan usaha yang dikeluarkan untuk perawatan aplikasi akan berkurang (Dehaghani & Hajrahimi, 2013; Jang & Kim, 2021; Ren et al., 2011; Singh et al., 2019). Rata-rata perusahaan pengembang perangkat lunak menghabiskan waktu kerja sebanyak 40% untuk pengembangan tahap awal, dan 60% untuk tahap perawatan setiap satu bulannya (Dehaghani & Hajrahimi, 2013; Jang & Kim, 2021; Ren et al., 2011; Singh et al., 2019). Biaya yang diperlukan untuk perawatan perangkat lunak juga mengalami peningkatan dari 50% menjadi 90% dari keseluruhan biaya pengembangan perangkat lunak pada dua dekade terakhir (Dehaghani & Hajrahimi, 2013; Jang & Kim, 2021; Ren et al., 2011; Singh et al., 2019).

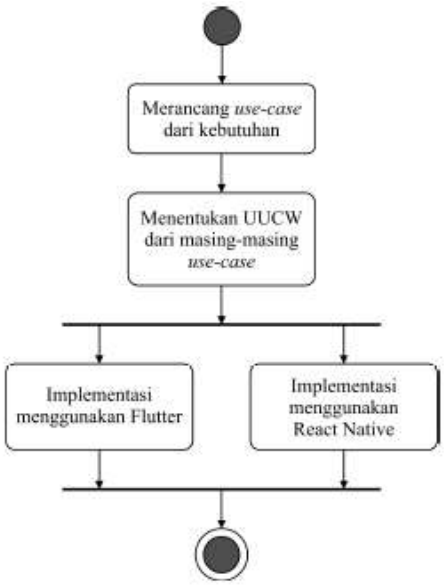
Dari data tersebut, dapat disimpulkan bahwa biaya, waktu, dan usaha untuk mengembangkan suatu aplikasi lebih besar dikeluarkan saat proses perawatan, termasuk aplikasi seluler. Karena itu, teknologi atau *framework* yang tepat untuk meningkatkan *maintainability* suatu aplikasi seluler menjadi sebuah pertimbangan yang penting bagi pengembang. Tujuan yang akan dicapai dari penelitian ini adalah dihasilkannya perbandingan *maintainability* yang objektif antara aplikasi yang dikembangkan menggunakan Flutter dan React Native. Aspek *maintainability* yang diukur dari aplikasi adalah kualitas kode menggunaan metriks *A Code Quality Metrics Model for React-Based Web Applications* (CQMMRBWA) (Lin et al., 2017). Penelitian ini diharapkan dapat membantu industri perangkat lunak yang mengembangkan aplikasi seluler dalam memilih *framework* aplikasi seluler lintas *platform* dengan *maintainability* tertinggi. Dengan pilihan tersebut, diharapkan pengembang dapat mengurangi biaya, waktu, dan usaha perawatan dari aplikasi seluler yang dikembangkan secara signifikan.

2. Metode Penelitian

Penelitian ini terdiri dari 3 tahapan yakni persiapan *prerequisite*, pengukuran *maintainability* dan analisis hasil pengukuran. Setiap tahap dijelaskan pada sub bagian terpisah.

2.1 Persiapan *Prerequisite*

Untuk memulai penelitian ini, prerequisite berupa aplikasi seluler perlu dikembangkan terlebih dahulu. Aplikasi seluler yang dikembangkan adalah aplikasi media komunikasi untuk orang tua dan guru taman kanak kanak yang dikembangkan menggunakan Flutter dan React Native (Arif, 2021). Penelitian ini dilaksanakan saat aplikasi sudah memasuki tahap *software* validation. Tahap *software* specification pada aplikasi ini telah dilakukan sebelumnya, sehingga untuk penelitian ini hanya *software* design and implementation yang perlu dilaksanakan. Diagram aktivitas pengembangan hingga tahap *software* design and implementation dapat dilihat pada Gambar 1.

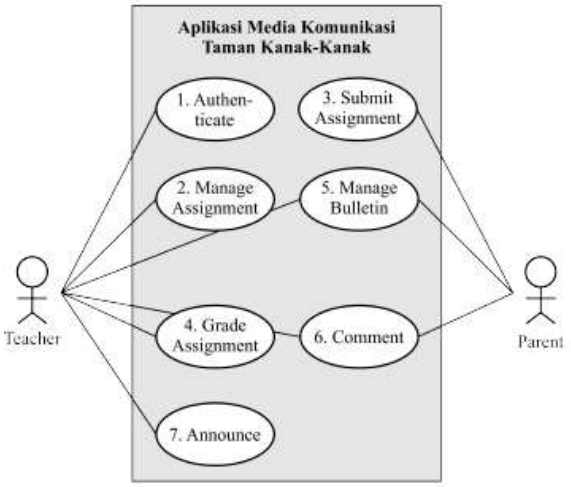


Gambar 1. Diagram aktivitas untuk persiapan *prerequisite*

Use-case diagram, use-case description, dan UUCW dirancang untuk menentukan kebutuhan yang akan dikembangkan dan mengetahui kompleksitas dari masing-masing kebutuhan. Use-case diagram dapat dilihat pada Gambar 2. Pada diagram ini, semua *communication* merupakan hubungan asosiasi. Perhitungan UUCW dapat dilihat pada Tabel 1. Tujuan dari perhitungan UUCW adalah untuk mengetahui keberagaman kompleksitas pada aplikasi yang akan diteliti.

Tabel 1. Perhitungan *unadjusted use-case weight* pada aplikasi *prerequisite*

Use Case	Jumlah Transaksi	Kompleksitas	Bobot
Authenticate	2	Sederhana	5
Manage assignment	9	Kompleks	15
Submit assignment	7	Sedang	10
Grade assignment	6	Sedang	10
Manage bulletin	5	Sedang	10
Comment	4	Sedang	10
Announce	2	Sederhana	5



Gambar 2. *Use-case* diagram untuk aplikasi *prerequisite*

Berdasarkan use-case yang telah dirancang, aplikasi dikembangkan sebanyak dua kali. Pengembangan pertama dilaksanakan menggunakan Flutter, dan kedua menggunakan React Native. Pengembangan dilakukan secara berurutan, dengan aplikasi yang menggunakan Flutter dilaksanakan sebelum aplikasi yang menggunakan React Native. Masing-masing basis kode dikembangkan dengan mengikuti prinsip clean code (Hunter-Zinck et al., 2021; Rachow et al., 2018). Kode antarmuka pengguna dari masing-masing basis kode juga dirancang agar mengikuti jumlah komponen yang sama, jumlah *props* dan state yang sama, dan pemanggilan fungsi eksternal yang sama. Hal ini dapat dilakukan karena Flutter dan React Native memiliki blok pembangun dan prinsip pemrograman yang sama (*Components and Props – React*, n.d.; *Introduction to Declarative UI | Flutter*, n.d.; *Introduction to Widgets | Flutter*, n.d.). Tujuannya adalah agar perbedaan yang tersisa dari kedua basis kode hanya dipengaruhi oleh sifat dari bahasa pemrograman yang dipakai dan sifat dari kedua *framework*.

2.2 Pengukuran *Maintainability*

Kode dari masing-masing aplikasi akan diukur *maintainability*-nya secara kuantitatif menggunakan CQMMRBWA dengan satuan metrik dapat dilihat pada Tabel 2. Masing-masing basis kode akan menghasilkan React-based Web Code Quality Score (RBWCQS) (Lin

et al., 2017). RBWCQS dari basis kode aplikasi yang dikembangkan pada Flutter kemudian dibandingkan dengan yang dikembangkan pada React Native. RBWCQS yang lebih tinggi menandakan kode yang ditulis pada *framework* tersebut memiliki tingkat *maintainability* lebih tinggi. Meskipun CQMMRBWA ditujukan untuk basis kode aplikasi web berbasis React, terdapat alasan berikut yang menyebabkan CQMMRBWA dapat digunakan untuk mengukur basis kode di atas *framework* yang menerapkan pengembangan antarmuka pengguna berbasis komponen secara umum, seperti Flutter dan React Native. Alasannya adalah sebagai berikut:

1. Meski terdapat kata web pada judul metriknya, Tidak ada satuan metrik atau kategori metrik CQMMRBWA yang bergantung pada teknologi web (Lin et al., 2017).
2. CQMMRBWA memiliki dua kategori metrik yang sesuai dengan struktur satuan modul pada *framework* berbasis komponen seperti Flutter dan React Native, yaitu kategori React Components (*Components and Props – React*, n.d.; *Introduction to Widgets | Flutter*, n.d.) dan React States (*Differentiate between Ephemeral State and App State | Flutter*, n.d.; *State and Lifecycle – React*, n.d.).
3. Kategori JavaScript pada CQMMRBWA tidak terbatas pada bahasa pemrograman JavaScript. Kategori tersebut memiliki parameter indentasi, anotasi, penamaan variabel dan fungsi, dan redundansi, yang perlu diperhatikan juga oleh bahasa pemrograman Dart (*Effective Dart | Dart*, n.d.) atau clean code secara umum (Hunter-Zinck et al., 2021; Rachow et al., 2018).

Tabel 2. Satuan metrik CQMMRBWA

Kategori	Satuan Metriks	Perhitungan	Parameter	Aturan Kuantisasi
Java Script	Code indentation ratio	JSCIR = $\frac{line_{cir}}{line_{total}}$	a = 2, b = 1	Logaritmik $y = \log_a x + b$
	Code annotation ratio	JSCAR = $\frac{line_{note}}{line_{total}}$	μ = 0.019	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	Variables and functions correct naming ratio	JSVFCNR = $\frac{V\,F_{cnr}}{V\,F_{total}}$	a = 2, b = 1	Logaritmik $y = \log_a x + b$
	Internal code replication ratio	JSICRR = $1 - \frac{line_{copy}}{line_{total}}$	a = 1, b = 0	Linier $y = ax + b$
React Components	Component line of code	RCLOC = $\frac{\sum_{i=0}^{n} Z_i}{n}$ $Z_i \begin{cases} 1, line_{c_i} \geq 300 \\ 0, line_{c_i} < 300 \end{cases}$	μ = 0.62	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	Component dependent graph leaf node	RCDGLNN = $\frac{\sum_{j=0}^{m} leaf_{CDG_j}}{m}$	μ = 3.65	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	Component dependent graph depth	RCDGD = $\frac{\sum_{j=0}^{m} deep_{CDG_j}}{m}$	μ = 1.84	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	Component functions average cyclomatic complexity	RCFACC = $\frac{\sum_{i=0}^{n}\sum_{k=0}^{l} CC_k}{n}$	μ = 1.2	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	Component coupling factors	RCCF = $\frac{\sum_{i=0}^{n} C_i}{n\,(n-1)}$	μ = 3.55	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	Component self-state ratio	RCSSR = $\sum_{j=0}^{m} \max\,(\frac{CS_j}{total})$	μ = 1	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	Parent component transition state ratio	RPCTSR = $\sum_{j=0}^{m} \max\,(\frac{CP_j}{total})$	μ = 0.93	Linier $y = ax + b$
React States	State utilization ratio	RSUR = $\frac{\sum_{i=0}^{n} \frac{US_i}{TS_i + PS_i}}{n}$	a = 1, b = 0	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	State transmissibility ratio	RSTR = $\frac{\sum_{i=0}^{n} \frac{PS_i}{SS_i}}{n}$	μ = 0.68	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	State transition weight	RSTW = $\frac{\sum_{i=0}^{n} BPS_i + 2\,CPS_i + 3APS_i}{n}$	μ = 8.68	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	State call basic function ratio	RSCBFR = $\frac{\sum_{i=0}^{n} \frac{FB_i}{FA_i}}{n}$	μ = 0.19	Normal $y = \frac{1}{e^{(x-\mu)^2}}$
	State required ratio	RSRR = $\frac{\sum_{i=0}^{n} \frac{RS_i}{AS_i}}{n}$	μ = 0.84	Normal $y = \frac{1}{e^{(x-\mu)^2}}$

Keterangan:
$line_{cir}$ = jumlah baris kode dengan format indentasi yang benar
$line_{note}$ = jumlah baris kode dengan anotasi
$line_{copy}$ = jumlah baris kode yang redundan
$line_{total}$ = jumlah seluruh baris kode
$line_{c_i}$= jumlah baris kode pada komponen i
$V\ F_{cir}$ = jumlah variabel dan fungsi dengan konvensi penamaan yang benar
$V\ F_{total}$ = jumlah variabel dan fungsi
$leaf_{CDG_j}$= jumlah daun pada graf j
$deep_{CDG_j}$= kedalaman graf j
CC_k = jumlah *cyclomatic complexity* pada fungsi k
C_i = jumlah dependensi antara komponen i dengan komponen lain
$\max\left(\frac{CS_j}{total}\right)$= *self-state ratio* maksimal pada komponen di graf j
$\max\left(\frac{CP_j}{total}\right)$= *parent component transition state* ratio *j*maksimal pada komponen di graf j
US_i = jumlah *used state* pada komponen i
TS_i = jumlah *self state* pada komponen i
PS_i = jumlah *state* yang dikirim oleh *parent* komponen i ke komponen i
SS_i = jumlah *state* yang dikirim oleh komponen i ke *parent* komponen i
BPS_i = jumlah *state* sederhana yang dikirim oleh *parent* komponen i ke komponen i
CPS_i = jumlah *state* kompleks yang dikirim oleh *parent* komponen i ke komponen i
APS_i = jumlah *state* dinamis yang dikirim oleh *parent* komponen i ke komponen i
FB_i = jumlah pemanggilan fungsi publik pada komponen i
FA_i = jumlah fungsi *member* pada komponen i
RS_i = jumlah *state* wajib yang dikirim oleh *parent* komponen i ke komponen i
AS_i = jumlah *state* yang dikirim oleh *parent* komponen i ke komponen i
n = jumlah komponen
m = jumlah *complete component dependent graphs*
l = jumlah fungsi pada komponen i

Setiap satuan metrik pada Tabel 2 dihitung menggunakan persamaan pada kolom persamaan. Hasil perhitungan setiap satuan metrik kemudian dihitung dengan aturan kuantisasi masing- masing dalam bentuk rumus $y = f(x)$. Dimana, x adalah nilai yang didapat melalui perhitungan di Tabel 2 dan f adalah fungsi kuantisasi seperti pada kolom aturan kuantisasi. Hasil kuantisasi ini kemudian dirata-rata untuk tiap kategori sehingga didapatkan nilai $RWCQ_{JS}$, $RWCQ_C$, , $RWCQ_S$.

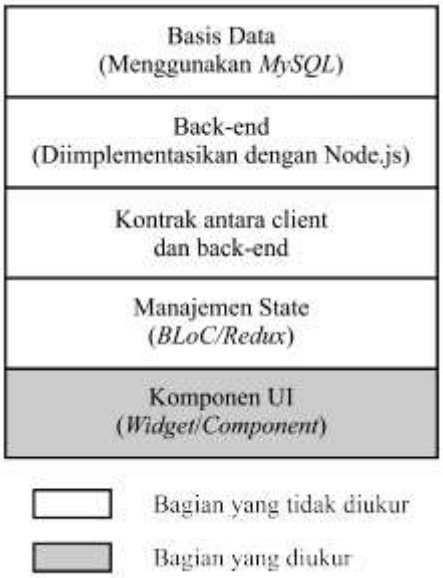
Agar kode-kode dapat diukur menggunakan CQMMRBWA, konversi kode menjadi parameter yang sesuai dengan kebutuhan metrik harus dilakukan terlebih dahulu. Proses konversi kode dilakukan dengan tahap-tahap berikut:

1. Membuat pseudo-code abstrak untuk setiap komponen yang diturunkan dari basis kode komponen pada Flutter dan React Native. Tahap ini dilakukan menggunakan metode transpilasi (*Transpiler*, n.d.) secara manual. Pseudo-code yang dihasilkan mengandung fitur-fitur yang terdapat pada kedua *framework*, seperti props, states, global dispatcher, agar menghasilkan pseudo-code yang merepresentasikan kedua *framework* tersebut.
2. Tokenisasi dengan hasil token-token berbentuk JSON dari pseudo-code yang dibuat. Tahap ini dilakukan dengan metode analisis leksikal (*Anatomy of a Compiler and The Tokenizer*, n.d.) secara manual.
3. Mengkonversi token-token JSON yang didapat pada tahap sebelumnya menjadi analisis komponen dan graf dependency antar komponen. Tahap ini dilaksanakan secara otomatis menggunakan *automation script*.
4. Mengkonversi hasil analisis komponen dan graf dependency menjadi parameter untuk perhitungan metrik.

Tahap ini dilaksanakan secara otomatis menggunakan *automation script*. Konversi kode-kode komponen menjadi parameter dilakukan secara manual dan otomatis. Konversi manual dilakukan dengan cara menganalisis setiap baris kode yang ada pada file komponen dan menentukan parameternya baris per baris. Cara tersebut mudah untuk dilakukan dan tidak memerlukan alat tambahan. Namun cara ini membutuhkan waktu dan tenaga yang besar serta rentan terhadap human error. Sedangkan cara yang otomatis dapat dilakukan dengan menggunakan perangkat lunak otomasi untuk mendeteksi *maintainability* seperti SonarQube (*Maintainability = Productivity | SonarQube*, n.d.). Cara ini lebih reliabel dan cepat dibandingkan dengan cara manual. Namun, perlu waktu yang cukup lama untuk mempelajari dan mengintegrasi basis kode pada alat otomasi. Selebih lagi, bahasa pemrograman yang digunakan Flutter, yaitu Dart, belum memiliki perangkat lunak otomasi yang kompatibel saat penelitian ini dilakukan. Untuk tahapan konversi otomatis, *automation script* khusus untuk penelitian ini perlu dikembangkan terlebih dahulu.

Bagian dari kode yang diukur adalah bagian antarmuka pengguna yang masing-masing dikembangkan menggunakan Flutter

dan React Native. Bagian tersebut mencakup seluruh *widget* atau component yang terdapat pada aplikasi. Gambaran dari arsitektur kedua aplikasi serta bagian yang diukur pada penelitian ini dapat dilihat pada Gambar 3.



Gambar 3. Arsitektur dari aplikasi *prerequisite* dan bagian yang diukur pada penelitian ini

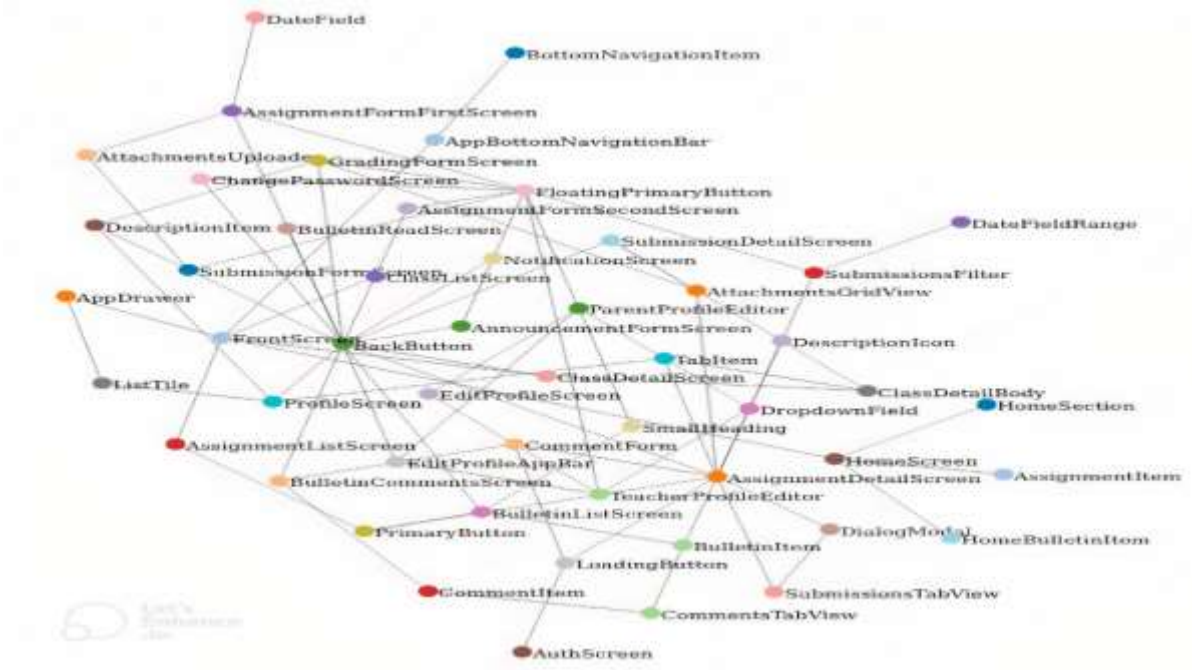
2.3 Analisis Hasil Pengukuran

Setelah RBWCQS dari basis kode yang ditulis pada Flutter dan React Native didapatkan, kedua nilai tersebut akan dibandingkan secara kuantitatif. Nilai yang lebih besar menandakan bahwa *framework* tersebut menghasilkan kode yang memiliki *maintainability* lebih tinggi. Pada tahap ini, alasan yang menyebabkan mengapa salah satu kode *framework* menghasilkan skor yang berbeda atau sama dengan yang lain akan dibahas secara terperinci.

3. Hasil dan Pembahasan

Aplikasi *prerequisite* yang dikembangkan untuk penelitian ini menghasilkan 51 komponen antarmuka pengguna. Dependency antar komponen membentuk graf yang divisualisasikan pada Gambar 4. Karena berdasar pada kebutuhan yang sama, aplikasi yang ditulis dengan Flutter dan React Native memiliki komponen-komponen yang sama.

Sebelum memulai konversi kode, perlu diperhatikan bahwa beberapa satuan metrik berbagi parameter satu sama lainnya. Contohnya, parameter seluruh baris kode dibutuhkan pada satuan metrik JSCIR, JSCAR, dan JSICRR. Untuk mempermudah perhitungan, parameter yang dibagikan pada lebih dari satu satuan metrik dihitung satu kali saja. Semua parameter tanpa redundansi untuk perhitungan CQMMRBWA dapat dilihat pada Tabel 3. Nama parameter menunjukkan sumber masing-masing parameter, parameter berakhiran `PER_KOMPONEN` bersumber dari setiap komponen, `PER_GRAPH` bersumber dari setiap *dependency* graph, dan yang tanpa akhiran bersumber dari seluruh komponen. Semua parameter menggunakan tipe data integer kecuali parameter yang menunjukkan perbandingan seperti MSR dan MPR yang menggunakan float. Setelah parameter ditentukan, proses konversi kode dan penulisan *automation script* dapat dimulai.



Gambar 4. Visualisasi graf *dependency* antar komponen

Tahap pertama konversi kode adalah membuat *pseudo-code* berdasarkan komponen-komponen yang telah dikembangkan. Pseudo-code ditulis dengan paradigma pemrograman deklaratif dan functional. Tujuannya adalah agar *pseudo-code* tersebut netral terhadap bahasa pemrograman Dart dan JavaScript, namun sesuai dengan basis dari pemrograman antarmuka pengguna berbasis komponen yang digunakan oleh Flutter dan React Native. Kelemahan dari *pseudo-code* ini adalah ketidakmampuannya untuk dikompilasi atau diinterpretasi secara otomatis. Oleh karena itu, tokenisasi atau analisis leksikal [10] dilakukan secara manual pada setiap baris *pseudo-code* tersebut dengan meniru perilaku *tokenizer* pada *compiler*. Hasil dari tokenisasi tersebut adalah token atau satuan leksikal dalam bentuk dokumen JSON. Daftar satuan leksikal yang digunakan pada proses ini dapat dilihat pada Tabel 4.

Tabel 3. Parameter Tanpa Redundansi

Parameter	Deskripsi
LOC_TOTAL	Baris kode
LOC_CORRECT_INDENT	Baris kode dengan indentasi benar
LOC_ANNOTATION	Baris kode komentar
LOC_REDUNDANT	Baris kode yang redundan
VF_TOTAL	Total variabel dan fungsi
VF_CORRECT_NAMING	Total variabel dan fungsi dengan penamaan benar
LOC_PER_COMP	Baris kode per komponen
CC_PER_F_PER_COMP	*Cyclomatic complexity* per fungsi per komponen
DEPS_PER_COMP	*Dependency* komponen lain per komponen
LEAVES_PER_GRAPHS	Jumlah daun pada setiap *dependency graphs*
DEPTH_PER_GRAPHS	Kedalaman setiap *dependency graphs*
MSR_PER_GRAPHS	Perbandingan *state* tertinggi pada setiap *dependency graphs*
MPR_PER_GRAPHS	Perbandingan *Props* tertinggi pada setiap *dependency graphs*
DATA_PER_COMP	*Props* dan *state*s yang ter pakai pada setiap komponen
STATES_PER_COMP	*state* pada setiap komponen
PROPS_PER_COMP	*Props* pada setiap komponen
SENT_PER_COMP	*Props* dan *state*s per komponen yang dikirimkan ke komponen lain
S_PROPS_PER_COMP	*Props* pada setiap komponen dengan tipe data primitif
C_PROPS_PER_COMP	*Props* pada setiap komponen dengan tipe data kompleks
D_PROPS_PER_COMP	*Props* pada setiap komponen dengan tipe data dinamis
F_CALL_PER_COMP	Pemanggilan fungsi per Komponen
F_OWN_PER_COMP	Kepemilikan method per komponen
REQ_PROPS_PER_COMP	Banyaknya *Props* wajib pada setiap komponen

Pada tahap selanjutnya, konversi dilakukan secara otomatis menggunakan *automation script*. *Automation script* dibuat agar dokumen token JSON dapat dikonversi menjadi skor metrik RBWCQS dengan cepat dan reliabel. *Automation script* tersebut dibuat menggunakan JavaScript. Proses otomasi ini dibagi menjadi empat bagian, yaitu analisis token, penentuan graf dan subgraf dependency antar komponen, konversi analisis dan graf ke parameter pada Tabel 3, dan konversi parameter ke skor.

Tabel 4. Satuan Leksikal Tokenisasi Manual

Satuan	Keterangan
COMPONENT_SHELL_STATELESS	Komponen tidak memiliki *local state*
COMPONENT_SHELL_STATEFUL	Komponen memiliki *local state*
COMPONENT_DEPENDENCY	Komponen bergantung pada komponen lain
OTHER_DEPENDENCY	Kebergantungan pada non-komponen (fungsi, *class*, konstanta, atau tipe data)
PROPS	Pendefinisian *props*
STATE	Pendefinisian *local state*
SET_STATE	Permintaan mutasi *local state*
STORE	Pengikatan komponen dengan *global state*
DISPATCH	Referensi pada fungsi *global dispatcher*
SUBSCRIBE	Langganan kepada *global state*
MOUNTED	Aksi yang dilaksanakan saat komponen pertama ditampilkan
COMPONENT_CALL	Pemanggilan komponen lain
IF_EXPRESSION	Expresi *if*
WHEN_EXPRESSION	Expresi *pattern matching*, bersifat seperti *switch-case*
EXPRESSION	Expresi lainnya

LAMBDA_EXPRESSION	Expresi lambda atau fungsi anonim
VARIABLE_DECLARATION	Deklarasi variabel
EXTERNAL_FUNCTION_CALL	Pemanggilan fungsi di luar komponen
PASS_STATE	Pemberian *local state* pada komponen *children*
PASS_PROPS	Pemberian *props* pada komponen *children*
PASS_STORE	Pemberian global state pada komponen *children*

Pada bagian pertama, token-token pada setiap dokumen token JSON diubah menjadi array dari objek analisis. Jumlah elemen pada array tersebut sama dengan jumlah komponen, yaitu sebanyak 51 elemen. Objek analisis memiliki skema yang dapat dilihat pada Tabel 5. Pada bagian selanjutnya, graf dependency diambil dengan menganalisis properti componentDependencies pada setiap objek analisis. Subgraf ditentukan dari komponen sink, yaitu komponen yang tidak dipakai sebagai dependency, hingga source, yaitu komponen yang tidak memiliki dependency. Banyaknya subgraf ditentukan dari banyaknya sink pada graf dependency, yaitu sebanyak 19 subgraf. Skema objek subgraf dapat dilihat pada Tabel 6. Bagian selanjutnya adalah konversi analisis dan graf menjadi parameter. Tahap ini dilakukan dengan memetakan kedua nilai tersebut dengan parameter yang berhubungan. Tabel pemetaan dapat dilihat di Tabel 7 hingga Tabel 9.

Perhitungan skor dilakukan secara otomatis menggunakan *automation script* yang sama pada tahap sebelumnya. Hasil perhitungan skor untuk Flutter dan React Native dapat dilihat pada Tabel 10. Dari kedua skor RBWCQS tersebut, dapat disimpulkan bahwa basis kode yang ditulis pada Flutter memiliki kualitas kode lebih tinggi dibandingkan React Native berdasarkan CQMMRBWA, dan basis kode Flutter lebih mudah untuk dirawat dibandingkan React Native.

Pada Tabel 10, perbedaan skor disebabkan oleh perbedaan skor RSTW, RSCBFR, dan RSRR. Skor RSTW Flutter lebih kecil dibandingkan milik React Native dengan jarak 0.2799, skor RSCBFR milik Flutter lebih besar dibandingkan milik React Native dengan jarak 0.5743, sedangkan skor RSRR Flutter lebih besar dibandingkan React Native dengan jarak 0.427. RSTW ditentukan oleh jumlah *props* sederhana, kompleks, dan dinamis, RSCBFR ditentukan oleh jumlah kepemilikan dan pemanggilan fungsi pada setiap komponen, sedangkan RSRR ditentukan oleh jumlah *props* yang wajib diberikan.

Pada Flutter, tipe data pada *props* dapat ditentukan sama halnya seperti variabel pada Dart. Ini menyebabkan komponen, atau *widget*, dapat mencegah kesalahan data yang dikirim saat pemanggilan komponen. Contohnya, komponen Text milik Flutter hanya menerima tipe data String sebagai tipe data *props* utamanya. Jika terdapat pemanggilan Text dengan memberikan tipe data selain String, contohnya Integer, maka compiler Flutter tidak akan melanjutkan proses kompilasi. Namun, hal ini membuat komponen pada Flutter menjadi kaku dan sulit untuk digunakan ulang. Berbeda dengan React Native, setiap komponennya memiliki *props* dengan tipe data yang dinamis. Hal ini mempermudah komponen untuk digunakan kembali. Namun, pemanggilan komponen menjadi tidak aman dan rentan terhadap runtime error. Meski React Native memiliki fitur opsional untuk memeriksa tipe data untuk *props* [35], fitur tersebut hanya memberikan pesan error saat runtime jika terdapat tipe data *props* yang tidak sesuai. Hal ini merupakan pengaruh dari bahasa pemrograman Dart dan JavaScript, yang masing-masing merupakan bahasa pemrograman statically typed [11] dan dynamically typed [24].

Tabel 5. Skema objek analisis komponen

Properti	Deskripsi
name	Nama dari komponen
locTotal	Jumlah baris kode
locCorrectIndentTotal	Jumlah baris kode dengan indentasi benar
locRedundantTotal	Jumlah baris kode yang redundan
name	Jumlah baris kode jika dipanggil pada komponen lain
locAnnotation	Jumlah baris kode komentar
varsAndFuncs	Jumlah variabel dan fungsi
varsAndFuncsCorrect	Jumlah variabel dan fungsi dengan penamaan yang benar
cyclomaticComplexity	*Cyclomatic complexity* pada komponen
componentDependencies	*Dependency* komponen lain, diidentifikasi dengan nama komponen
props	*Props* yang dimiliki komponen. Objek terdiri atas nama, *flag* wajib, *flag children*, kompleksitas, dan jumlah pengiriman ke komponen *children*
states	*Local states* yang dimiliki komponen. Objek terdiri atas nama, flag digunakannya state pada komponen `TextField`, dan jumlah pengiriman ke komponen *children*
stores	*Global states* yang digunakan oleh komponen. Objek terdiri atas

	nama, nama *parent*, kompleksitas, dan jumlah pengiriman ke komponen *children*
initStateOverriden	*Flag* yang menandakan apakah fungsi *mounted* sudah tergantikan
disposeOverriden	*Flag* yang menandakan apakah fungsi *dispose* sudah tergantikan
isStoreLibraryImported	*Flag* yang menandakan apakah *library* untuk *global state* sudah dipanggil
storeParents	Nama-nama *parent* dari *global state*
dispatchScopes	Nama-nama *parent* dari *global dispatcher*
externalFunctionCalls	Jumlah pemanggilan fungsi di luar komponen
memberFunctions	Jumlah fungsi milik komponen

Pada CQMMRBWA, satuan metrik RSTW dihitung menggunakan fungsi kuantisasi normal. Fungsi ini akan menghasilkan nilai terbaik jika parameter mendekati nilai optimal [19]. Jika dilihat dari hasil perhitungan RSTW pada penelitian ini, basis kode Flutter menghasilkan skor lebih kecil karena memiliki mayoritas *props* bertipe kompleks, sedangkan React Native lebih besar karena mayoritas *props* bertipe dinamis. Sehingga, dapat disimpulkan bahwa jumlah *props* yang jatuh pada kategori dinamis berbanding lurus dengan nilai optimal untuk satuan metrik RSTW.

Flutter dan React Native memiliki cara yang berbeda dalam mendefinisikan komponen. Dengan Flutter, komponen didefinisikan sebagai class, sedangkan komponen React Native didefinisikan sebagai fungsi. Hal ini menyebabkan komponen pada React Native akan selalu memiliki satu fungsi utama, sedangkan Flutter dapat memiliki lebih dari satu. Minimalnya, komponen tanpa state pada Flutter memiliki 1 fungsi method saja, sedangkan komponen yang memiliki state memiliki 3 fungsi method. Selain itu, React Native memerlukan pemanggilan pada fungsi hooks [33] untuk mengakses local state, global state, dan global dispatcher.

Tabel 6. Skema objek subgraf *dependency*

Properti	Deskripsi
root	Komponen yang bertindak sebagai *sink* atau *root*
leaves	Komponen-komponen yang bertindak sebagai *source* atau *leaf*
depth	Kedalaman subgraf yang dihitung dari *root* hingga *leaf* terdalam.
maxStateRatio	Perbandingan tertinggi *state* komponen dengan seluruh *state* pada subgraf
maxPropsRatio	Perbandingan tertinggi *props* komponen dengan seluruh *props* pada subgraf

Satuan metrik RSCBFR pada CQMMRBWA dihitung dengan perbandingan jumlah pemanggilan fungsi dengan jumlah kepemilikan fungsi pada setiap komponen [19]. Fungsi kuantisasi yang digunakan adalah fungsi kuantisasi normal, sehingga nilai terbaik adalah nilai yang mendekati nilai optimal. Pada penelitian ini, skor RSCBFR lebih tinggi pada Flutter dibandingkan React Native, yang memiliki jumlah memiliki fungsi lebih banyak dan memanggil fungsi lebih sedikit dibandingkan React Native. Dapat disimpulkan bahwa nilai RSCBFR mendekati optimal jika jumlah fungsi kepemilikan lebih banyak dan pemanggilan fungsi lebih sedikit.

Tabel 7. Pemetaan Token ke Analisis Komponen dan Subgraf *Dependency*

Token	Komponen dan Subgraf *Dependency*
COMPONENT_SHELL_STATELESS	memberFunctions, varsAndFuncs, locTotal, root
COMPONENT_SHELL_STATEFUL	memberFunctions, varsAndFuncs, locTotal, root
COMPONENT_DEPENDENCY	componentDependencies, locTotal, root, leaves, depth
OTHER_DEPENDENCY	locTotal
PROPS	locTotal, props, locCallTotal, varsAndFuncs, maxPropsRatio
STATE	locTotal, initStateOverriden, disposeOverriden, states, varsAndFuncs, memberFunctions, externalFunctionCalls, maxStateRatio
SET_STATE	locTotal, varsAndFuncs
STORE	locTotal, isStoreLibraryImported, stores, storeParents, varsAndFuncs, externalFunctionCalls
DISPATCH	locTotal, isStoreLibraryImported, dispatchScopes, varsAndFuncs, externalFunctionCalls
SUBSCRIBE	locTotal, isStoreLibraryImported,

MOUNTED	storeParents, varsAndFuncs locTotal, initStateOverriden, varsAndFuncs, externalFunctionCalls
COMPONENT_CALL	locTotal
IF_EXPRESSION	locTotal, cyclomaticComplexity
WHEN_EXPRESSION	locTotal, cyclomaticComplexity
EXPRESSION	locTotal
LAMBDA_EXPRESSION	locTotal, varsAndFuncs
VARIABLE_DECLARATION	locTotal, varsAndFuncs
EXTERNAL_FUNCTION_CALL	locTotal, externalFunctionCalls
PASS_STATE	locTotal, states
PASS_PROPS	locTotal, props
PASS_STORE	locTotal, store

Tabel 8. Pemetaan Analisis Komponen dan Subgraf *Dependency* ke Parameter

Use Case	Jumlah Transaksi
name	-
locTotal	LOC_TOTAL, LOC_PER_COMP
locCorrectIndentTotal	LOC_CORRECT_INDENT
locRedundantTotal	LOC_REDUNDANT
locCallTotal	-
locAnnotation	LOC_ANNOTATION
varsAndFuncs	VF_TOTAL
varsAndFuncsCorrect	VF_CORRECT_NAMING
cyclomaticComplexity	CC_PER_F_PER_COMP
componentDependencies	DEPS_PER_COMP
props	DATA_PER_COMP, SENT_PER_COMP, PROPS_PER_COMP, S_PROPS_PER_COMP, C_PROPS_PER_COMP, D_PROPS_PER_COMP, REQ_PROPS_PER_COMP
states	DATA_PER_COMP, STATES_PER_COMP, SENT_PER_COMP
stores	DATA_PER_COMP, SENT_PER_COMP
initStateOverriden	-
disposeOverriden	-
isStoreLibraryImported	-
storeParents	-
dispatchScopes	-
externalFunctionCalls	F_CALL_PER_COMP
memberFunctions	F_OWN_PER_COMP
root	-
leaves	LEAVES_PER_GRAPHS
depth	DEPTH_PER_GRAPHS
maxStateRatio	MSR_PER_GRAPHS
maxPropsRatio	MPR_PER_GRAPHS
leaves	LEAVES_PER_GRAPHS
depth	DEPTH_PER_GRAPHS
maxStateRatio	MSR_PER_GRAPHS

Berkat bahasa pemrograman Dart, Flutter mengizinkan pengembang untuk mengontrol kewajiban pemberian *props* pada setiap komponen. Pada dasarnya, semua *props* pada Flutter adalah named parameter. Named parameter bersifat opsional kecuali yang ditandai kata kunci required. Kemampuan tersebut dapat dilakukan pada React Native menggunakan PropTypes. Tetapi, PropTypes hanya memberikan error saat runtime. Sehingga pada dasarnya setiap *props* pada komponen React Native bersifat opsional. Sama seperti RSTW, hal ini disebabkan oleh perbedaan sifat bahasa pemrograman Dart yang *statically typ*e dan JavaScript yang *dynamically typed*.

Tabel 9. Pemetaan Parameter ke Metriks CQMMRBWA

Parameter	Metriks
LOC_TOTAL	JSCIR, JSCAR, JSICRR
LOC_CORRECT_INDENT	JSCIR
LOC_ANNOTATION	JSCAR
LOC_REDUNDANT	JSICRR
VF_TOTAL	JSVFNCR
VF_CORRECT_NAMING	JSVFNCR
LOC_PER_COMP	RCLOC
CC_PER_F_PER_COMP	RCFACC
DEPS_PER_COMP	RCCF
LEAVES_PER_GRAPHS	RCDGLNN
DEPTH_PER_GRAPHS	RCDGD
MSR_PER_GRAPHS	RCSSR
MPR_PER_GRAPHS	RPCTSR
DATA_PER_COMP	RSUR
STATES_PER_COMP	RSUR
PROPS_PER_COMP	RSUR, RSTR, RSRR
SENT_PER_COMP	RSTR
S_PROPS_PER_COMP	RSTW
C_PROPS_PER_COMP	RSTW
D_PROPS_PER_COMP	RSTW
F_CALL_PER_COMP	RSCBFR
F_OWN_PER_COMP	RSCBFR
REQ_PROPS_PER_COMP	RSRR

Tabel 10. Skor RBWCQS Flutter dan React Native menggunakan CQMM- RBWA

Metriks	Flutter	React Native
JSCIR	1.0000	1.0000
JSCAR	0.9996	0.9996
JSVFCNR	1.0000	1.0000
JSICRR	1.0000	1.0000
RCLOC	0.6809	0.6809
RCDGLNN	0.9238	0.9238
RCDGD	0.9575	0.9575
RCFACC	0.9352	0.9352
RCCF	0.0176	0.0176
RCSSR	1.0000	1.0000
RPCTSR	0.9951	0.9951
RSUR	1.0000	1.0000
RSTR	0.9996	0.9996
RSTW	0.2087	0.4886
RSCBFR	0.5744	0.0001
RSRR	0.9537	0.5266
$RWCQ_{JS}$	0.9999	0.9999
$RWCQ_C$	0.7030	0.7030
$RWCQ_S$	0.8188	0.7157
RWCQ	**0.8333**	**0.7972**

Pada CQMMRBWA, satuan metrik RSRR ditentukan dengan jumlah dari setiap perbandingan *props* yang bersifat wajib dengan seluruh *props* pada setiap komponen. Aturan kuantisasi yang digunakan oleh RSRR adalah kuantisasi normal, sehingga nilai tertinggi adalah nilai yang mendekati optimal. Jika dilihat pada hasil perhitungan di penelitian ini, nilai RSRR pada Flutter lebih tinggi dibandingkan React Native. Komponen Flutter memiliki lebih banyak *props* wajib dibandingkan React Native. Dari kedua pernyataan tersebut, disimpulkan bahwa nilai RSRR yang optimal didapatkan jika jumlah *props* wajib lebih banyak.

Skor RWCQJS, RWCQC, dan beberapa skor satuan metrik pada kategori RWCQS bernilai sama antara basis kode Flutter dan React Native. Hal ini disebabkan oleh kedua basis kode mengikuti komponen yang sama, yang grafnya diilustrasikan pada Gambar 4. Akibatnya, nama komponen, banyaknya komponen, dan hubungan antara komponen pada basis kode Flutter dan React Native tidak memiliki perbedaan. Alasan ini mempengaruhi kesamaan skor RWCQC dan beberapa skor satuan metrik RWCQS. Selain itu, terdapat juga skor dengan nilai ekstrim atau 1 seperti JSCIR, JSVFCNR, JSICRR, RCSSR, dan RSUR. Hal ini dipengaruhi oleh prinsip clean code yang diikuti oleh basis kode Flutter dan React Native. Sehingga, kedua basis kode memiliki redundansi minimal, penamaan variabel dan fungsi yang benar, dan utilisasi variabel yang maksimal.

4. Kesimpulan

Penelitian ini telah dilaksanakan dengan luaran aplikasi media komunikasi orang tua dan guru taman kanak-kanak yang ditulis menggunakan Flutter dan React Native, *automation script* untuk menganalisis basis kode Flutter dan React Native, hasil perhitungan skor RBWCQS antara kedua basis kode tersebut menggunakan CQMMRBWA, dan *framework* yang menghasilkan skor lebih tinggi, yaitu Flutter. Sebanyak 51 komponen antarmuka pengguna telah dikembangkan menggunakan Flutter dan React Native. Masing-masing komponen pada masing-masing basis kode diabstrakkan menjadi pseudo-code, dianalisis, dikonversi menjadi parameter, dan skor RBWCQS dihitung menggunakan metrik CQMMRBWA. Hasilnya, skor RBWCQS akhir Flutter dan React Native masing-masing adalah 0.8333 dan 0.7972. Skor Flutter yang lebih tinggi menandakan bahwa basis kode Flutter memiliki kualitas lebih tinggi dan lebih maintainable dibandingkan basis kode React Native. Faktor pertama yang mempengaruhinya adalah sifat typing system dari bahasa pemrograman yang digunakan oleh Flutter dan React Native, yaitu Dart dan JavaScript. Faktor kedua adalah sifat dari komponen Flutter yang cenderung memiliki fungsi lebih banyak dan memanggil fungsi lebih sedikit dibandingkan komponen React Native.pengembangan hasil penelitian dan prospek penerapan studi lanjutan.

Aspek *maintainability* selain kualitas kode tidak diukur pada penelitian ini. Untuk penelitian selanjutnya, sebaiknya aspek-aspek lain pada *maintainability* diukur menggunakan metrik yang sesuai. CQMMRBWA hanya mampu mengukur kualitas kode pada lapisan antarmuka pengguna, sehingga bagian lain dari aplikasi tidak tersentuh sama sekali. Hal ini sebaiknya dipertimbangkan untuk diukur pada penelitian selanjutnya. Untuk hasil yang lebih akurat, aplikasi yang diteliti sebaiknya dikembangkan oleh tim pengembang yang berbeda dengan kemampuan yang setara. Proses tokenisasi pseudo-code sebaiknya dilakukan menggunakan interpreter untuk mengurangi human error. Kemudian, pseudo-code yang dibuat pada penelitian ini dapat dikembangkan menjadi domain specific language khusus untuk mengembangkan antarmuka pengguna berbasis komponen.

Referensi

Anatomy of a Compiler and The Tokenizer. (n.d.). Retrieved May 21, 2022, from https://www.cs.man.ac.uk/~pjj/farrell/comp3.html

Arif. (2021, July 23). *Aplikasi CERIA Bantu*

Belajar Daring Lebih Efektif. https://channel-indonesia.com/pendidikan/2021/07/23/aplikasi-ceria-bantu-belajar-daring-lebih-efektif/

Components and Props – React. (n.d.). Retrieved December 28, 2021, from https://reactjs.org/docs/components-and-props.html

Dehaghani, S. M. H., & Hajrahimi, N. (2013). Which factors affect software projects maintenance cost more? *Acta Informatica Medica*, *21*(1), 63–66. https://doi.org/10.5455/aim.2012.21.63-66

Differentiate between ephemeral state and app state | Flutter. (n.d.). Retrieved December 28, 2021, from https://docs.flutter.dev/development/data-and-backend/state-mgmt/ephemeral-vs-app

Effective Dart | Dart. (n.d.). Retrieved May 21, 2022, from https://dart.dev/guides/language/effective-dart

FAQ - Flutter. (n.d.). Retrieved October 19, 2020, from https://flutter.dev/docs/resources/faq#why-did-flutter-choose-to-use-dart

GitHub - facebook/react-native: A framework for building native applications using React. (n.d.). Retrieved December 28, 2021, from https://github.com/facebook/react-native

GitHub - flutter/flutter: Flutter makes it easy and fast to build beautiful apps for mobile and beyond. (n.d.). Retrieved December 28, 2021, from https://github.com/flutter/flutter

GitHub - ionic-team/ionic-framework: A powerful cross-platform UI toolkit for building native-quality iOS, Android, and Progressive Web Apps with HTML, CSS, and JavaScript. (n.d.). Retrieved December 28, 2021, from https://github.com/ionic-team/ionic-framework

GitHub - xamarin/Xamarin.Forms: Xamarin.Forms Official Home. (n.d.). Retrieved December 28, 2021, from https://github.com/xamarin/Xamarin.Forms

Hunter-Zinck, H., De Siqueira, A. F., Vásquez, V. N., Barnes, R., & Martinez, C. C. (2021). Ten simple rules on writing clean and reliable open-source scientific software. *PLOS Computational Biology*, *17*(11), e1009481. https://doi.org/10.1371/JOURNAL.PCBI.1009481

Introduction to declarative UI - Flutter. (n.d.). Retrieved October 19, 2020, from https://flutter.dev/docs/get-started/flutter-for/declarative

Introduction to declarative UI | Flutter. (n.d.). Retrieved December 28, 2021, from https://docs.flutter.dev/get-started/flutter-for/declarative

Introduction to widgets - Flutter. (n.d.). Retrieved October 19, 2020, from https://flutter.dev/docs/development/ui/widgets-intro

Introduction to widgets | Flutter. (n.d.). Retrieved December 28, 2021, from https://docs.flutter.dev/development/ui/widgets-intro

ISO - ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. (n.d.). Retrieved May 21, 2022, from https://www.iso.org/standard/35733.html

J.Clement. (2019). *Annual number of mobile app downloads worldwide 2020*. Statista. https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/

Jagiełło, J. (2019). *PERFORMANCE COMPARISON BETWEEN REACT NATIVE AND FLUTTER*. UMEA University.

Jang, K. ae, & Kim, W. J. (2021). A method of activity-based software maintenance cost estimation for package software. *Journal of Supercomputing*, *77*(8), 8151–8171. https://doi.org/10.1007/s11227-020-03610-6

JavaScript Environment · React Native. (n.d.). Retrieved December 28, 2021, from https://reactnative.dev/docs/javascript-environment

Lin, Y., Li, M., Yang, C., & Yin, C. (2017). A code quality metrics model for react-based web applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10363 LNAI*, 215–226. https://doi.org/10.1007/978-3-319-63315-2_19

Maintainability = productivity | SonarQube. (n.d.). Retrieved May 21, 2022, from https://www.sonarqube.org/features/maintainability/

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship* (1st ed.). Prentice Hall PTR.

Rachow, P., Schroder, S., & Riebisch, M. (2018). Missing clean code acceptance and support in practice - An empirical study. *Proceedings - 25th Australasian Software Engineering Conference, ASWEC 2018*, 131–140.

https://doi.org/10.1109/ASWEC.2018.00026

Rahul Raj, C. P., & Tolety, S. B. (2012). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *2012 Annual IEEE India Conference, INDICON 2012*, 625–629. https://doi.org/10.1109/INDCON.2012.6420693

Ren, Y., Xing, T., Chen, X., & Chai, X. (2011). Research on software maintenance cost of influence factor analysis and estimation method. *2011 3rd International Workshop on Intelligent Systems and Applications, ISA 2011 - Proceedings*. https://doi.org/10.1109/ISA.2011.5873461

S.O'Dea. (2020, October). *Mobile OS market share 2021*. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

Singh, C., Sharma, N., & Kumar, N. (2019). Analysis of software maintenance cost affecting factors and estimation models. *International Journal of Scientific and Technology Research*, *8*(9), 276–281.

Sommerville, I. (2016). *Software Engineering Tenth Edition*. Pearson.

State and Lifecycle – React. (n.d.). Retrieved December 28, 2021, from https://reactjs.org/docs/state-and-lifecycle.html

Transpiler. (n.d.). Retrieved May 21, 2022, from https://devopedia.org/transpiler

Wu, W. (2018). *React Native vs Flutter, Cross-platforms mobile application frameworks*.