BAB 2
TINJAUAN PUSTAKA

2.1 Profil RSIA Bunda Nanda

RSIA "Bunda Nanda" adalah unit layanan kesehatan khusus ibu dan anak tipe "C". RSIA "Bunda Nanda" berlokasi di Jl. Raya Cipadung No.35-39 RT 01/RW 02, Kelurahan Cipadung, Kecamatan Cibiru, Kota Bandung. Layanan kesehatan rawat inap dan rawat jalan didukung oleh tenaga medis yang berpengalaman dengan berbagai spesialisasi seperti spesialis Kandungan, Bedah, Jantung & Pembuluh Darah, Kulit & Kelamin, Gigi, THT, Mata, dan Anak. Peralatan medis yang disediakan sudah memadai untuk memberikan layanan terbaik bagi masyarakat. Tersedia unit layanan rawat inap untuk pasien dewasa dan anak, ditambah dengan ruangan rawat bayi. Sarana penunjang yang terdiri dari fasilitas instalasi farmasi, instalasi laboratorium, dan ambulance akan mendukung lancarnya pelayanan. Jaminan pemenuhan hak-hak pasien dan keluarga dipastikan dengan tersedianya manual atau prosedur pelayanan yang memenuhi standar nasional.

RSIA "Bunda Nanda" juga melakukan kerjasama penggunaan beberapa fasilitas penunjang medis yaitu pelayanan instalasi laboratorium, instalasi farmasi, radiologi dan pengolahan limbah padat medis.

2.2. Tinjauan Pustaka

Tinjauan Pustaka adalah apa yang dilakukan untuk menjelaskan hal-hal apa saja yang digunakan dalam penelitian ini, untuk membantu pembaca memahami hal-hal yang belum mereka ketahui.

2.2.1. Software Re-engineering

Reengineering aplikasi, juga dikenal sebagai *Reengineering* perangkat lunak atau *Reengineering* sistem, adalah proses mendalam dan holistik dalam mengubah atau memperbarui aplikasi atau sistem perangkat lunak yang sudah ada dengan tujuan untuk meningkatkan kualitas, efisiensi, dan kinerjanya. [1]

Pada dasarnya, *reengineering* aplikasi melibatkan penilaian kembali dan perancangan ulang keseluruhan atau bagian-bagian tertentu dari aplikasi dengan pendekatan yang lebih modern dan efisien. Hal ini sering kali dilakukan untuk mengatasi masalah yang ada dalam aplikasi yang telah ada selama beberapa waktu atau untuk mengakomodasi kebutuhan bisnis yang terus berkembang. Tujuan dari *reengineering* aplikasi dapat mencakup:

1. Meningkatkan Kualitas

   Memperbaiki masalah yang ada dalam aplikasi yang sudah ada, termasuk bug, ketidaksempurnaan, dan ketidakstabilan.

2. Meningkatkan Kinerja

   Mengoptimalkan kinerja aplikasi, seperti waktu respon, kecepatan, dan efisiensi.

3. Menggunakan Teknologi Terbaru

   Memperbarui teknologi yang digunakan dalam aplikasi untuk menggunakan teknologi terbaru dan lebih efisien.

4. Meningkatkan Skalabilitas

   Mengoptimalkan aplikasi agar dapat dengan mudah berkembang dan mengakomodasi pertumbuhan bisnis di masa depan.

5. Meningkatkan Pengalaman Pengguna

   Menghadirkan antarmuka pengguna yang lebih intuitif, menarik, dan mudah digunakan.

6. Mengurangi Biaya dan Kompleksitas

   Meminimalkan biaya operasional dan menghilangkan bagian-bagian aplikasi yang tidak diperlukan atau usang.

Langkah-langkah dalam proses *reengineering* aplikasi meliputi:

1. Evaluasi

   Melakukan analisis mendalam terhadap aplikasi yang sudah ada untuk mengidentifikasi masalah dan kebutuhan perbaikan.

2. Perancangan Ulang

   Merancang ulang struktur, alur kerja, dan fitur aplikasi dengan menggunakan pendekatan yang lebih modern dan efisien.

3. Pemetaan Proses

   Memetakan alur kerja dan proses aplikasi yang ada untuk memahami interaksi dan ketergantungannya.

4. Pemilihan Teknologi

   Memilih teknologi yang sesuai untuk mendukung tujuan *s* dan kebutuhan aplikasi.

5. Pengembangan

   Mengimplementasikan perancangan ulang ke dalam kode sumber aplikasi.

6. Pengujian

   Melakukan pengujian menyeluruh untuk memastikan bahwa aplikasi baru berfungsi dengan baik dan sesuai dengan harapan.

7. Pelatihan dan Implementasi

   Melatih pengguna dan mengimplementasikan aplikasi baru secara menyeluruh.

1. Taksonomi Software Re-engineering

Taksonomi *software reengineering* adalah pengelompokkan atau klasifikasi dari berbagai jenis dan pendekatan dalam proses *reengineering* perangkat lunak. Ini membantu dalam memahami dan mengorganisasi berbagai aspek yang terlibat dalam *reengineering software*. Meskipun tidak ada satu taksonomi yang pasti atau standar, berikut adalah beberapa kategori umum yang dapat digunakan untuk menggambarkan taksonomi *software reengineering*: [2]

1. *Reverse Engineering*

   Melibatkan proses untuk memahami perangkat lunak yang sudah ada dengan menganalisis kode sumber atau kode biner untuk mendapatkan pemahaman tentang struktur, fungsionalitas, dan logika program.

2. *Code Refactoring*

   Aktivitas untuk meningkatkan kualitas kode dan mengurangi utang teknis dengan melakukan perubahan kecil pada kode sumber tanpa mengubah perilaku eksternal [3].

3. *Program Restructuring*

   Fokus pada perubahan struktur internal perangkat lunak, termasuk modul, kelas, dan interaksi antara komponen, untuk meningkatkan efisiensi dan kualitas keseluruhan.

4. *Data Migration*

   Melibatkan pemindahan atau transformasi data dari satu bentuk atau format ke bentuk atau format lainnya, seringkali dalam rangka migrasi dari *platform* lama ke *platform* baru [1].

5. *Architecture Redesign*

   Melibatkan perubahan arsitektur perangkat lunak untuk memenuhi kebutuhan yang berubah dan meningkatkan kemampuan sistem.

6. *User Interface (UI) Revamping*

   Berfokus pada perubahan dan perbaikan antarmuka pengguna untuk meningkatkan pengalaman pengguna dan kegunaan aplikasi.

7. *Legacy System Migration*

   Proses untuk memindahkan sistem perangkat lunak yang sudah ada ke *platform* atau teknologi yang lebih baru agar sistem tetap berjalan dan mendukung kebutuhan bisnis.

8. *Software Testing and Validation*

   Proses validasi yang penting dalam semua tahap *reengineering*, untuk memastikan bahwa perubahan yang dilakukan tidak mengenai fungsionalitas yang ada dan sistem berfungsi dengan baik.

2. Migrasi *Platform*

Migrasi *platform* mengacu pada proses atau tindakan mengubah atau memindahkan sistem atau aplikasi dari satu *platform* atau lingkungan teknologi ke *platform* atau lingkungan teknologi yang berbeda. *Platform* dalam konteks ini dapat merujuk pada sistem operasi, perangkat keras, perangkat lunak, atau infrastruktur lain yang digunakan untuk menjalankan aplikasi atau layanan [4].

Migrasi *platform* sering kali dilakukan ketika organisasi atau pengembang ingin menghadapi perubahan teknologi, meningkatkan kinerja, mengoptimalkan biaya operasional, atau meningkatkan keamanan. Alasan lain untuk migrasi *platform* termasuk *end-of-life* dari *platform* yang ada, keinginan untuk mengadopsi teknologi terbaru, atau kebutuhan untuk skalabilitas yang lebih baik. Proses migrasi *platform* melibatkan beberapa tahapan, di antaranya:

1. Perencanaan

Tahap ini melibatkan analisis mendalam tentang kebutuhan, tujuan, dan risiko dari migrasi. Di sini, rencana dan strategi migrasi akan dibuat, termasuk pemilihan *platform* sasaran dan perencanaan langkah-langkah teknis yang diperlukan.

2. Persiapan

   Persiapan melibatkan memastikan bahwa lingkungan sumber dan lingkungan target siap untuk migrasi. Ini dapat mencakup pengujian kompatibilitas aplikasi, backup data, dan persiapan sumber daya yang diperlukan.

3. Pengujian

   Sebelum migrasi sebenarnya, penting untuk melakukan pengujian menyeluruh untuk memastikan bahwa aplikasi atau sistem akan berfungsi dengan baik di *platform* baru. Ini termasuk pengujian fungsionalitas, keamanan, dan kinerja.

4. Pelaksanaan

   Tahap pelaksanaan adalah ketika migrasi *platform* sebenarnya dilakukan. Data, aplikasi, dan konfigurasi dipindahkan dari *platform* lama ke *platform* baru. Proses ini harus diawasi dengan cermat untuk menghindari gangguan layanan.

5. Pengujian Lanjutan

   Setelah migrasi, pengujian lanjutan harus dilakukan untuk memastikan bahwa sistem berfungsi dengan baik dan sesuai dengan harapan.

6. Optimasi

   Setelah migrasi selesai, mungkin diperlukan beberapa penyesuaian dan pengoptimalan untuk memastikan performa dan kestabilan yang baik.

7. Pelatihan dan Pendukung

Pengguna dan tim dukungan harus diberikan pelatihan dan pengetahuan yang cukup tentang *platform* baru untuk dapat menjalankan sistem dengan baik.

Migrasi *platform* dapat menjadi proses yang kompleks dan berisiko, terutama untuk sistem besar dan kritis. Oleh karena itu, perencanaan yang matang, pengujian yang baik, dan dukungan teknis yang tepat sangat penting untuk kesuksesan migrasi.

3. Enhanced Reengineering

Enhanced Reengineering adalah proses rekayasa ulang perangkat lunak yang memanfaatkan banyak metode dan level abstraksi, untuk mengubah sistem perangkat lunak yang sudah ada menjadi sistem perangkat lunak yang baru. Pada metode *enhanced reengineering* ini menggunakan sistem *forward engineering* dan *reverse engineering* [3].

Pada awalnya metode *enhanced reengineering* ini melakukan studi kelayakan untuk menguji kompatibilitas sistem, dan kemudian membangun komponen yang diperlukan. Kemudian setelah mengumpulkan persyaratan, masuk ke fase kedua yaitu pemetaan restrukturisasi spesifikasi persyaratan perangkat lunak (SRS) untuk menyelesaikan dokumen desain. Dokumen yang dirancang ulang hasil dari fase kedua.

Pada tahap ketiga, bagian pemrograman disesuaikan dengan perubahan yang dilakukan pada dokumen yang didesain ulang, pada tahap ini dimungkinkan untuk memutar kembali ke tahap sebelumnya. Kemudian uji ulang dan integrasikan kembali modul perangkat lunak yang ada untuk menjalankan fungsi tertentu. Selama fase ini, membandingkan kinerja sistem perangkat lunak yang ada dengan perangkat lunak baru. Hasil yang diperoleh konsisten, algoritma yang lebih baik

akan ditukar dengan sistem yang ada [5].

Manfaat dari proses ini adalah mengurangi kompleksitas dan meningkatkan kualitas perangkat lunak baru. Setelah menyelesaikan berbagai integrasi unit, sistem yang dimodifikasi akan dikerahkan untuk mendapatkan sistem target sesuai dengan kebutuhan pengguna[7]. Berikut ini adalah beberapa tahap yang dilakukan pada *Enhanced Reengineering* dan penjelasannya:

a. Studi Kelayakan dan Kebutuhan

Pada tahap ini dilakukan studi kelayakan dan perlu dilakukan verifikasi konfigurasi dan kompatibilitas sistem komputer. Setelah studi kelayakan selesai, kebutuhan sistem didefinisikan ulang sesuai dengan keinginan pengguna. SRS memiliki semua persyaratan yang tertulis dalam dokumen resmi.

b. Restrukturisasi Spesifikasi Kebutuhan Sistem.

Tahap ini menggambarkan secara rinci proses SRS yang direstrukturisasi. Dokumentasi adalah atribut penting dalam proses pengembangan perangkat lunak karena memproduksi komponen dari keseluruhan proses rekayasa ulang dan berfungsi sebagai perencana untuk produk akhir. Pada tahap ini, para ahli membandingkan kebutuhan sistem yang ada dengan mekanisme sistem yang baru. SRS digunakan untuk mengintegrasikan SRS baru dengan SRS sudah yang ada.

c. *Design to Code*

Langkah ini memberikan informasi rinci tentang desain proses kode. Pada tahap ini sudah tepat untuk mendesain ulang kode-kode terdokumentasi yang telah dilakukan oleh programmer. Biasanya, algoritma dari sistem lama akan diimplementasikan dengan cara pengembangan tradisional dan bahasa dengan fitur yang ada akan ditulis ulang dalam fitur baru.

d. Evaluasi Performa Sistem baru dengan yang sudah ada

Tahap ini memberikan rincian tentang proses pengujian ulang. Untuk pengujian ulang, terlebih dahulu ambil aplikasi perangkat lunak yang sudah ada maupun yang baru. Kemudian membandingkan kinerja fungsionalitas dari aplikasi yang ada dengan fungsionalitas dari aplikasi yang baru.

e. Implementasi

Tahap ini adalah tahap akhir dari proses mekanisme *Enhanced Re-engineering.* Sesuai dengan tahapan hasil dari rekayasa ulang sebelumnya, implementasi suatu perangkat lunak dapat dilakukan. Dalam implementasinya, semua bagian dimigrasi dari sistem yang sebelumnya ke sistem yang baru, berdasarkan empat tahap sebelumnya.

4. SRS

Menurut Wisnu (2012), menyatakan bahwa Software Requirements Specification (SRS) adalah dokumen yang menjelaskan tentang berbagai kebutuhan yang harus dipenuhi oleh suatu software. Dokumen ini dibuat oleh developer (pengembang software) setelah menggali informasi dari calon pemakai software. Pembuatannya mengikuti standar yang ada dan paling dianggap benar oleh para praktisi rekayasa software di dunia. Oleh karena itu, standar yang akan dibahas di sini adalah standar dari IEEE, singkatan dari Institute of Electrical and Electronics Engineers.

SRS harus bermanfaat bagi pengguna, penyedia, atau perorangan. Adapun manfaat dari SRS antara lain sebagai berikut:

a. Sebagai bentuk perjanjian antara pengguna dan penyedia tentang software apa yang akan dibuat.

b. Mengurangi beban dalam proses pengembangan software.
c. Sebagai bahan perkiraan biaya dan rencana penjadwalan.
d. Sebagai dasar validasi dan verifikasi software di ujung penyelesaian proyek nantinya.
e. Memfasilitasi transfer, semisal software tersebut ingin ditransfer ke pengguna atau mesin-mesin yang lain. Penggunaakan lebih mudah jika ingin mentransfer software ke bagian-bagian lain dalam organisasinya. Bahkan, jika terjadi pergantian personil pengembang, proyek dapat mudah ditransfer ke personil baru dengan memahami SRS ini.
f. Mendasari perbaikan produk software dikemudian hari. Jadi, SRS boleh diperbaiki dengan alasan dan mekanisme tertentu serta atas kesepakatan antara pengunadan pengembang.
g. Dengan menggunakan SRS, pengguna dapat menuangkan semua ide terkait software dengan jelas dan akurat sehingga pengembangdapat memahami apa yang diinginkan pengguna dengan tepat. Standar ini akan membantu dalam mengembangkan outline SRS yang baku untuk pengguna, membantu membuat dokumen SRS dengan format dan isi yang standar (minimal), serta membantu mengembangkanrincian-rincian pendukung lainnya.

SRS harus memiliki jaminan kualitas perangkat lunak dan pengujian aplikasi. Jaminan perangkat lunak adalah aktivitas pelindung yang diaplikasikan pada seluruh proses perangkat lunak. Tujuan dari jaminan kualitas adalah untuk memberikan data yang diperlukan oleh manajemen dan menginformasikan masalah kualitas produk, sehingga dapat memberikan kepastian dan konfidensi bahwa kualitas produk dapat memenuhi sasaran, tidak hanya berkualitas menurut pengembang tapi juga berkualitas dan sesuai dengan

keinginan pengguna (Nastiti, 2012: 35).

Menurut McCall dalam Nastiti (2012: 36), faktor-faktor penentu kualitas perangkat lunak adalah sebagai berikut:

a. Correctness, sejauh mana suatu perangkat lunak memenuhi spesifikasi dan tujuan penggunaan perangkat lunak dari user.
b. Reliability, sejauh mana keakuratan suatu perangkat lunak dalam melaksanakan fungsinya.
c. Efficiency, banyaknya kode program yang dibutuhkan suatu perangkat lunak untuk melakukan fungsinya.
d. Integrity, sejauh mana akses ke perangkat lunak dan data oleh pihak yang tidak berhak dapat dikendalikan.
e. Usability, usaha yang diperlukan untuk mempelajari, mengoperasikan, menyiapkan input, dan mengartikan output dari perangkat lunak.
f. Maintainability, usaha yang diperlukan untuk menetapkan dan memperbaiki kesalahan dalam program.
g. Testability, usaha yang diperlukan dalam pengujian program untuk memastikan bahwa program melaksanakan fungsi yang ditetapkan.
h. Flexibility, usaha yang diperlukan untuk memodifikasi program operasional.
i. Portability, usaha yang diperlukan untuk memindahkan program dari perangkat keras/lingkungan sistem perangkat lunak tertentu ke yang lainnya.
j. Reusability, tingkat kemampuan program/bagian dari program yang dapat dipakai ulang dalam aplikasi lainnya, berkaitan dengan paket dan lingkup dari fungsi yang dilakukan oleh program.
k. Interoperability, usaha yang diperlukan untuk menggabungkan satu sistem dengan yang lainnya.

Pengujian software adalah metode yang dilakukan untuk menjelaskan tentang pengoperasian perangkat lunak yang terdiri dari perangkat pengujian, metode pengujian dan pelaksanaan pengujian. Ada 2 jenis pengujian, yaitu:

a. Pengujian *Alpha*

Pengujian Alpha dilakukan pada sisi pengembang. Software digunakan pada pengaturan yang natural dengan pengembang yang memandang sisi pemakai dan merekam semua kesalahan dan masalah pemakaian. Pengujian Alpha bertujuan untuk mengidentifikasi dan menghilangkan sebanyak mungkin masalah sebelum akhirnya sampai ke pengguna, dilakukan setelah software selesai oleh orang-orang yang tidak terlibat dalam pengembangan dan memang ahli dibidangnya menggunakan formulir evaluasi resmi.

b. Pengujian *Betha*

Pengujian Betha dilakukan pada satu atau lebih pengguna software dalam lingkungan yang sebenarnya, pengembang tidak terlibat pengujian ini. Pengguna merekam semua masalah (nyata atau imajiner) yang ditemui selama pengujian dan melaporkan pada pengembang pada interval waktu tertentu. Berdasarkan hasil pengujian Betha, dicari presentase masing-masing jawaban dengan menggunakan rumus:

$$Y = (P/Q) \times 100\%$$

Keterangan:

Y = Nilai presentase

P = Banyaknya jawaban responden tiap soal

Q = Jumlah responden

5. Framework

Framework dalam sistem berorientasi objek, merupakan kumpulan class yang melambangkan bentuk abstrak untuk pemecahan sejumlah masalah yang berhubungan

Framework adalah sebuah software untuk memudahkan para programmer untuk membuat sebuah aplikasi website maupun aplikasi mobile phone yang didalamnya berbagai fungsi diantaranya plugin, dan konsep untuk membentuk suatu sistem tertentu agar tersusun dan terstruktur dengan rapi.

Dengan menggunakan framework bukan berarti akan terbebas dari pengkodean. Karena sebagai pengguna framework harus menggunakan fungsi-fungsi dan variable yang ada di dalam sebuah framework yang digunakan. Saat ini ada beberapa framework untuk pembuatan website dan framework untuk pembuatan aplikasi (Idcloudhost, 2017)

6. Flutter

Flutter dirilis resmi pada bulan Desember 2018 oleh Google. Flutter merupakan SDK untuk mengembangkan aplikasi baik mobile, web maupun desktop. Flutter adalah framework yang membantu developer untuk membuat aplikasi mobile multiplatform.

Banyak perusahaan yang telah menggunakan framework ini seperti Alibaba, ebay, hingga Google Ads (Flutter, 2018). Terdapat 2 komponen penting dalam Flutter yaitu:

a. Software Development Kit (SDK) adalah kumpulan *tools* yang berfungsi untuk meng*compile code* agar aplikasi dapat dijalankan di berbagai platform.
b. Framework UI adalah komponen UI seperti teks, *button*, *navigation* dan lainnya yang dapat dikustomisasi.

Setiap membuat UI di dalam Flutter akan menciptakan berbagai widget. Beberapa karakteristik dari widget yaitu dapat digunakan untuk membuat UI seperti column, row, text, stack, card, form, padding. Developer juga dapat melakukan styling seperti tipe font, size, color, margin, shadow. Selain itu widget dapat berupa kumpulan bentuk dan formular. Di dalam Flutter terdapat fitur hot-reload. Fitur ini memudahkan developer dalam membangun UI, menambahkan fitur dan memperbaiki bug dengan melihat langsung perubahan UI yang terdapat pada aplikasi. Hot- reload bekerja dengan menanamkan kode file terbaru ke dalam Dart Virtual Machine (VM). Kerangka pada Flutter membuat ulang semua widget secara otomatis setelah VM diperbarui. Fitur ini yang memudahkan developer untuk melihat langsung perubahan UI yang dikembangkan.

Untuk pengembangan aplikasi mobile, Flutter dapat digunakan untuk mengembangkan aplikasi berbasis Android dan iOS dengan menggunakan bahasa Dart. Dart merupakan bahasa pemrograman berbasis class dan berorientasi objek. Di dalam Flutter terdapat dua kombinasi class yang digunakan yaitu: Stateless Widgets dan Stateful Widgets. Class ini merupakan widget yang bertugas menampilkan tampilan user interface dengan membangun widget lainnya agar menjadi lebih terlihat nyata. Stateless Widgets adalah widgets yang bersifat statis dan seluruh konfigurasi yang dimuat sudah diinisialisasi dari awal atau hanya dapat menggambar satu kali saat widget dimuat sehingga tidak dapat

menggunakan fitur hot-reload. Sedangkan Stateful widgets adalah widgets yang bersifat dinamis dan dapat berubah pada kondisi tertentu sehingga dapat menggunakan fitur hot-reload (Boukhary & Colmenares, 2019).

Dengan Flutter kita dapat mengembangkan aplikasi ke semua jenis platform. Hal ini dikarenakan Dart memiliki AOT-Compile (Ahead Of Time). Ketika developer mengembangkan aplikasi dengan bahasa Dart, AOT akan mengcompile code yang dibuat dari bahasa Dart ke native menuju platform yang diinginkan. Jika pada Android code tersebut akan diterjemahkan ke dalam bahasa Kotlin. Pada Android code tersebut akan dicompile dengan mengunakan mesin C++ dan menggunakan Native Development Kit (NDK). Kemudian code native pada Android akan tercompile lagi dengan Dart Compiler. Sedangkan di iOS akan dicompile dengan mesin LLVM (Low-Level Virtual Machine). Setelah itu code akan disesuaikan dengan masing-masing perangkat agar dapat dijalankan pada berbagai platform.

Dalam konteks pengembangan aplikasi web menggunakan Flutter, arsitektur yang diadopsi memiliki beberapa komponen inti yang bekerja sama untuk menghasilkan aplikasi yang responsif, efisien, dan mudah di-maintain. Flutter, sebagai framework yang dikembangkan oleh Google, menawarkan pendekatan yang unik dalam pengembangan aplikasi lintas platform, termasuk web. Secara umum, arsitektur aplikasi web Flutter dapat dibagi menjadi beberapa bagian penting yang saling berinteraksi.

Pertama, terdapat lapisan presentasi yang bertanggung jawab atas antarmuka pengguna (UI). Dalam Flutter, lapisan ini dibangun menggunakan widget, yang merupakan elemen dasar dari UI. Setiap elemen dalam aplikasi, seperti tombol, teks, atau gambar, merupakan widget yang dapat dikombinasikan dan disusun

menjadi struktur UI yang lebih kompleks. Pengembang dapat memanfaatkan widget bawaan atau membuat widget kustom sesuai kebutuhan desain aplikasi.

Selanjutnya, terdapat lapisan logika bisnis yang mengelola interaksi antara UI dan data. Flutter sering mengadopsi pola arsitektur seperti BLoC (Business Logic Component) atau Provider untuk memisahkan logika bisnis dari UI. Pola ini membantu menjaga kebersihan kode dan memudahkan dalam pengujian serta pengelolaan aplikasi. Di dalam lapisan ini, logika bisnis seperti pengolahan data, validasi, dan pengambilan keputusan diletakkan, sehingga UI hanya bertanggung jawab menampilkan data yang sudah diolah.

Pada lapisan data, Flutter dapat berinteraksi dengan berbagai sumber data seperti database, API, atau sumber data lainnya melalui penggunaan paket dan plugin. Untuk aplikasi web, ini biasanya melibatkan komunikasi dengan backend melalui HTTP request menggunakan paket seperti http atau dio. Data yang diperoleh kemudian diproses dan disalurkan ke lapisan logika bisnis sebelum akhirnya ditampilkan pada UI.

Di sisi lain, dalam arsitektur Flutter web, penting untuk mempertimbangkan aspek optimisasi performa dan skalabilitas. Flutter menggunakan mesin rendering yang disebut Skia untuk merender UI secara langsung ke kanvas HTML. Ini memungkinkan aplikasi Flutter memiliki performa yang mendekati native, meskipun dijalankan di browser. Namun, karena sifatnya yang heavy-weight, pengembang perlu berhati-hati dalam memanajemen state dan meminimalkan re-rendering UI yang tidak diperlukan untuk menjaga performa tetap optimal.

Integrasi dengan alat pengembangan dan pengujian juga merupakan bagian penting dari arsitektur Flutter. Flutter menyediakan alat seperti Flutter DevTools untuk debugging dan monitoring performa aplikasi. Penggunaan alat ini

memungkinkan pengembang untuk mengidentifikasi bottleneck performa dan memecahkan masalah dengan lebih efektif.

Dengan demikian, arsitektur aplikasi web Flutter mencakup berbagai aspek mulai dari desain UI berbasis widget, pengelolaan logika bisnis, interaksi dengan sumber data, hingga optimisasi performa. Pendekatan ini menawarkan fleksibilitas dan efisiensi dalam pengembangan aplikasi web yang modern dan responsif.

7. Codeigniter 4

CodeIgniter 4 adalah *framework* PHP sumber terbuka (*open-source*) yang digunakan untuk pengembangan aplikasi *website*. *Framework* ini merupakan penerus dari CodeIgniter 3 dan dikembangkan oleh *British Columbia Institute of Technology* (BCIT). CodeIgniter 4 dirilis untuk memberikan fitur-fitur terbaru, perbaikan keamanan, dan meningkatkan kinerja dibandingkan dengan versi sebelumnya. Berikut adalah beberapa poin penting tentang CodeIgniter 4:

1. MVC *Architecture*:

   CodeIgniter 4 mengikuti pola arsitektur MVC (*Model-View-Controller*) yang memisahkan logika aplikasi, tampilan, dan interaksi dengan *database* menjadi tiga komponen terpisah.

2. Ringan dan Cepat:

   CodeIgniter 4 dirancang untuk menjadi *framework* yang ringan dan cepat. Ini membantu mengurangi waktu *loading* dan meningkatkan performa aplikasi *website*.

3. Composer-based:

CodeIgniter 4 menggunakan *Composer*, manajer dependensi PHP, untuk mengelola paket-paket eksternal dan memudahkan integrasi dengan *library* dan komponen pihak ketiga.

4. Fitur-fitur Baru

   CodeIgniter 4 menyediakan sejumlah fitur baru, termasuk manajemen layanan yang lebih baik, konfigurasi dengan format YAML dan Env, dukungan untuk HTTP/2, dan banyak lagi.

5. Dokumentasi yang Kuat

   CodeIgniter 4 menyediakan dokumentasi yang lengkap dan informatif untuk membantu pengembang memahami dan menggunakan *framework* ini dengan lebih baik.

6. Keamanan

   Framework ini dilengkapi dengan lapisan keamanan yang kuat, termasuk fitur seperti proteksi terhadap CSRF (*Cross-Site Request Forgery*), XSS (*Cross-Site Scripting*), dan sebagainya.

7. Komunitas Aktif:

   CodeIgniter memiliki komunitas yang aktif dan berdedikasi, yang berarti Anda dapat dengan mudah menemukan dukungan, sumber daya, dan ekstensi tambahan dari komunitas tersebut.

Dengan segala fitur dan perbaikan yang ada, CodeIgniter 4 tetap menjadi pilihan populer bagi pengembang PHP untuk membangun aplikasi *website* dengan cepat, efisien, dan aman.

8. Website

Website atau yang sering disebut dengan web adalah kumpulan dari halaman yang berisi beberapa laman yang menampilkan informasi-informasi dalam bentuk digital baik itu teks, gambar, animasi yang disediakan melalui internet dan seluruh dunia dapat mengaksesnya, menurut Rudianto (2011). *Website* dibagi menjadi 2, yaitu :

1. *Website* Statis

   Situs web statis adalah situs web dengan halaman yang tidak berubah, biasanya untuk melakukan perubahan dilakukan secara manual dengan memodifikasi kode. Pada *website* statis informasinya hanyalah satu arah, yaitu hanya dari berdasarkan pemilik *software*-nya saja. Profil perusahaan adalah salah satu contoh dari *website* statis..

2. *Website* Dinamis

   Website dinamis adalah situs web yang halamannya terus diperbarui dan biasanya memiliki halaman internal (halaman admin) yang digunakan untuk menambah atau mengubah konten. Web dinamis membutuhkan database untuk penyimpanan. Situs web dinamis memiliki aliran informasi dua arah yaitu pengguna dan pemilik. Pengguna serta pemilik situs web dapat memperbaikinya.

9. PHP

PHP adalah bahasa pemrograman tingkat tinggi yang dikhususkan untuk pengembangan aplikasi web dan pemrosesan data di sisi server. Singkatan PHP mengandung arti "PHP: *Hypertext Preprocessor*", menunjukkan bahwa bahasa ini awalnya dirancang untuk memproses konten berbasis *hypertext* (HTML) di situs web. Namun, saat ini PHP telah berkembang menjadi bahasa pemrograman serba

guna dan dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak. Fitur Utama PHP antara lain:

1. Sintaks yang Mudah Dipahami

   PHP menggunakan sintaks yang mudah dipahami dan mirip dengan sintaks bahasa pemrograman C, membuatnya mudah diakses oleh pengembang dengan berbagai tingkat pengalaman.

2. Pemrograman Server-side

   PHP dijalankan di sisi server, yang berarti kode PHP diproses di server sebelum hasilnya dikirim ke *browser* pengguna. Ini memungkinkan pembuatan aplikasi *website* dinamis dan interaktif.

3. Integrasi HTML

   PHP dirancang untuk diintegrasikan dengan mudah ke dalam kode HTML. Ini memungkinkan pengembang untuk menyisipkan kode PHP ke dalam halaman *website* untuk membuat konten dinamis dan interaktif.

4. Dukungan Database

   PHP menyediakan dukungan yang luas untuk berbagai jenis basis data, seperti MySQL, PostgreSQL, MongoDB, dan banyak lagi. Ini memudahkan pengembangan aplikasi yang berhubungan dengan penyimpanan data.

5. Framework Populer

   Ada banyak kerangka kerja (*framework*) PHP populer, seperti Laravel, CodeIgniter, dan Symfony, yang menyediakan alat dan struktur untuk mempercepat pengembangan aplikasi *website*.

6. Penggunaan Peralatan Lain

PHP dapat diintegrasikan dengan perangkat lunak lain seperti server *website* (misalnya, Apache), sistem manajemen basis data (DBMS), dan perangkat lunak server lainnya.

7. Sumber Terbuka

PHP bersifat *open-source*, yang berarti sumber kode bahasa ini dapat diakses dan dimodifikasi oleh siapa saja. Hal ini mendorong komunitas pengembang yang besar dan aktif.

10. Standar ISO/IEC 9126

ISO/IEC 9126 adalah standar internasional yang dikembangkan oleh International Organization for Standardization (ISO) dan International Electrotechnical Commission (IEC) untuk mengevaluasi kualitas perangkat lunak. Standar ini memberikan kerangka kerja untuk menilai kualitas perangkat lunak berdasarkan enam karakteristik utama, yaitu fungsionalitas, keandalan, kegunaan, efisiensi, pemeliharaan, dan portabilitas.

Fungsionalitas mengacu pada sejauh mana perangkat lunak memenuhi kebutuhan yang ditentukan ketika digunakan dalam kondisi tertentu. Ini mencakup beberapa aspek penting, yaitu kecocokan, akurasi, interoperabilitas, kepatuhan, dan keamanan. Kecocokan merujuk pada kemampuan perangkat lunak untuk menyediakan fungsi-fungsi yang sesuai dengan kebutuhan pengguna, sementara akurasi menggambarkan kemampuan perangkat lunak untuk memberikan hasil yang benar atau sesuai dengan persyaratan. Interoperabilitas menekankan kemampuan perangkat lunak untuk berinteraksi dengan sistem lain, kepatuhan berkaitan dengan kemampuan perangkat lunak untuk mematuhi standar, regulasi,

atau konvensi tertentu, dan keamanan berfokus pada kemampuan perangkat lunak untuk melindungi data dan informasi dari akses tidak sah.

Keandalan mengacu pada kemampuan perangkat lunak untuk mempertahankan kinerjanya di bawah kondisi tertentu untuk periode waktu tertentu. Sub-karakteristik keandalan meliputi kemantapan, toleransi kesalahan, dan pemulihan. Kemantapan menggambarkan kemampuan perangkat lunak untuk menghindari kegagalan dalam operasi, sedangkan toleransi kesalahan merujuk pada kemampuan perangkat lunak untuk mempertahankan tingkat kinerja yang ditentukan meskipun terjadi kesalahan atau kegagalan. Pemulihan berkaitan dengan kemampuan perangkat lunak untuk memulihkan data dan kembali ke keadaan normal setelah terjadi kegagalan.

Kegunaan mengacu pada kemudahan penggunaan perangkat lunak oleh pengguna akhir. Kegunaan mencakup aspek-aspek seperti kemudahan dipahami, kemudahan belajar, dan kemudahan operasi. Kemudahan dipahami merujuk pada kemampuan perangkat lunak untuk memungkinkan pengguna memahami fungsi-fungsi yang ada, sementara kemudahan belajar menekankan pada kemampuan perangkat lunak untuk memfasilitasi proses pembelajaran oleh pengguna. Kemudahan operasi merujuk pada kemampuan perangkat lunak untuk memfasilitasi pengguna dalam mengoperasikan sistem.

Efisiensi mengacu pada kemampuan perangkat lunak untuk memberikan kinerja yang memadai relatif terhadap sumber daya yang digunakan. Karakteristik ini mencakup waktu respons dan pemanfaatan sumber daya. Waktu respons menggambarkan kemampuan perangkat lunak untuk memberikan waktu respons yang sesuai dengan kebutuhan pengguna, sedangkan pemanfaatan sumber daya mengacu pada efisiensi perangkat lunak dalam menggunakan sumber daya, seperti

memori dan prosesor.

Pemeliharaan mengacu pada kemudahan dengan mana perangkat lunak dapat dimodifikasi untuk memperbaiki kesalahan, meningkatkan kinerja, atau menyesuaikan dengan lingkungan yang berubah. Sub-karakteristik pemeliharaan meliputi kemudahan analisis, kemudahan modifikasi, stabilitas, dan kemudahan pengujian. Kemudahan analisis menggambarkan kemampuan perangkat lunak untuk mendukung analisis terhadap masalah atau perubahan, sedangkan kemudahan modifikasi merujuk pada kemampuan perangkat lunak untuk dimodifikasi tanpa menimbulkan kesalahan baru atau kerusakan pada komponen lain. Stabilitas berfokus pada kemampuan perangkat lunak untuk mencegah efek negatif dari perubahan, dan kemudahan pengujian menekankan pada kemampuan perangkat lunak untuk diuji guna memastikan bahwa perubahan tidak menimbulkan kesalahan baru.

Portabilitas mengacu pada kemudahan perangkat lunak untuk dipindahkan dari satu lingkungan atau platform ke lingkungan lain. Ini mencakup atribut seperti kemudahan adaptasi, kemudahan instalasi, kesesuaian dengan standar, dan kemudahan penggantian. Kemudahan adaptasi merujuk pada kemampuan perangkat lunak untuk diadaptasi ke lingkungan yang berbeda tanpa memerlukan usaha yang signifikan. Kemudahan instalasi menggambarkan kemampuan perangkat lunak untuk diinstal dalam lingkungan yang berbeda, sementara kesesuaian dengan standar berkaitan dengan kemampuan perangkat lunak untuk mematuhi standar dan regulasi yang relevan di berbagai lingkungan. Kemudahan penggantian menekankan pada kemampuan perangkat lunak untuk digantikan oleh perangkat lunak lain dengan fungsi yang sama dalam lingkungan yang berbeda.

Standar ISO/IEC 9126 memberikan panduan yang komprehensif untuk mengevaluasi kualitas perangkat lunak dengan mempertimbangkan berbagai aspek yang relevan, sehingga membantu pengembang dalam merancang, mengembangkan, dan memelihara perangkat lunak yang memenuhi kebutuhan pengguna dengan lebih baik.

11. Maintainability

Maintainability (*Maintain ability*) adalah salah satu karakteristik kualitas perangkat lunak atau sistem yang menggambarkan seberapa mudah dan efisien suatu aplikasi dapat dipelihara, diperbaiki, ditingkatkan, dan diubah oleh pengembang setelah diterbitkan. Sebagai bagian dari atribut kualitas perangkat lunak, *maintainability* menjadi penting karena perangkat lunak sering kali harus melewati siklus hidup yang panjang, dan kebutuhan bisnis serta lingkungan teknologi bisa berubah seiring waktu.

Ada beberapa upaya yang dilakukan untuk mengukur tingkat pemeliharaan suatu perangkat lunak yang biasa disebut dengan metrik. Metrik yang paling banyak digunakan untuk mengukur tingkat pemelihara pada suatu perangkat lunak dikenal dengan *Maintainability Index* (MI)

Dengan kata lain, *maintainability* menyoroti seberapa mudah dan cepat pengembang dapat melakukan perubahan atau pemeliharaan pada perangkat lunak tanpa mempengaruhi atau merusak bagian lain dari sistem. Tingkat *maintainability* yang tinggi akan membantu mengurangi biaya dan risiko yang terkait dengan pemeliharaan perangkat lunak. Karakteristik dari *maintainability* meliputi:

1. Keterbacaan Kode

Kode yang mudah dibaca dan dipahami oleh pengembang akan mempermudah mereka untuk mengidentifikasi masalah atau melakukan perubahan.

2. Modularitas

   Perangkat lunak yang dibagi menjadi modul atau komponen terpisah memudahkan perbaikan atau penggantian bagian-bagian tertentu tanpa mempengaruhi bagian lain dari sistem.

3. Dokumentasi yang Baik

   Dokumentasi yang jelas dan komprehensif membantu pengembang memahami struktur dan fungsi perangkat lunak.

4. Desain yang Baik

   Desain perangkat lunak yang baik, termasuk arsitektur yang baik, akan memudahkan perubahan dan pemeliharaan.

5. Konsistensi Kode

   Penggunaan pola dan konvensi koding yang konsisten membuat kode lebih mudah diubah dan dipahami oleh pengembang lain.

6. Tes yang Efisien

   Adanya pengujian yang baik dan efisien memastikan bahwa perubahan atau pemeliharaan tidak mempengaruhi fungsionalitas yang ada.

7. Pemisahan Logika Bisnis dari Presentasi

   Memisahkan logika bisnis dari kode tampilan (presentasi) akan memudahkan perubahan pada logika tanpa mempengaruhi tampilan.

Maintainability merupakan salah satu dari beberapa atribut kualitas perangkat lunak, termasuk keandalan (*reliability*), efisiensi (*efficiency*), keamanan

(*security*), dan fleksibilitas (*flexibility*). Tingkat *maintainability* yang tinggi sangat penting untuk memastikan perangkat lunak tetap dapat digunakan dan berkembang seiring kebutuhan bisnis dan teknologi berubah.

ISO 9126 adalah standar internasional untuk mengevaluasi kualitas perangkat lunak. Salah satu aspek penting dari ISO 9126 adalah maintainability, yang terdiri dari beberapa subkarakteristik:

1. **Analysability**: Kemudahan untuk menganalisis kode ketika ada masalah atau ketika perubahan diperlukan.
2. **Changeability**: Kemudahan untuk memodifikasi kode, baik untuk memperbaiki kesalahan atau menambahkan fitur baru.
3. **Stability**: Kemampuan kode untuk meminimalkan dampak perubahan, menghindari pengenalan kesalahan baru.
4. **Testability**: Kemudahan untuk menguji kode setelah perubahan dilakukan.

12. Maintainability Index (MI)

Maintainability Index (MI) terdiri dari ekspresi *polynomial* dan menghasilkan angka yang menunjukkan tingkat kemampuan pemeliharaan perangkat lunak secara keseluruhan. Dalam buku *Khan dkk* mendefinisikan MI sebagai kombinasi dari metrik perangkat lunak yaitu *Cyclomatic Complexity* (CC),*Halstead`s Volume* (V) dan *Lines of Code* (LOC) yang mempengaruhi dalam pemeliharaan suatu perangkat lunak[12].

Maintainability Index (MI) yang dihitung menggunakan metrik seperti Lines of Code (LOC), Cyclomatic Complexity (CC), dan Halstead Metrics dapat mencerminkan subkarakteristik ISO 9126:

1. **Analysability**: Diwakili oleh Cyclomatic Complexity (CC). Kode dengan CC tinggi lebih sulit dianalisis.
2. **Changeability**: Dipengaruhi oleh LOC dan Halstead Metrics. Kode yang panjang (LOC tinggi) atau kompleks (Halstead tinggi) lebih sulit diubah.
3. **Stability**: Tidak secara langsung diukur oleh MI, tetapi kode dengan MI rendah biasanya memiliki stabilitas yang lebih rendah karena perubahan lebih mungkin menimbulkan masalah.
4. **Testability**: Diwakili oleh MI. Kode dengan MI tinggi cenderung lebih mudah diuji.

Untuk formula dari *Maintainability Index* dapat dilihat seperti berikut:

$$MI = \frac{171 - 5.2 * \log(HV) - 0.23 * CC - 16.2 * \ln(LOC)}{171} * 100$$

Keterangan :

MI = *Maintainability Index*

HV = *Halstead Volume*

CC = *Cyclomatic Complexity*

LOC = *Line of Code* per *module*

Tabel 2 1 Rentang Nilai Maintainability Index

Rentang	Keterangan
$80 \leq MI \leq 100$	Dapat dipelihara dengan baik
$60 \leq MI \leq 80$	Cukup untuk dipelihara dengan baik
$0 \leq MI \leq 60$	Sulit untuk dipelihara

A. Halstead Volume

Halstead Volume adalah salah satu metrik yang dikenal dalam ilmu perangkat lunak. Adapun formula pada metrik *Halstead Volume* sebagai berikut:

n1 = jumlah operator yang berbeda

n2 = jumlah operan yang berbeda
N1 = jumlah total operator yang berbeda
N2 = jumlah total operan yang berbeda
Kalkulasi *Halstead Vocabulary* (n) dan *length* (N)
N = n1 + n2
N = N1 + N2
Program volume (V)
V = N * 2log(n)

B. Cyclomatic Complexity

V (g) = e – n + p

Keterangan :

e = jumlah rusuk dalam suatu graf

n = jumlah node dalam suatu graf

p = jumlah komponen penghubung

13. Portability

Portability aplikasi merujuk pada kemampuan sebuah aplikasi untuk berjalan di berbagai platform atau lingkungan tanpa memerlukan perubahan yang signifikan. Dalam pengembangan perangkat lunak, portability menjadi faktor penting yang memungkinkan aplikasi untuk beradaptasi dengan berbagai sistem operasi dan perangkat keras. Berdasarkan standar internasional ISO/IEC 9126 yang mendefinisikan kualitas perangkat lunak, portability adalah salah satu karakteristik utama dari kualitas perangkat lunak. Aspek-aspek yang tercakup dalam portability meliputi adaptabilitas, instalabilitas, dan kemampuan pemeliharaan aplikasi dalam lingkungan yang berbeda.

Adaptabilitas mengacu pada kemampuan aplikasi untuk dikonfigurasi atau dimodifikasi agar dapat beroperasi dalam lingkungan yang berbeda. Instalabilitas berkaitan dengan kemudahan proses instalasi aplikasi di sistem baru, sedangkan kemampuan pemeliharaan merujuk pada kemudahan dalam mempertahankan dan memperbarui aplikasi di berbagai platform. Semua elemen ini berkontribusi pada seberapa baik sebuah aplikasi dapat dipindahkan atau diterapkan di lingkungan yang berbeda dari lingkungan awalnya.

Portability aplikasi dipengaruhi oleh beberapa faktor penting. Salah satunya adalah ketergantungan pada platform tertentu, yang dapat memengaruhi kemampuan aplikasi untuk beroperasi di platform lain. Aplikasi yang sangat tergantung pada API atau fitur spesifik dari satu platform akan menghadapi tantangan yang lebih besar saat dipindahkan ke platform yang berbeda. Dalam hal ini, penggunaan bahasa pemrograman yang memiliki dukungan lintas platform seperti Java atau Python dapat meningkatkan tingkat portability, karena bahasa-bahasa ini dirancang dengan mempertimbangkan kemampuan untuk berjalan di

berbagai platform.

Modularitas adalah faktor lain yang berperan dalam meningkatkan portability. Desain aplikasi yang modular memungkinkan bagian-bagian aplikasi untuk diperbarui atau diganti tanpa memengaruhi keseluruhan sistem, sehingga mempermudah proses adaptasi ke platform baru. Selain itu, penggunaan lapisan abstraksi untuk memisahkan logika aplikasi dari sistem operasi atau perangkat keras juga dapat meningkatkan portability, karena hal ini mengurangi ketergantungan langsung pada komponen tertentu dari platform.

Beberapa teknik yang umum digunakan untuk mengukur portability index melibatkan penggunaan alat analisis kode statis, seperti SonarQube, yang menilai kualitas dan modularitas kode sumber aplikasi. Alat ini membantu mengidentifikasi masalah yang dapat mempengaruhi portability. Selain itu, alat untuk pengujian kompatibilitas seperti BrowserStack memungkinkan pengujian aplikasi di berbagai platform untuk menilai masalah yang mungkin timbul selama proses migrasi.

Metode pengukuran lainnya termasuk penggunaan framework seperti Portable Application Metrics (PAM) atau Framework for Software Portability Metrics (FSPM), yang menyediakan metrik kuantitatif untuk menghitung nilai portability index.

Rumus umum yang digunakan untuk menghitung portability index melibatkan penilaian beberapa faktor yang mempengaruhi kemampuan aplikasi untuk dipindahkan. Salah satu rumus yang sering digunakan adalah:

Portability Index = $\frac{P1+P2+\cdots+Pn}{n}$

di mana P1, P2,…, Pn adalah nilai portability dari setiap parameter yang

dievaluasi, dan n adalah jumlah parameter yang digunakan dalam penilaian. Parameter-parameter ini dapat mencakup ketergantungan platform, kepatuhan terhadap standar industri, dan hasil dari uji coba kompatibilitas.

ISO 9126 adalah standar internasional yang digunakan untuk mengevaluasi kualitas perangkat lunak. Salah satu karakteristik penting dalam standar ini adalah *portability* atau portabilitas. *Portability* mengukur sejauh mana perangkat lunak dapat beradaptasi dengan berbagai lingkungan tanpa memerlukan perubahan yang signifikan.

Penentuan skor untuk setiap sub-karakteristik dalam pengujian *portability* berdasarkan ISO 9126 dilakukan melalui evaluasi manual berdasarkan kriteria yang telah ditetapkan dalam standar dan pengalaman teknis. Skor ini dihasilkan dengan mengevaluasi sejauh mana aplikasi memenuhi atau gagal memenuhi kriteria tertentu pada setiap sub-karakteristik (*Adaptability*, *Installability*, *Co-existence*, *Replaceability*) di berbagai platform yang diuji.

Berikut adalah penjelasan mengenai bagaimana skor-skor ini dapat ditentukan:

1. Adaptability

Tabel 2. 1 Tabel *Adaptability*

No.	Skor	Keterangan
1	0	Aplikasi tidak dapat dijalankan di platform lain selain Windows tanpa modifikasi kode.
2	1-3	Aplikasi memerlukan perubahan signifikan dalam kode untuk berfungsi di platform lain, seperti penggantian pustaka atau pemrograman ulang bagian-bagian utama.
3	4-6	Hanya perlu perubahan kecil pada kode atau penyesuaian konfigurasi untuk menjalankan aplikasi di platform lain.
4	7-9	Aplikasi dapat berjalan dengan sedikit atau tanpa perubahan kode, hanya membutuhkan penyesuaian lingkungan.
5	10	Aplikasi sepenuhnya portabel dan tidak memerlukan perubahan kode atau konfigurasi untuk dijalankan di platform manapun.

2. Installability

Tabel 2. 2 Tabel Installability

No.	Skor	Keterangan
1	0	Aplikasi tidak dapat diinstal pada platform lain selain Windows.
2	1-3	Proses instalasi pada platform lain memerlukan modifikasi manual atau pengaturan tambahan yang signifikan.
3	4-6	Instalasi dapat dilakukan dengan beberapa langkah tambahan atau skrip khusus.
4	7-9	Proses instalasi relatif mudah dengan sedikit atau tanpa langkah tambahan.
5	10	Instalasi dapat dilakukan secara langsung di semua platform tanpa penyesuaian atau skrip tambahan.

3. Co-existence

Tabel 2. 3 Tabel Co-existence

No.	Skor	Keterangan
1	0	Aplikasi tidak dapat bekerja bersama perangkat lunak lain pada platform lain selain Windows.
2	1-3	Aplikasi memerlukan perubahan signifikan atau konfigurasi khusus untuk bekerja bersama aplikasi lain.
3	4-6	Beberapa penyesuaian diperlukan agar aplikasi berfungsi bersama perangkat lunak lain.
4	7-9	Aplikasi berfungsi dengan baik bersama perangkat lunak lain dengan sedikit atau tanpa penyesuaian.
5	10	Aplikasi sepenuhnya kompatibel dan dapat berfungsi bersama perangkat lunak lain tanpa penyesuaian.

4. Replaceability

Tabel 2. 4 Tabel Replaceability

No.	Skor	Keterangan
1	0	Komponen aplikasi tidak dapat diganti atau di-upgrade di platform selain Windows.
2	1-3	Penggantian komponen memerlukan modifikasi kode yang signifikan.
3	4-6	Penggantian komponen dapat dilakukan dengan beberapa penyesuaian dalam kode.
4	7-9	Komponen dapat diganti dengan sedikit atau tanpa perubahan kode.
5	10	Komponen sepenuhnya modular dan dapat diganti tanpa mengganggu keseluruhan aplikasi.

Sumber dan Pendekatan Penentuan Skor:

Dokumentasi Teknis: Berdasarkan pemahaman teknis mengenai bagaimana aplikasi bekerja di berbagai platform, misalnya apakah aplikasi menggunakan teknologi yang mendukung multiplatform seperti Java atau menggunakan komponen yang spesifik untuk satu platform saja. Pengujian Praktis: Evaluasi manual dapat dilakukan dengan menguji langsung apakah aplikasi dapat diinstal, dijalankan, atau dimodifikasi di berbagai platform yang berbeda. Skor diberikan berdasarkan hasil observasi ini. Dengan demikian, skor-skor ini ditentukan melalui kombinasi antara evaluasi berbasis standar, pengalaman teknis, dan

pengujian langsung terhadap aplikasi pada berbagai platform.

Tabel 2. 5 Rentang Nilai 1

No.	Skor	Keterangan
1	0-20	Portabilitas sangat rendah. Perangkat lunak hanya dapat berjalan di satu lingkungan tertentu dan memerlukan perubahan besar atau konfigurasi khusus untuk dapat dijalankan di lingkungan lain.
2	21-40	Portabilitas rendah. Perangkat lunak dapat berjalan di beberapa lingkungan, tetapi memerlukan modifikasi yang signifikan untuk dapat beradaptasi dengan lingkungan baru.
3	41-60	Portabilitas sedang. Perangkat lunak dapat dijalankan di beberapa lingkungan dengan sedikit penyesuaian atau konfigurasi.
4	61-80	Portabilitas tinggi. Perangkat lunak dapat dijalankan di berbagai lingkungan dengan sedikit atau tanpa modifikasi.
5	81-100	Portabilitas sangat tinggi. Perangkat lunak dapat dijalankan di berbagai lingkungan tanpa memerlukan perubahan atau konfigurasi apa pun.

Tabel 2. 6 Rentang Nilai 2

No.	Skor	Keterangan
1	0	Portabilitas sangat rendah. Perangkat lunak hanya dapat berjalan di satu lingkungan tertentu dan memerlukan perubahan besar atau konfigurasi khusus untuk dapat dijalankan di lingkungan lain.
2	1-3	Portabilitas rendah. Perangkat lunak dapat berjalan di beberapa lingkungan, tetapi memerlukan modifikasi yang signifikan untuk dapat beradaptasi dengan lingkungan baru.
3	4-6	Portabilitas sedang. Perangkat lunak dapat dijalankan di beberapa lingkungan dengan sedikit penyesuaian atau konfigurasi.
4	7-9	Portabilitas tinggi. Perangkat lunak dapat dijalankan di berbagai lingkungan dengan sedikit atau tanpa modifikasi.
5	10	Portabilitas sangat tinggi. Perangkat lunak dapat dijalankan di berbagai lingkungan tanpa memerlukan perubahan atau konfigurasi apa pun.

Dalam konteks pengujian perangkat lunak, evaluasi *portability* dapat dilakukan dengan beberapa langkah, sebagai berikut:

1. Identifikasi Aspek *Portability*

Portability dalam ISO 9126 terdiri dari beberapa sub-karakteristik yang perlu dievaluasi. Sub-karakteristik ini mencakup:

Adaptability : Kemampuan perangkat lunak untuk beradaptasi pada lingkungan yang berbeda tanpa modifikasi besar.

Installability : Kemampuan perangkat lunak untuk diinstal di berbagai lingkungan.

Co-Existence : Kemampuan perangkat lunak untuk berfungsi secara harmonis dengan perangkat lunak lain dalam lingkungan yang sama.

Replaceability : Kemampuan perangkat lunak untuk digantikan oleh perangkat lunak lain dalam lingkungan yang sama.

2. Persiapan Pengujian

Sebelum melakukan pengujian *portability*, langkah pertama adalah menentukan berbagai lingkungan uji yang relevan, seperti sistem operasi atau perangkat keras yang berbeda. Selain itu, kriteria sukses untuk setiap sub-karakteristik juga perlu ditetapkan. Misalnya, untuk aspek *Adaptability*, perangkat lunak harus dapat berjalan tanpa kesalahan di semua lingkungan uji.

3. Pelaksanaan Pengujian

Misalkan sebuah aplikasi desktop diuji dalam tiga lingkungan berbeda, yaitu:

Windows 10 (64-bit)

Ubuntu 20.04 (64-bit)

macOS 11.0 (Big Sur)

4. Pengukuran dan Perhitungan Skor

Setiap sub-karakteristik dievaluasi dengan memberikan skor dari 0 hingga 10, di mana 10 menunjukkan tingkat *portability* yang sangat tinggi, sementara 0 menunjukkan tidak adanya *portability*.

Berikut adalah contoh skor untuk masing-masing sub-karakteristik:

A. Adaptability :

Windows 10 : 9

Ubuntu 20.04 : 8

macOS 11.0 : 7

Rata-rata : (9 + 8 + 7) / 3 = 8.00

B. Installability :

Windows 10 : 9

Ubuntu 20.04 : 7

macOS 11.0 : 6

Rata-rata : (9 + 7 + 6) / 3 = 7.33

C. Co-Existence :

Windows 10 : 8

Ubuntu 20.04 : 8

macOS 11.0 : 7

Rata-rata : (8 + 8 + 7) / 3 = 7.67

D. Replaceability :

Windows 10 : 7

Ubuntu 20.04 : 6

macOS 11.0 : 6

Rata-rata : (7 + 6 + 6) / 3 = 6.33

5. Menghitung Skor *Portability* Keseluruhan

Untuk mendapatkan skor *portability* keseluruhan, rata-rata dari semua sub-karakteristik dihitung sebagai berikut:

$$Portability = \frac{8.00 + 7.33 + 7.67 + 6.33}{4} = 7.33$$

Skor keseluruhan ini memberikan gambaran umum mengenai *portability* aplikasi tersebut.

6. Interpretasi Hasil

Skor *portability* keseluruhan sebesar 7.33 dari 10 menunjukkan bahwa aplikasi memiliki tingkat *portability* yang cukup baik. Namun, masih terdapat ruang untuk perbaikan, khususnya dalam aspek *Replaceability* dan *Installability*.

7. Rekomendasi Pengembangan

Berdasarkan hasil pengujian, tim pengembang dianjurkan untuk fokus pada peningkatan *Installability* dan *Replaceability* guna meningkatkan *portability* aplikasi di berbagai lingkungan uji.