**PROPOSAL TUGAS AKHIR**

**ANALISIS *MAINTAINABILITY* PENGEMBANGAN APLIKASI *PLATFORM* PEMBELAJARAN PEMROGRAMAN YANG MENERAPKAN SOLID *DESIGN* *PRINCIPLE* MENGGUNAKAN CHIDAMBER-KEMERER METRICS**

Logo

Description automatically generated with medium confidence

**Disusun Oleh:**

**Mujahid Ansori Majid 1197050093**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS ISLAM NEGERI SUNAN GUNUNG DJATI**

**BANDUNG**

**2022**

# DAFTAR BAGAN

[Bagan 1. Variabel yang digunakan 8](#_Toc123909161)

[Bagan 2. Kerangka Pemikiran 9](#_Toc123909162)

# DAFTAR GAMBAR

**No table of figures entries found.**

# DAFTAR ISI

[DAFTAR BAGAN 2](#_Toc123912042)

[DAFTAR GAMBAR 2](#_Toc123912043)

[DAFTAR ISI 2](#_Toc123912044)

[PENDAHULUAN 3](#_Toc123912045)

[1. Latar Belakang 3](#_Toc123912046)

[2. Rumusan Masalah 6](#_Toc123912047)

[3. Batasan Masalah 6](#_Toc123912048)

[4. Tujuan Penelitian 6](#_Toc123912049)

[5. Manfaat Penelitian 6](#_Toc123912050)

[6. *The State of The Art* 7](#_Toc123912051)

[7. Kerangka Pemikiran 8](#_Toc123912052)

[8. Metode Penelitian 10](#_Toc123912053)

[9. Jadwal dan Lokasi Penelitian 11](#_Toc123912054)

[DAFTAR PUSTAKA 13](#_Toc123912055)

# PENDAHULUAN

## Latar Belakang

Msaukin berapa apps yang diupdate dalam beberapa bulan kebelakang (Sudah)

Mengapa kebanyakan aplikasi tidak melakukan update (belum)

Menjelaskan bad updates (sudah)

Fragile apps (done)

Apa yang menyebabkan bad updates (done)

Definisi software design (done)

Peran dalam pengembangan sebuah aplikasi (done)

Pengertian SOLID design (onworking)

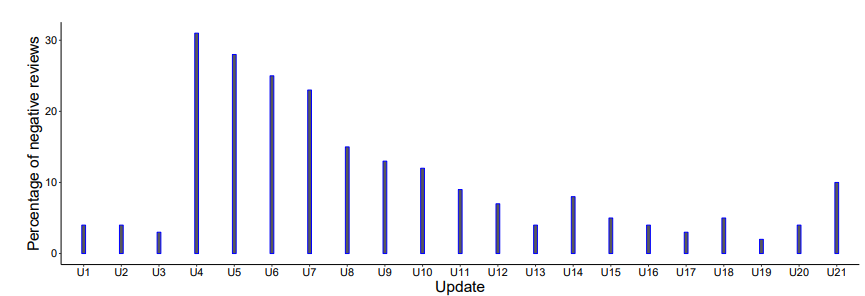
Why SOLID

How SOLID works

How SOLID solve the problem

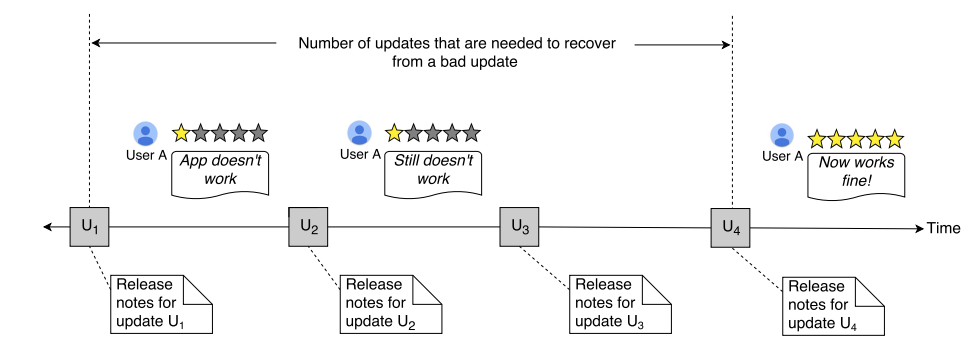
Jelasin mengenai judul yang diambil

Perangkat lunak yang baik merupakan perangkat lunak yang dapat memenuhi kebutuhan pengguna. Dengan berjalannya waktu kebutuhan user akan semakin bertambah. Oleh karena itu perangkat lunak juga harus fleksibel untuk melakukan perubahan. Menurut 42matters.com dalam rentang waktu 31 agustus 2022 sampai dengan 12 januari 2023 menunjukan bahwa 859.771 dari 3.792.074 apps di playstore melakukan update aplikasi. Hal tersebut menunjukan bahwa hanya 22.6% dari keseluruhan aplikasi yang terdapat dalam playstore dalam rentang waktu 4 bulan melakukan *update* aplikasi (42matters, 2022). Dengan data tersebut dapat dikatakan bahwasanya aplikasi yang melakukan *maintenance* hanyalah sedikit. sayangnya aplikasi yang melakukan update belum tentu menjadikan aplikasi lebih baik. Dalam penelitian yang dilakukan oleh Safwat Hassan, Cor-Paul Bezemer, dan Ahmed E. Hassan menunjukan bahwa 26.726 *update* dan *review* sebanyak 26.192.781 dari top 2.526 aplikasi tidak berbayar yang ada di google playstore memperoleh data yang ditunjukan pada grafik dibawah ini



Penulis membagi *updates* menjadi dua bagian *reguler update* atau rutinan dan juga *bad updates* atau merupakan sebuah *update* namun menimbulkan permasalahan. *Negative reviews* dari user untuk *update* yang dikategorikan sebagai *bad updates* berbeda dengan *negative reviews* dari user setelah *reguler updates*. Dari 250 top *bad updates* para user mengeluhkan mengenai fungsionalitas aplikasi yang bermasalah, penambahan biaya, perilaku *user interface* yang tidak sesuai dengan seharusnya, pengurangan fitur, dan *crashes*. (Hassan, Bezemer, & Hassan, 2018)

Untuk mengembalikan kepercayaan *user* terhadap aplikasi yang telah melakukan *bad updates* yaitu tentu saja dengan melakukan *fixing* dari *bugs* yang tedapat di dalam aplikasi. selain itu juga Ketika melukakan *updates* lebih baik cantumkan dalam *release notes* terkait issue yang berkemungkinan terdapat di dalam aplikasi. Hal tersebut dikarenakan *negative reviews* pada aplikasi yang telah melakukan *bad updates* memiliki *negative reviews* dengan median 1.9 sedangkan aplikasi yang secara eksplisit mencantumkan kemungkinan *issues* yang terdapat dalam aplikasi itu sekitar 1.7. dalam penelitian yang berjudul “*Studying Bad Updates of Top Free-to-Download Apps in the Google Play Store*” menjelaskan bahwa dalam mengembalikan kepercayaan user membutuhkan beberapa kali *updates* seperti yang tergambarkan pada gambar berikut



Seperti yang terlihat pada gambar di atas bahwa untuk mengembalikan kepercayaan *user* membutuhkan beberapa kali updates sehingga mengembalikan kepercayaan *user* (Hassan et al., 2018). Dengan mengharuskannya aplikasi unntuk sering melakukan *updates* dibutuhkannya sebuah arsitektur dari aplikasi yang baik sehingga jika melakukan perubahan persentase melakukan *bad updates* akan berkurang. Sehingga seorang *software architect* harus bisa menciptakan sebuah arsitektur perangkat lunak yang tidak rapuh dan fleksibel sehingga perangkat lunak dapat menyerap perubahan, berubah sementara dan mengambalikannya. Juga menjadikan aplikasi lebih baik setelahnya (Grassi & Mirandola, 2021)

Dalam pengembangan sebuah perangkat 70% itu digunakan dalam testing dan perawatan. Penjadwalan dan perkiraan biaya sangatlah buruk, perangkat lunak masih memiliki kualitas yang buruk dan produktifitas perangkat lunak tersebut meningkat lebih lambat. Schneidewind mengungkap bahwa 70% - 80% dari perangkat lunak yang ada diproduksi dengan *structured programming.* Karena sulitl menentukan bahwa perubahan kode akan mempengaruhi sesuatu. Juga masalah tersebesar dalam perawatan perangkat lunak, perangkat lunak tidak akan bisa dirawat apabila perangkat lunak tersebut tidak di *design* untuk perawatan. Dengan persentase yang besar untuk tahap perawatan perangkat lunak membuat *design p*perangkat lunak yang bagus sangatlah penting untuk kesukesan pengembangan perangkat lunak (Zuse, 2013)

Oleh karena itu dibutuhkannya sebuah software design yang matang bahkan pada saat proses pengembangan aplikasi di tahap awal. Software design bertanggung jawab pada tingkatan kode *design* seperti mengatur bagaimana tiap modul bekerja, mengatur ruang lingkup *classes*, menentukan tujuan tiap fungsi dan yang lainnya. Setiap struktur berisi bagian-bagian kode, hubungan di antaranya, dan properti dari setiap bagian dan relasi. Desain dari suatu sistem menggambarkan bagaimana sebuah sistem organisasi itu berperilaku. Perangkaian sistem desain sangatlah berdampak pada kualitas, kinerja, pemeliharaan, dan keseluruhan keberhasilan sistem. Sebuah sistem mewakili pengumpulan elemen yang mencapai satu set fungsi (Jaiswal, 2019).

Semakin besarnya sebuah aplikasi dan banyaknya permintaan penambahan fitur dari *stackholders,* algoritma dan struktur data bukan lagi menjadi masalah yang besar perihal *design* aplikasi. Hal ini dikarenakan jika sebuah aplikasi mempunyai algoritma yang baik namun *design* yang kurang baik ketika sebuah sistem dibangun dengan skala besar dan terus mengalami peningkatan maka akan menghasilkan sekumpulan masalah *design* yang baru. Masalah struktural seperti organisasi dan *global control* perangkat lunak yang busuk, data akses yang kurang baik, komposisi elemen *design,* dan pemilihan di antara alternatif *design* (Garlan & Shaw, 2011). Permasalahan tersebut dapat diselesaikan dengan pengelolaan sistem desain yang baik.

Dengan adanya software design membantu dalam pemahaman sistem secara keseluruhan selain dari itu juga software design dapat mengurangi biaya pengembangan sebuah aplikasi. Dikarenakan sebuah sistem membutuhkan pengembangan ataupun perbaikan diperllukannya sebuah aplikasi dengan kemampuan untuk di*maintenance* dengan mudah. Untuk melakukan hal tersebut terdapat beberapa hal yang perlu dihindari yaitu kekakuan, kerapuhan, imobilitas dan viskositas. Kekakuan merupakan sebuah masalah dimana sulitnya mengukur aplikasi jika terjadi perubahan. Kerapuhan disini merupakan kecendurangan perangkat lunak akan rusak jika terjadi perubahan. Imobilitas merupakan ketidakmampuan aplikasi untuk menggunakan Kembali perangkat lunak dari proyek lain atau bagian dari perangkat lunak dari proyek yang sama. Viskositas adalah ketidakmampuan untuk melestarikan desain sistem yang dapat menurun jika solusi yang tepat tidak dimasukkan sehubungan dengan perubahan apa pun dalam persyaratan sistem. Aplikasi yang mengalami hal tersebut terjadi dikarenakan sistem desain yang tidak baik. Untuk mengatasinya terdapat sebuah panduan yang diperkenalkan oleh Robert Martin yaitu SOLID desain principle (Singh & Hassan, 2015)

Dalam mengembangkan sebuah software. Sistem perangkat lunak harus dimulai dengan *clean code.* Jika diibaratkan dengan bangungan jika batanya tidak bagus, maka *blueprint* atau arsitektur perangkat lunak yang baik pun akan menghasilkan sebuah bangungan yang tidak baik. Begitupun sebuah bangungan jika memiliki memiliki *building blocks* yang baik namun memiliki arsitektur yang tidak baik berpotensi akan menghasilkan bangungan yang tidak baik. Hal tersebut juga berlaku pada saat pengembangan perangkat lunak. Oleh karena itu SOLID design principle ada (Martin, Grenning, Brown, Henney, & Gorman, 2018). SOLID *design principle* bertujuan untuk membuat *building blocks* yang baik dan juga arsitektur aplikasi yang baik sehingga aplikasi dapat secara fleksibel dikembangkan atau diperbaiki.

SOLID merupakan gabungan dari *principles* dalam pengembangan perangkat lunak *principles* tersebut diantaranya SRP (*single responsibility principle*), OCP (*open close principle)*, LSP (*liscov substitution principle)*, ISP (*interface segregation principle)*, DIP (*dependency inversion principle).* Yang pertama yaitu SRP merupakan salah satu principle dimana setiap *class, module,* atau *function* harus memiliki hanya memiliki satu tanggung jawab. Selanjutnya yaitu OCP, OCP merupakan sebuah aturan dimana entitas suatu perangkat lunak harus bersifat terbuka untuk pengembangan atau pelebaran tapi tertutup untuk perubahan. Jika dimisalkan bahwa sebuah *class, module,* atau *function* merupakan sebuah kotak mainan yang dapat dimasukkan dengan mainan lain, kotak mainan tersebut bersifat terbuka karena dapat ditambahkan mainan baru kedalamnya namun bersifat tertutup karena mainan di dalamnya masih bisa dimainkan. Seperti halnya sebuah program yang menggunakan prinsip OCP program tersebut jika terdapat penambahan hal baru atau mengganti cara program tersebut bekerja, hal tersebut harus tidak mengubah cara program tersebut bekerja.

*Principle* selanjutnya merupakan LSP, LSP merupakan salah satu principle dalam SOLID yang menerapkan bahwasanya setiap *subclass* harus dapat melakukan apa-apa saja yang dilakukan oleh *parent class*. Setelah *principle* LSP selanjutnya adalah ISP. ISP merupakan sebuah *principle* yang mana memisahkan *interface-interface* agar *class* yang bergantung pada *interface* tersebut hanya mengambil apa-apa saja yang dibutuhkan. Dan *principle* yang terakhir merupakan DIP. DIP merupakan sebuah *principle* dalam SOLID *design principle* dimana *high-level module* yang menerapkan kompleks logic dalam sebuah aplikasi harus bisa digunakan Kembali dan tidak terpengaruh Ketika *low-level modules* terjadi perubahan. Jadi kedua high-level module dan low-level module tidak boleh tergantung satu sama lain tetapi bergantung kepada abstractions. Dan abstranctions tidak boleh bergantung kepada implementasinya. Dan implementasinya harus bergantung kepada absctractions. (Martin et al., 2018). Dengan penelitian-penelitian tersebut penulis ingin meliti bagaimana fleksibilitas dari aplikasi setelah menerapkan

Untuk objek (aplikasi) yang akan diterapkan SOLID *design principle* yaitu media pembelajaran online. Aplikasi tersebut merupakan sebuah aplikasi berbasis website yang berisi mengenai kumpulan persoalan mengenai *competitive programming.* Competitive programming merupakan aktifitas untuk memecahkan sebuah permasalahan *well-known computer science*  yang merupakan sebuah permasalahan yang jawabanya sudah diketahui. Persoalan-persoalan tersebut diberikan dengan Batasan-batasan tertentu dan juga waktu tertentu (Halim, Halim, Skiena, & Revilla, 2013). Ide dari aplikasi ini yaitu *user* mendapatkan soal-soal *competitive programming* diberikan dan *user* diwajibkan untuk menyelesaikan soal tersebut.

Hal yang mendasari penulis mengambil objek tersebut ialah karena dalam sebuah penelitian yang berjudul “*Increasing Employability of Indian Engineering Graduates through Experiential Learning Programs and Competitive Programming: Case Study*” mengatakan bahwa menurut statistik di laporan NASSACOM memperkirakan bahwasanya dari 3 juta orang yang bergabung dalam industri IT hanya 25% dari lulusan dengan background IT yang *employable*. Aspiring Mind telah melakukan pengujian berbasis komputer yang disebut dengan AMCAT kepada 100.000 pelajar di 650 institusi berbeda, ujian tersebut untuk mengukur kelayakan bekerja untuk lulusan Teknik. Hasilnya menunjukan bahwa 47% tidak layak di berbagai sector. 17,91% layak sebagai software service, 3,67% layak sebagai software products dan 40,57% layak sebagai BPO. Hanya 3,84% lulusan *start-up ready* dan 6.56% layak di bidang *design.* Melihat angka tersebut menunjukan bahwa terdapat urgensi untuk meningkatkan kelayakan para lulusan IT di bidang industry saat ini. Ada sebuah cara untuk mengatasi hal tersebut yaitu mengasah *lateral thinking* dan *thinking out-of-the-box* melalui *competitive programming* (Nair, 2020)

## Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah pada penelitian ini adalah:

* Bagaimana mengukur fleksibilitas sebuah perangkat lunak berorientasi objek dengan menggunakan *the halstead metrics*
* Bagaimana fleksibilitas sebuah perangkat lunak jika menerapkan SOLID *design principle*

## Batasan Masalah

Batasan masalah dari penelitian ini adalah sebagai berikut:

* Bahan uji dalam penelitian ini hanya perangkat lunak media pembelajaran pemrograman
* Metode untuk menguji fleksibilitas pada penilitian ini yaitu *the halsteaed metrics*

## Tujuan Penelitian

Tujuan dari penelitian ini sebagai berikut:

* Mengimplementasikan *the halstead metrics* dalam mengukur fleksibilitas perangkat lunak
* Menganalisis efektifitas penggunaan metode *the halstead metrics* dalam mengukur fleksibilitas sebuah perangkat lunak

## Manfaat Penelitian

Manfaat yang diharapkan dari hasil penelitian ini adalah sebagai berikut:

* Mengetahui fleksibilitas dari perangkat lunak untuk pemeliharaan
* Mengetahui efektifitas penggunaan metode *the halstead metrics* dalam mengukur fleksibilitas sebuah perangkat lunak

## *The State of The Art*

Penelitian sebelumnya membantu peneliti untuk proses Analisa dan memperbanyak bahasan penelitian, penelitian ini menjadi sebuah pengembangan dari penelitian-penelitian sebelumnya yang telah dilakukan. Dalam penelitian ini menyertakan 5 jurnal international yang berhubungan mengenai *software maintainability.* Berikut jurnal-jurnal tersebut:

1. Penelitian yang berjudul “*Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics”* membahas mengenai identifikasi potensi masalah yang akan muncul dengan memanfaatkan *software metrics* dalam proses pengembangan. Metrics yang digunakan pun berupa metrics yang sudah tervalidasi. Tujuan dari penelitian ini yaitu mengetes kevalidan tiga buah OO design complexity metrics yang sudah ada dan juga meguji kemampuan masing-masing metrics untuk melakukan predisksi waktu perawatan. Hasil dari penelitian ini berupa hasil validitas dari ketiga metrics tersebut yaitu Intercation Level (IL), Interface Size (IS), dan Operation Argument Complexity (OAC). Hasilnya yaitu bahwa dari ketiga metrics tersebut terbukti bahwa ketiga metrics tersebut bermanfaat dalam menghitung waktu perawatan.
2. Penelitian yang berjudul “*Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison”* membahas mengenai identifikasi OO (object-oriented) metrics yang bisa digunakan sebagai prediktor yang tepat untuk kepentingan perawatan perangkat lunak. Penelitian ini dibangun dengan data historis yang dikumpulkan dari *history* perawatan dari *medium-sized* sistem Object-oriented. Penelitian ini menggunakan history perawatan dari dua buah sistem sebagai pembuktian bahwa metrics yang disarankan dengan *software quality attributes*. Hasilnya membuktikan bahwa *size* dan *import direct coupling metrics* merupakan prediktor yang terkuat untuk menghitung *maintainability of classes.* Sementara turunan, cohesion, dan penghitungan *indirect/export coupling* bukan merupakan prediktor yang kuat.
3. Penelitian yang berjudul “*Construction and Testing of Polynomials Predicting Software Maintainability*” penelitian ini menunjukan bagaimana kumpulan Langkah-langkah yang untuk dengan metrics-metrics yang dapat diaplikasikan untuk mengukur *software maintainability* dapat dikurangi untuk memprediksi *software maintainability*. Polinomial penilaian perawatan yang disajikan di sini adalah model akurat dari data uji yang mereka dibangun. Model yang disajikan bukanlah satu-satunya model yang ada sebagai pengukur juga bukan model yang terbaik dalam keseluruhan, namun model ini bekerja secara otomatis yang dapat digunakan secara cepat dan mudah dalam mengukur *software maintainability*.
4. Penelitian yang berjudul “*Software Maintainability: Systematic Literature Review and Current Trends*” dalam penelitian ini menunjukan mengenai sistematik review mengenai *software maintainability* dari tahun 1991 hingga tahun 2015, secara total terdapat 96 penelitian yang teridentifikasi. Penelitian-penelitian tersebut disusun dalam bentuk yang terstruktur dan dianalisis dengan berbagai perspektif seperti *design metrics, prediction model, tools, data sources* dan *prediction accuracy.* Hasilnya menunjukan bahwa penggunaan machine learning dalam memprediksi *software maintainability* semakin meningkat sejak tahun 2005. Penggunaan *evolutionary algorithm* juga mulai digunakan untuk sub-field yang berhubungan semenjak tahun 2010. *Design metrics* masih menjadi pilihan yang disukai untuk mendapatkan karakteristik dari perangkat lunak yang diberikan sebelum melakukan *deployment* lebih jauh dalam model prediksi untuk menentukan pemeliharaan perangkat lunak yang sesuai. Dalam penelitian ini menunjukan bahwa metrics yang banyak digunakan untuk saat ini yaitu C&K metric dan Li dan Henry.
5. Penelitian yang berjudul “*METRICS TO QUANTIFY SOLID SOFTWARE DESIGN PRINCIPLES”*  menjelaskan mengenai *metrics* salah satu *design principle* untuk aplikasi berbasis OO yaitu SOLID *design principle.* dalam penelitian ini bertujuan untuk mengidentifikasi metrics yang digunakan untuk peniliaian kuantitatif SOLID *design principle.* penelitian ini menumukan bahwa Chidamber and Kemerer (CK) telah tervalidasi oleh beberapa peneliti sebagai rawan dari kesalahan. Penelitian ini menghasilkan bahwa design version yang menerapkan SOLID *design principle* terbukti memiliki kualitas yang lebih baik karena dapat meningkatkan *cohesion* dan *reuseability*. (Dubey & Rana, 2010)
6. Penelitian yang berjudul “*A COMPREHENSIVE REVIEW AND ANALYSIS ON OBJECT-ORIENTED SOFTWARE METRICS IN SOFTWARE MEASUREMENT”* penelitian yang dilakukan oleh K.P. Srinivasan dan Dr.T. Devi pada tahun 2014 menjelaskan bahwa penggunaan *metrics* perangkat lunak telah membuktikan efisiensi proses dan efektifitas sebuah produk. Dengan banyaknya *software metrics* yang ada penelitian ini ingin melakukan *review* atas *software metrics* yang ada. Metrics yang direview antara lain Result Based Software Metrics (RBSM), The Chidamber dan Kemerer (CK), Metrics For Object-Oriented (MOOD), dan The Lorenz dan Kidd (L-K). dari review-review tersebut menghasilkan bahwasanya meskipun CK metrics menjadi salah satu metrics yang tertua namun masih menjadi metrics yang dapat diandalkan dibandingkan dengan metrics lainnya. (Srinivasan & Devi, 2014)

// empirical analysis of ck metrics

1. dalam penelitian yang berjudul "Assessment of Maintainability Metrics for Object-Oriented Software System" menjelaskan bahwa pentingnya maintainability dalam pengembangan software bahkan sebelum tahap pengembangan fitur yang sangat penting dalam object-oriented metrics yaitu kemampuan untuk mencakup beberapa isu seperti classes, methods, pewarisan, encapsulation, dan polimorpishm. Chidamber dan kemerer merupakan peneliti yang paling banyak dikutip. mereka mendefinisikan enam metrics. *Weighted methods per class (*WMC), *Response sets for Class* (RFC), *lack of cohesion in methods* (LCOM), *coupling between object classes* (CBO), *depth of inheritance tree of a class* (DIT), dan *number of children of a class* (NOC) CK metrics dapat digunakan untuk menganalisa *coupling*, *cohsesion*

dan *complexity* dengan baik (Dubey & Rana, 2010)

## Kerangka Pemikiran

Dengan sedikitnya developer yang melakukan perawatan perangkat lunak dari keselurhan aplikasi yang telah di-*deploy* menjadi sebuah masalah dalam pengembangan perangkat lunak itu sendiri. Perawatan perangkat lunak memang suatu hal yang sulit dilakukan karena dibutuhkan perencanaan yang matang dan biaya yang cukup, melihat bahwasanya lebih dari setengah biaya dilakukang saat proses *maintenance*. Sehingga, ketika sebuah aplikasi dibuat tanpa perencanaan yang matang, maka aplikasi tersebut akan susah untuk dilakukan *maintenance* dimana hal tersebut akan menyebabkan aplikasi akan susah melakukan *update* dan kebutuhan *user* pun tidak akakn terpenuhi.

jumlah data untuk proses *training*, resolusi gambar, dan kondisi pencahayaan pada gambar

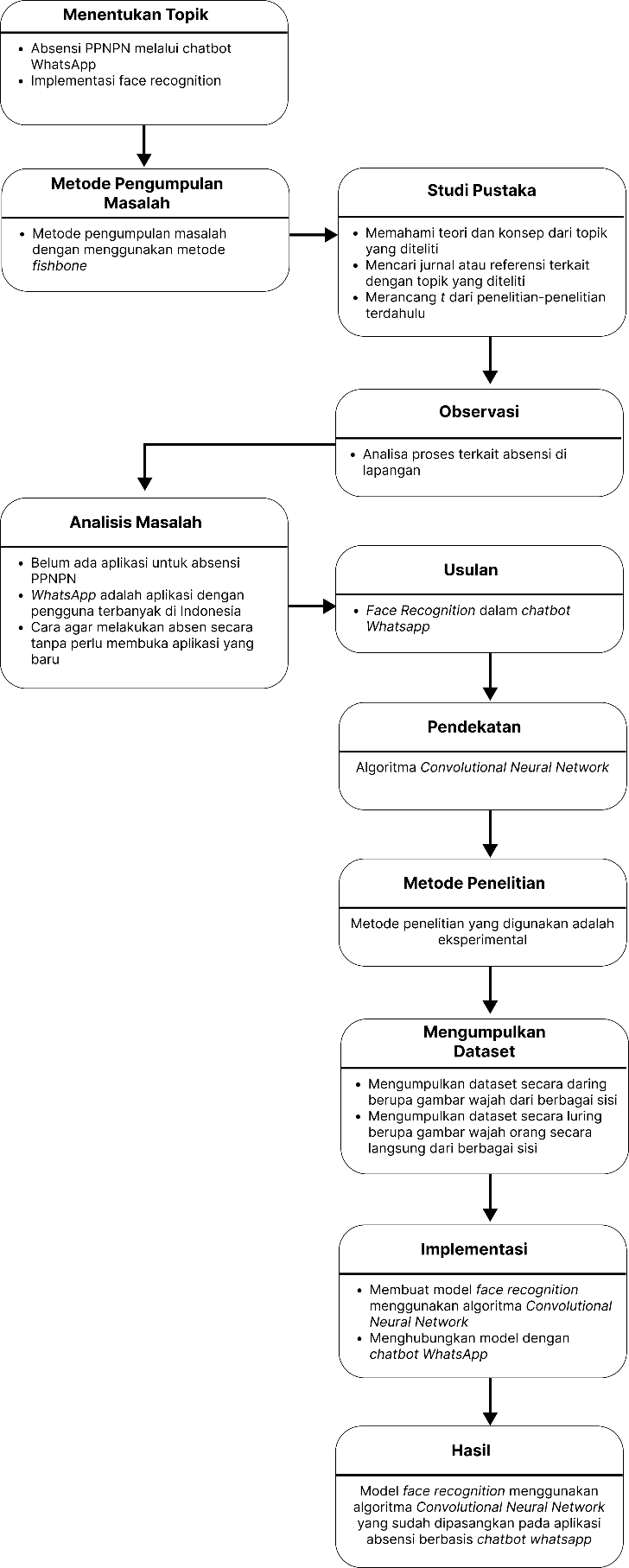
Tingkat akurasi model *face recognition* pada *chatbot WhatsApp*

Variabel Independen

Variabel Dependen

Bagan 1. Variabel yang digunakan

Adapun variabel yang digunakan dalam penelitian ini terbagi menjadi dua variabel, yaitu variabel Independen dan variabel dependen. Variabel independent merupakan variable yang mempengaruhi atau menjadi penyebab perubahan pada variabel dependen. Sedangkan variabel dependen adalah tipe variabel yang dijekaskan atau dipengaruhi oleh variabel independent.



Bagan 2. Kerangka Pemikiran

Pada bagan 1 di atas menunjukan kerangka pemikiran yang merupakan bentuk pola pikir pada penelitian ini. Diawali dengan menentukan topik yang akan diangkat. Selanjutnya melakukan analisis pengumpulan masalah dengan metode fishbone. Setelah melakukan metode pengumpulan masalah selanjutnya melanjutkan studi pustaka yang berhubungan dengan hasil dari pengumpulan masalah. Langkah selanjutnya adalah melakukan observasi secara langsung di lapangan untuk menselaraskan masalah yang sudah ditemukan dengan kondisi lapangan sebenarnya. Selanjutnya melakukan analisis masalah sehingga menemukan masalah masalah yang harus diselesaikan. Setelah menemukan masalah langkah selanjutnya adalah dengan memberikan usulan, merancang metode penelitian, mengumpulkan dataset, dan mengimplementasikan menjadi sebuah produk yang dapat digunakan untuk menyelesaikan masalah. Hasil akhir akan dibuat dalam bentuk laporan tugas akhir berupa hasil dari penyelesaian masalah.

## Metode Penelitian

Metode penelitian yang digunakan dalam peneltian ini adalah metode eksperimental. Penelitian eksperimental adalah penelitian yang digunakan agar bisa mengetahui pengaruh pembelrian perlakuan terhadap subjek tertentu. Penelitian ini menggunakan metode eksperimental karena objek yang diproses melalui suatu eksperimen yang menggunakan algoritma *Convolutional Neural Network* (CNN) dengan menambahkan implementasi pada *chatbot* *WhatsApp*. Adapun rencana langkah-langkah kegiatan penellitian meliputi sebagai berikut:

* Studi Literatur *(Study Research)*. Pada tahap ini penulis melakukan proses pengumpulan referensi yang berkaitan dengan pengolahan citra, jaringan saraf tiruan, *face recognition*, dan algoritma *convolutional neural network* (CNN). Tahapan ini dilakukan agar penulis mengetahui apa yang akan dikerjakan dan apa yang sudah dikerjakan oleh peneliti sebelumnya.
* Analisis Permasalahan. Pada tahap ini penulis akan mengidentifikasi masalah apa saja yang akan dipecahkan berdasar kepada referensi yang telah ditemukan.
* Pengumpulan Data *(Data Collection).* Pada tahap ini penulis akan melakukan proses riset dimana penulis menerapkan metode ilmiah dalam mengumpulkan data secara sistematis untuk dianalisa.
* Implementasi. Pada tahap ini akan dilakukan penerapan dari algoritma *Convolutional Neural Network* (CNN) untuk *face recognition* pada pendeteksi wajah down syndrome dengan proses *training* dan *testing* menggunakan data yang sudah dikumpulkan pada tahap sebelumnya. Berikut ini merupakan langkah kerja pada penelitian ini:

1. *Retraining* *MobileNets* model. *MobileNets* merupakan salah satu arsitektur dari algoritma Convolutional Neural Network (CNN) yang dapat digunakan untuk mengatasi kebutuhan akan computing resource berlebih. *MobileNets* dapat digunakan untuk klasifikasi citra salag satunya untuk *face recognition*
2. *Capture* data. Data akan diambil melalui web dengan menggunakan webcam untuk mendapatkan dataset yang akan dikumpulkan untuk dilatih.
3. Membagi data ke dalam kelas-kelas. Pata tahap ini kita akan membagi data berdasarkan ID sehingga dapat dikenali sebagai wajah orang dengan pemilik ID tersebut.
4. *Training* *network*. Langkah selanjutnya adalah dengan melatih jaringan menggunakan data yang sudah dimiliki sebelumnya dari proses *capture*.
5. Input gambar dari *chatbot* *WhatsApp*. Pada tahap gambar akan diinput pada *chatbot* *WhatsApp* dan dimasukkan ke dalam database.
6. Web akan mengambil gambar yang sudah diinput melalui *chatbot* *WhatsApp* dari database kemudian dilakukan pengujian pada model yang telah dibuat. Keluaran dari proses ini akan berupa label yang akan dimasukkan ke dalam database berupa hadir (data valid dan ID sesuai) dan tidak hadir (data tidak valid dan ID tidak sesuai)

* Evaluasi dan Analisis Hasil. Pada tahap ini akan dilakukan evaluasi dan analisis dari model yang dihasilkan oleh algoritma *Convolutional Neural Network* (CNN) untuk *face recognition* pada absensi berbasis *chatbot* *WhatsApp*.

## Jadwal dan Lokasi Penelitian

**Lokasi Penelitian**

Lokasi penelitian yang akan dilakukan berada pada bagian umum Direktorat Jenderal Pendidikan Islam Kementrian Agama.

**Jadwal Penelitian**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **NO** | **KEGIATAN** | **MINGGU** | | | | | | | | | | | | **HASIL KESELURUHAN** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| 1 | Studi Pustaka |  |  |  |  |  |  |  |  |  |  |  |  | Cara penerapan algoritma pada model dan cara penerapan model pada aplikasi *chatbot* *WhatsApp* |
| 2 | Analisis variabel yang dibutuhkan |  |  |  |  |  |  |  |  |  |  |  |  | Variabel data yang akan diajukan |
| 3 | Pengumpulan data dan Data Cleaning |  |  |  |  |  |  |  |  |  |  |  |  | Data yang sudah siap untuk dibuat model |
| 4 | Pembangunan Model |  |  |  |  |  |  |  |  |  |  |  |  | Model *face recognition* yang siap untuk dievaluasi |
| 5 | Evaluasi model |  |  |  |  |  |  |  |  |  |  |  |  | Hasil analisis dari proses evaluasi berupa hasil pengujian dan validasi terhadap model |
| 6 | Deployment pada aplikasi |  |  |  |  |  |  |  |  |  |  |  |  | *Chatbot* *WhatsApp* yang sudah bisa digunakan untuk proses absensi |
| 7 | Penyusunan Laporan |  |  |  |  |  |  |  |  |  |  |  |  | Laporan Tugas Akhir |

# DAFTAR PUSTAKA

42matters. (2022). Updated apps in the last 135 days. Retrieved from https://42matters.com/app-market-explorer/android/5b891991c8453b038008897b

Dubey, S. K., & Rana, A. (2010). Assessment of usability metrics for object-oriented software system. *ACM SIGSOFT Software Engineering Notes*, *35*(6), 1–4.

Garlan, D., & Shaw, M. (2011). An Introduction to Software Architecture. *School of Computer Science, Carnegie Mellon University, June*.

Grassi, V., & Mirandola, R. (2021). The Tao way to anti-fragile software architectures: the case of mobile applications. *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, 86–89. IEEE.

Halim, S., Halim, F., Skiena, S. S., & Revilla, M. A. (2013). *Competitive programming 3*. Lulu Independent Publish Morrisville, NC, USA.

Hassan, S., Bezemer, C.-P., & Hassan, A. E. (2018). Studying bad updates of top free-to-download apps in the google play store. *IEEE Transactions on Software Engineering*, *46*(7), 773–793.

Jaiswal, M. (2019). Software architecture and software design. *International Research Journal of Engineering and Technology (IRJET) e-ISSN*, 56–2395.

Martin, R. C., Grenning, J., Brown, S., Henney, K., & Gorman, J. (2018). *Clean architecture: a craftsman’s guide to software structure and design*. Prentice Hall.

Nair, P. R. (2020). Increasing employability of Indian engineering graduates through experiential learning programs and competitive programming: Case study. *Procedia Computer Science*, *172*, 831–837.

Singh, H., & Hassan, S. I. (2015). Effect of solid design principles on quality of software: An empirical assessment. *International Journal of Scientific & Engineering Research*, *6*(4).

Srinivasan, K. P., & Devi, T. (2014). A comprehensive review and analysis on object-oriented software metrics in software measurement. *International Journal on Computer Science and Engineering*, *6*(7), 247.

Zuse, H. (2013). *A framework of software measurement*. Walter de Gruyter.