**PROPOSAL TUGAS AKHIR**

**ANALISIS MAINTAINABILITY PERANGKAT LUNAK MENERAPKAN DESIGN PRINCIPLES DENGAN *CHIDAMBER AND KEMERER* (CK) 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**

**2023**

# DAFTAR ISI

[DAFTAR ISI i](#_Toc129030992)

[DAFTAR GAMBAR ii](#_Toc129030993)

[DAFTAR TABEL iii](#_Toc129030994)

[PENDAHULUAN 1](#_Toc129030995)

[1. Latar Belakang 1](#_Toc129030996)

[2. Rumusan Masalah 5](#_Toc129030997)

[3. Batasan Masalah 5](#_Toc129030998)

[4. Tujuan Penelitian 6](#_Toc129030999)

[5. Manfaat Penelitian 6](#_Toc129031000)

[6. *The State of The Art* 6](#_Toc129031001)

[7. Kerangka Pemikiran 10](#_Toc129031002)

[8. Metode Penelitian 11](#_Toc129031003)

[1. Perumusan Masalah 11](#_Toc129031004)

[2. Studi literatur 11](#_Toc129031005)

[3. Pengembangan objek teliti 12](#_Toc129031006)

[4. Pengukuran maintainability 12](#_Toc129031007)

[5. Analisis hasil penelitian 12](#_Toc129031008)

[6. Jadwal dan Lokasi Penelitian 12](#_Toc129031009)

[DAFTAR PUSTAKA 14](#_Toc129031010)

# DAFTAR GAMBAR

[Gambar 1 Negative Review for each updates 1](#_Toc129030782)

[Gambar 2 updates to recover consumen's satisfaction 2](#_Toc129030783)

[Gambar 5 Metode Penelitian 11](#_Toc129030784)

# DAFTAR TABEL

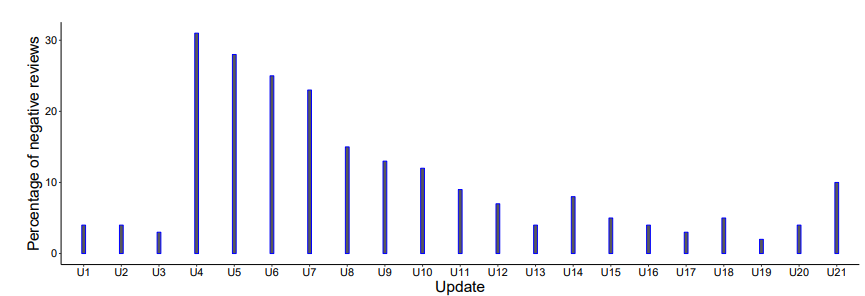
[Tabel 1 State of the art 9](#_Toc129030931)

[Tabel 2 Kerangka Pemikiran 10](#_Toc129030932)

# PENDAHULUAN

## Latar Belakang

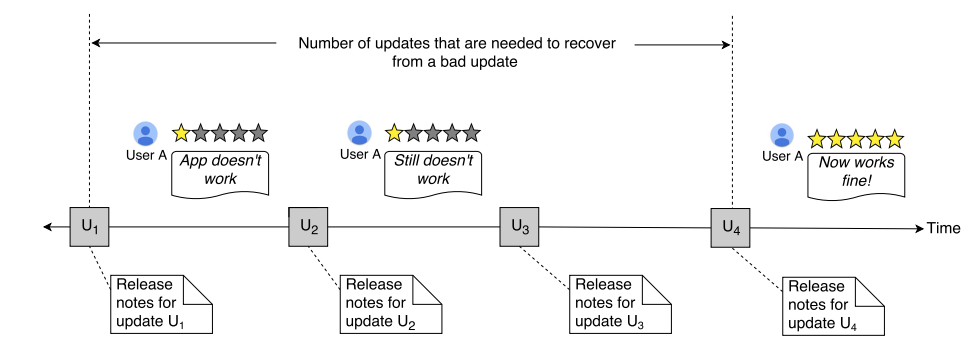
Perangkat lunak yang baik merupakan perangkat lunak yang dapat memenuhi kebutuhan pengguna. Dengan berjalannya waktu kebutuhan user akan semakin bertambah. Oleh karena itu perangkat lunak juga harus fleksibel untuk melakukan perubahan. Menurut 42matters.com dalam rentang waktu 31 agustus 2022 sampai 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 [1]. 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 dari 26.726 *update* mendapatkan *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



Gambar 1 Negative Review for each updates

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*. [2]

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 melakukan *updates* lebih baik mencantumkan issue yang memungkinkan akan terjadi pada aplikasi dalam *release notes*. 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 2.



Gambar 2 updates to recover consumen's satisfaction

Seperti yang terlihat pada Gambar 2 di atas bahwa untuk mengembalikan kepercayaan *user* membutuhkan beberapa kali updates sehingga mengembalikan kepercayaan *user* [2]. Dengan mengharuskannya aplikasi unntuk sering melakukan *updates* dibutuhkannya sebuah desian dari aplikasi yang baik sehingga jika melakukan perubahan persentase melakukan *bad updates* akan berkurang. Sehingga seorang pengembang aplikasi harus bisa menciptakan sebuah desain perangkat lunak yang tidak rapuh dan fleksibel sehingga perangkat lunak dapat mengadaptasi perubahan, perubahan sementara dan mengambalikannya. Juga menjadikan aplikasi lebih baik setelahnya [3]

Dalam pengembangan sebuah perangkat 70% itu digunakan dalam testing dan perawatan. Dalam pengembangan sebuah perangkat lunak 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 sulit 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 didesain untuk perawatan. Dengan persentase yang besar untuk tahap perawatan perangkat lunak membuat desain *p*perangkat lunak yang bagus sangatlah penting untuk kesukesan pengembangan perangkat lunak [4]

Oleh karena itu dibutuhkannya sebuah software desain yang matang bahkan pada saat proses pengembangan aplikasi di tahap awal. Software desainer bertanggung jawab pada tingkatan kode desain 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 [5].

Semakin besarnya sebuah aplikasi dan banyaknya permintaan penambahan fitur dari *stackholders,* algoritma dan struktur data bukan lagi menjadi masalah yang besar perihal desainaplikasi. Hal ini dikarenakan meskipun sebuah aplikasi mempunyai algoritma yang baik namun desainyang kurang baik, ketika sebuah sistem dibangun dengan skala besar dan terus mengalami peningkatan maka akan menghasilkan sekumpulan masalah desainyang 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* [6]. Permasalahan tersebut dapat diselesaikan dengan pengelolaan sistem desain yang baik.

Dengan adanya software desain 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 diperlukannya 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 [7]

Dalam mengembangkan sebuah perangkat lunak. Sistem perangkat lunak harus dimulai dengan *clean code.* Jika diibaratkan dengan sebuah 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 desain 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 [8]. 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. [8]. Dengan penelitian-penelitian tersebut penulis ingin meliti bagaimana fleksibilitas dari aplikasi setelah menerapkan

Untuk objek (aplikasi) yang akan diterapkan SOLID *design principle* yaitu aplikasi kepegawaian PPNPN. Aplikasi ini berupa aplikasi pendataan para pegawai pemerintah non pegawai negeri yang khusus diperuntukan untuk pegawai pemerintah non pegawai negeri (PPNPN). Aplikasi kepegawaian merupakan salah satu aplikasi yang cukup dinamis dan membutuhkan banyak peningkatan seiring berkembangnya waktu. Dengan demikian aplikasi mengharuskan sebuah desian aplikasi yang dapat memenuhi kebutuhan tersebut.

## Rumusan Masalah

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

1. Bagaimana mengukur *maintainability* sebuah perangkat lunak berorientasi objek dengan menggunakan Chidamber and Kemerer *(CK) metrics*?
2. Bagaimana kemudahan sebuah perangkat lunak dalam melakukan pembaruan jika menerapkan SOLID *design principle* ?

## Batasan Masalah

Batasan masalah dari penelitian ini adalah sebagai berikut:

1. Bahan uji dalam penelitian ini hanya perangkat lunak kepegawaian
2. Metode untuk mengukur *maintainability* pada penilitian ini yaitu Chidamber and Kemerer *(CK) metrics*

## Tujuan Penelitian

Tujuan dari penelitian ini sebagai berikut:

1. Mengimplementasikan Chidamber and Kemerer *(CK) metrics* dalam mengukur *maintainability* perangkat lunak.
2. Menganalisis kemudahan sebuah aplikasi dengan menerapkan SOLID design principle dalam melakukan pembaruan.

## Manfaat Penelitian

Manfaat yang diharapkan dari hasil penelitian ini adalah sebagai berikut:

1. Mengetahui kemudahan aplikasi dengan menerapkan SOLID design principle dalam melakukan pembaruan dari perangkat lunak menggunakan C&K metrics.

## *The State of The Art*

Beberapa Penelitian telah dilakukan untuk membuat software maintainability yang baik dari masa ke masa. Dalam upaya mengembangkan sebuah aplikasi dengan metode yang telah dikembangkan maka dibutuhkannya proses studi literatur. Berikut merupakan penelitian yang sebelumnya telah dilakukan dengan metode yang serupa:

1. R Prajana, dkk (2021). Dalam jurnalnya yang berjudul *Adoption of Desgin principles and Design patterns for developing Software Application.* Membahas mengenai “SOLID” design principle dan design patterns yang meliputi Factory Method Pattern dan Singleton Design Pattern untuk membantu software architecture pemula dalam menghindari kesalahan dalam membuat *object-oriented* desain. Penerapan “SOLID” dan design patterns ini digunakan pada aplikasi Inpatient pharmacy medication management dengan menggunakan .NET framework dan Bahasa pemrograman C#. pengukuran dengan menggunakan NDepend hasilnya menunjukan bahwa *abstractness metric (*metric yang digunakan) menunjukan hasil yang baik untuk software *maintainability* [9].
2. Dr Sunnil Sikka (2018). Dalam jurnalnya yang berjudul Metrics To Quantify SOLID Software Design Principles membahas mengenai sebuah aplikasi dengan menerapkan “SOLID” design principles dengan menggunakan Chidamber and Kemerer (*CK)* metrics.Hasilnya mengungkakpkan bahwasanya dibutuhkannya metrics baru yang memang dikhususkan untuk mengukur “SOLID” design principles [10].
3. Apostolos Ampatzoglou, dkk (2019). Dalam prosiding yang berjudul Applying the single responsibility principle in industry: Modulatiry Benefits and Trade-offs membahas mengenai metode *single responsibility principle* (SRP) mengimplementasikan dalam p[3]rojek sebuah sistem yang memiliki lebih dari 1500 *classes*. penerapan *single responsibility principle* (SRP) menghasilkan *modularity* hasilnya dari prosiding ini adalah bahwasanya single responsibility principle (SRP) dapat membuat coupling yang rendah dan cohesion yang tinggi [11].
4. Aisha Kurmangali, dkk (2019). Dalam jurnal yang berjudul Impact of Abstract Factory and Decorator Design Patterns on Software Maintainability: Empirical Evaluation using CK Metrics. Menjelaskan mengenai dua buah design principles abstract factory dan decorator design patterns dalam menciptakan sebuah software yang maintainability dengan pengukuran menggunakan Chidamber and Kemerer (*CK)* metrics. Hasilnya menunjukan bahwa kedua design patterns tersebut secara positif mempengaruhi software maintainability [12].
5. Bhaumik Tyagi, dkk (2022). Dalam jurnal yang berjudul An Experimental Assesment on Effects of Solid Design Principles on the quality of Software using CKJM Metrics Analysis. Menjelaskan prinsip-prinsip yang terdapat Solid terhadap software desain yang baik. Setelah SOLID diimplementasikan dalam class diagram. Dari proses tersebut diuji dengan menggunakan CKJM metrics menghasilkan bahwasanya design principles dapat membuat arsitektur perangkat lunak menjadi lebih *scalable* [13]*.*

| **No** | **Judul Jurnal dan Peneliti** | **Metode** | **Tujuan** |
| --- | --- | --- | --- |
| 1 | Adoption of Design principles and Design patterns for Developing Software Application  Peneliti:  R Prajana, Prof.Kavitha S N (2021) | SOLID design principle, Factory Method pattern, dan Singleton Design Pattern | Hasil dari penelitian ini berupa effect dari penggunaan prinsip desain SOLID, hasilnya menunjukan bahwa *abstractness metric (*metric yang digunakan) menunjukan hasil yang baik untuk software *maintainability*. [9] |
| 2 | Metrics to Quantify SOLID Software Design Principles  Peneliti:  Dr. Sunil Sikka, 2018 | SOLID design principle with CK metrics. | Hasil dari penelitian ini memvalidasi hipotesis peneliti bahwa SOLID *design principle* merupakan sebuah design principle yang dapat membuat desain object oriented semakin dan membuat kualitas perangkat lunak semakin baik pula. |
| 3. | Applying the Single Responsibility Principle in Industry: Modularity Benefits and Trade-offs  Penulis:  Apostolos Ampatzoglou, dkk (2019) | Single responsibility Principle (SRP) | Menerapkan single responsibility principle (SRP) untuk melihat kelebihan dan kekurangan dari design patterns tersebut. |
| 4. | Impact of Abstract Factory and Decorator Design Patterns on Software Maintainability: Empirical Evaluation using CK Metrics, 2022 Malaysia  Peneliti:  Aisha Kurmangali, Muhammad Ehsan Rana, Wan Nurhayati Wan Ab Rahman (2022) | Abstract Factory and Decorator Design patterns | Penelitian ini membahas mengenai dua buah design principle yaitu abstract factory dan decorator dengan pengukuran menggunakan CKJM. Penelitian ini menghasilkan bahwasanya design patterns secara positif mempengaruhi software maintainability |
| 5. | An Experimental Assessment on Effects of Solid Design Principles on the quality of Software using CKJM Metric Analysis  Peneliti:  Bhaumik Tyagi, Yusra Beg (2022) | SOLID design principle with CKJM metrics | Jurnal penelitian ini menekankan pengawasan eksperimental untuk membuktikan prinsip-prinsip desain SOLID dengan menggunakan CKJM (Chidamber & Kemerer Java Metrics). |

Tabel 1 State of the art

Pada Tabel 1 telah dipaparkan mengenai beberapa penelitian yang beririsan dengan penulis bahkan serupa. Seperti yang terlihat bahwa penelitian-penelitian sebelumnya pengukuran dengan Chidamber and Kemerer (CK) metrics terhadap software dengan menerapkan SOLID design principle. penelitian ini merupakan pengembangan dari penelitian sebelumnya dimana belum adanya indikasi *maintainability* dari hasil Chidamber and Kemerer (CK) metrics yang telah didapatkan.

## Kerangka Pemikiran

Kerangka pemikiran dalam penelitian tugas akhir ini akan dipaparkan pada Tabel 2 sebagai berikut:

|  |
| --- |
| Fenomena |
| Software yang terdapat di playstore hanya 22.6% aplikasi saja yang melakukan update dari rentang waktu 4 bulan. |

|  |
| --- |
| Problem |
| Dengan tidak menerapkannya aturan dalam menuliskan sebuah code. Menyebabkan perangkat lunak sulit untuk dilakukan *maintenance* dan juga sukar untuk dimengerti antar *developer.* |

|  |
| --- |
| Approach |
| Menggunakan “SOLID” Design principles dalam proses pengembangan software. |

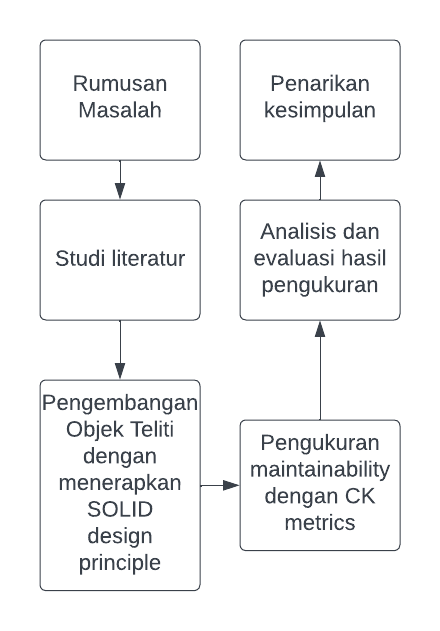
|  |
| --- |
| Software Development |
| Metode perancangan: UML Class diagram  Metode pengembangan: Agile Software Development |

|  |
| --- |
| Software Implementation |
| Bahasa Pemrograman: Javascript  Database: Postgresql  IDE: Visual Studio Code  Framework Javascript: NextJS |

Tabel 2 Kerangka Pemikiran

## Metode Penelitian

Penelitian ini dilakukan dengan beberapa tahap pengerjaan. Tahap-tahap tersebut di antara lain rumusan masalah, studi literatur, pengembangan objek yang akan diteliti, pengukuran *maintainability,* analisis dan evaluasi hasil pengukuran, dan terakhir penarikan kesimpulan. Alur tahapan tersebut dapat dilihat di gambar berikut:



Gambar 5 Metode Penelitian

## Perumusan Masalah

Tahap perumusan masalah merupakan tahap awal penelitian. Pada tahap ini dilakukannya identifikasi masalah, latar belakang, tujuan penelitian, pemilihan metode atau algoritma yang digunakan dalam menyelesaikan masalah yang telah diidentifikasi.

## Studi literatur

Studi literatur merupakan suatu proses pencarian teori-teori atau penelitian terhadap sebuah masalah. Literatur yang digunakan dapat berupa buku, jurnal, artikel, ataupun *paper.* Literatur penelitian ini berfokus pada topik software design, software maintainability, SOLID design principle. dan metode pengukuran maintainability menggunakan CK metrics.

## Pengembangan objek teliti

Pengembangan objek teliti merupakan tahapan dimana peniliti mengembangkan sebuah perangkat lunak media pembelajaran pemrograman. dalam pengembangan perangkat lunak yang akan dilakukan peneliti akan menggunakan SOLID design principle.

## Pengukuran maintainability

Pada tahap ini ketika perangkat lunak sudah jadi dilakukannya pengukuran dari perangkat lunak yang menerapkan SOLID design principle tersebut.

## Analisis hasil penelitian

Setelah dilakukannya pengukuran dengan menggunakan CK metrics maka akan menghasilkan beberapa *sub-metric*

1. WMC metric (weighted methods per class)
2. DIT metric (Depth of inheritance Tree)
3. NOC metric (Number of children)
4. CBO metric (Coupling between object)
5. RFC metric (Response for a class)
6. LCOM metric (lack of cohesion in methods).

## Jadwal dan Lokasi Penelitian

**Lokasi Penelitian**

Lokasi penelitian dapat dilakukan dimana saja dikarenakan penelitian tidak membutuhkan tempat khusus dalam pengambilan data maupun metode yang digunakan.

**Jadwal Penelitian**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **NO** | **KEGIATAN** | **MINGGU** | | | | | | | | | | | | **HASIL KESELURUHAN** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| 1 | Studi Literatur |  |  |  |  |  |  |  |  |  |  |  |  | Mengetahui bagaimana implementasi SOLID design principle dalam pengembangan sebuah perangkat lunak dan mengukurnya menggunakan C&K metrics |
| 2 | Pengembangan objek teliti |  |  |  |  |  |  |  |  |  |  |  |  | perangkat lunak yang sudah jadi |
| 3 | Pengukuran maintainability |  |  |  |  |  |  |  |  |  |  |  |  | Data hasil dari C&K metrics |
| 4 | Analisis hasil pengkuran |  |  |  |  |  |  |  |  |  |  |  |  | Data evaluasi dari hasil C&K metrics yang dihasilkan. |
| 5 | Penarikan Kesimpulan |  |  |  |  |  |  |  |  |  |  |  |  | Kesimpulan dari keseluruhan proses |

# DAFTAR PUSTAKA

[1] 42matters, “Updated apps in the last 135 days,” 2022. https://42matters.com/app-market-explorer/android/5b891991c8453b038008897b

[2] S. Hassan, C.-P. Bezemer, and A. E. Hassan, “Studying bad updates of top free-to-download apps in the google play store,” *IEEE Trans. Softw. Eng.*, vol. 46, no. 7, pp. 773–793, 2018.

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

[4] H. Zuse, *A framework of software measurement*. Walter de Gruyter, 2013.

[5] M. Jaiswal, “Software architecture and software design,” *Int. Res. J. Eng. Technol. e-ISSN*, pp. 56–2395, 2019.

[6] D. Garlan and M. Shaw, “An Introduction to Software Architecture,” *Sch. Comput. Sci. Carnegie Mellon Univ. June*, 2011.

[7] H. Singh and S. I. Hassan, “Effect of solid design principles on quality of software: An empirical assessment,” *Int. J. Sci. Eng. Res.*, vol. 6, no. 4, 2015.

[8] R. C. Martin, J. Grenning, S. Brown, K. Henney, and J. Gorman, *Clean architecture: a craftsman’s guide to software structure and design*, no. s 31. Prentice Hall, 2018.

[9] R. Prajna, “Adoption of Design Principles and Design Patterns for Developing Software Application,” *J. Emerg. Technol. Innov. Res.*, vol. 8, no. 5, 2021.

[10] S. Sikka, “Metrics To Quantify Solid Software,” *J. Emerg. Technol. Innov. Res.*, vol. 5, no. 6, pp. 9–10, 2018.

[11] A. Ampatzoglou *et al.*, “Applying the single responsibility principle in industry: modularity benefits and trade-offs,” in *Proceedings of the Evaluation and Assessment on Software Engineering*, 2019, pp. 347–352.

[12] A. Kurmangali, M. E. Rana, and W. N. W. Ab Rahman, “Impact of Abstract Factory and Decorator Design Patterns on Software Maintainability: Empirical Evaluation using CK Metrics,” in *2022 International Conference on Decision Aid Sciences and Applications (DASA)*, 2022, pp. 517–522.

[13] B. Tyagi and Y. Beg, “An Experimental Assessment on Effects of Solid Design Principles on the quality of Software using CKJM Metric Analysis”.