

PROPOSAL TUGAS AKHIR
ANALISIS *MAINTAINABILITY* PENGEMBANGAN APLIKASI KEPEGAWAIAN
PPNPN DI KEMENTRIAN AGAMA PUSAT DENGAN MENERAPKAN SOLID
***DESIGN PRINCIPLE* MENGGUNAKAN CK METRICS**



Disusun Oleh:

Mujahid Ansori Majid 1197050093

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SUNAN GUNUNG DJATI
BANDUNG
2023

LEMBAR PERSETUJUAN PROPOSAL TUGAS AKHIR

**Analisis *Maintainability* Pengembangan Aplikasi Kepegawaian
PPNPN di Kementerian Agama Pusat dengan Menerapkan
SOLID Design Principle Menggunakan *CK Metrics***

Disusun Oleh:

Mujahid Ansori Majid 1197050093

Proposal ini telah diperiksa dan disetujui oleh pembimbing untuk diseminarkan

Bandung, 10 Februari 2023

Mengetahui,

Ketua Jurusan Teknik Informatika

Calon Pembimbing



Cepi Slamet, Ph.D.

NIP. 198002252011011007



Prof. Dr. Muhammad Ali Ramdhani, S. TP., M.T

NIP. 197111062008011009

DAFTAR TABEL

Table 1 state of the art.....	9
-------------------------------	---

DAFTAR GAMBAR

Gambar 1 Negative Review for each updates.....	4
Gambar 2 updates to recover konsumen's satisfaction	5
Gambar 3 variabel yang digunakan	11
Gambar 4 Kerangka Pemikiran	12
Gambar 5 Metode Penelitian.....	13

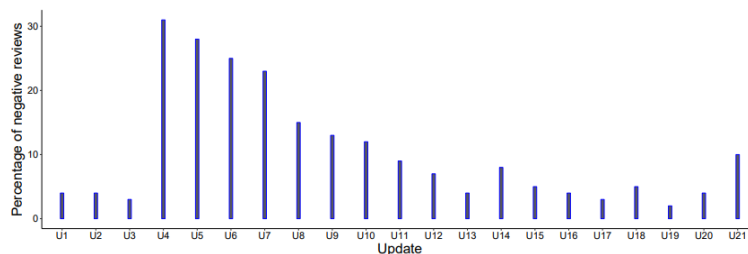
DAFTAR ISI

DAFTAR TABEL	2
DAFTAR GAMBAR.....	3
DAFTAR ISI.....	3
PENDAHULUAN	4
1. Latar Belakang.....	4
2. Rumusan Masalah.....	8
3. Batasan Masalah	8
4. Tujuan Penelitian	9
5. Manfaat Penelitian.....	9
6. <i>The State of The Art</i>	9
7. Kerangka Pemikiran	11
8. Metode Penelitian	12
1. Perumusan Masalah.....	13
2. Studi literatur	13
3. Pengembangan objek teliti	13
4. Pengukuran maintainability	14
5. Analisis hasil penelitian.....	14
6. Jadwal dan Lokasi Penelitian.....	14
DAFTAR PUSTAKA	16

PENDAHULUAN

1. 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 menunjukkan bahwa 859.771 dari 3.792.074 apps di playstore melakukan update aplikasi. Hal tersebut menunjukkan 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 menunjukkan 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 ditunjukkan pada grafik dibawah ini

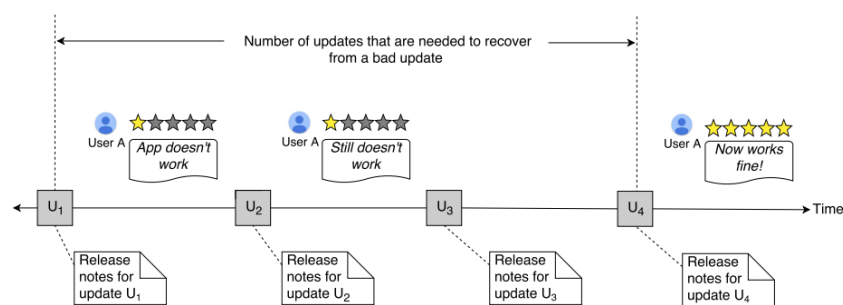


Gambar 1 Negative Review for each updates

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*. (Hassan, Bezemer, & Hassan, 2018)

Untuk mengembalikan kepercayaan *user* terhadap aplikasi yang telah melakukan *bad updates* yaitu tentu saja dengan melakukan *fixing* dari *bugs* yang

terdapat 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 **Gambar 2**



Gambar 2 updates to recover konsumen's satisfaction

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 untuk sering melakukan *updates* dibutuhkannya sebuah desain 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 mengembalikannya. Juga menjadikan aplikasi lebih baik setelahnya (Grassi & Mirandola, 2021)

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 mengungkapkan bahwa 70% - 80% dari perangkat lunak yang ada diproduksi dengan *structured programming*. Karena sulit menentukan bahwa perubahan kode akan mempengaruhi sesuatu. Juga masalah terbesar 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 perangkat lunak yang bagus sangatlah penting untuk kesuksesan pengembangan perangkat lunak (Zuse, 2013)

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 (Jaiswal, 2019).

Semakin besarnya sebuah aplikasi dan banyaknya permintaan penambahan fitur dari *stackholders*, algoritma dan struktur data bukan lagi menjadi masalah yang besar perihal desain aplikasi. Hal ini dikarenakan meskipun sebuah aplikasi mempunyai algoritma yang baik namun desain yang kurang baik, ketika sebuah sistem dibangun dengan skala besar dan terus mengalami peningkatan maka akan menghasilkan sekumpulan masalah desain 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 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 kecenderungan 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 perangkat lunak. Sistem perangkat lunak harus dimulai dengan *clean code*. Jika diibaratkan dengan sebuah bangunan jika batanya tidak bagus, maka *blueprint* atau arsitektur perangkat lunak yang baik pun akan menghasilkan sebuah bangunan yang tidak baik. Begitupun sebuah bangunan jika memiliki memiliki *building blocks* yang baik namun memiliki desain yang tidak baik berpotensi akan menghasilkan bangunan 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 *abstractions* tidak boleh bergantung kepada implementasinya. Dan implementasinya harus bergantung kepada *abstractions*. (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 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 desain aplikasi yang dapat memenuhi kebutuhan tersebut.

2. Rumusan Masalah

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

- Bagaimana mengukur *maintainability* sebuah perangkat lunak berorientasi objek dengan menggunakan *C&K metrics*
- Bagaimana kemudahan sebuah perangkat lunak dalam melakukan pembaruan jika menerapkan *SOLID design principle*

3. Batasan Masalah

Batasan masalah dari penelitian ini adalah sebagai berikut:

- Bahan uji dalam penelitian ini hanya perangkat lunak kepegawaian PPNPN
- Metode untuk menguji *maintainability* pada penelitian ini yaitu *C&K metrics*

4. Tujuan Penelitian

Tujuan dari penelitian ini sebagai berikut:

- Mengimplementasikan *C&K metrics* dalam mengukur *maintainability* perangkat lunak.
- Menganalisis kemudahan sebuah aplikasi dengan menerapkan SOLID design principle dalam melakukan pembaruan.

5. Manfaat Penelitian

Manfaat yang diharapkan dari hasil penelitian ini adalah sebagai berikut:

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

6. *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. Berikut jurnal-jurnal tersebut:

Table 1 state of the art

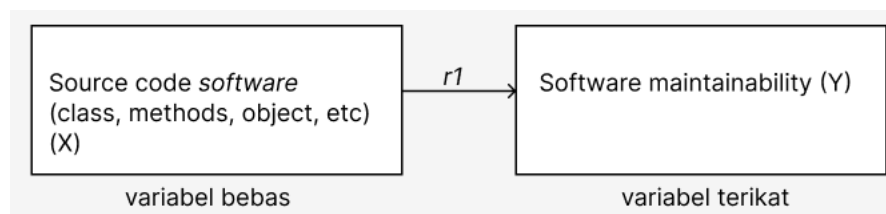
No	Judul Jurnal dan Peneliti	Tahun dan tempat penelitian	Metode Penelitian	Objek penelitian	Perbandingan yang dijadikan alasan tujuan penelitian
1	<i>Adoption of Design principles and Design patterns for Developing Software Application</i> Peneliti: R Prajana dan Prof.Kavitha S N	2021, India	Kuantitatif	Inpatient pharmacy medication management.	Hasil dari penelitian ini berupa effect dari penggunaan prinsip desain SOLID, hasilnya menunjukkan bahwa <i>abstractness metric</i> (metric yang digunakan) menunjukkan hasil yang baik untuk software <i>maintainability</i> .
2	METRICS TO QUANTIFY	2018, India	Kuantitatif	Penelitian ini menggunakan	Hasil dari penelitian ini

No	Judul Jurnal dan Peneliti	Tahun dan tempat penelitian	Metode Penelitian	Objek penelitian	Perbandingan yang dijadikan alasan tujuan penelitian
	<p>SOLID SOFTWARE DESIGN PRINCIPLES</p> <p>Peneliti: Dr. Sunil Sikka</p>			<p>hasil dari (Singh & Hassan, 2015) dan Jason Garmon</p>	<p>memvalidasi hipotesis peneliti bahwa SOLID <i>design principle</i> merupakan sebuah design principle yang dapat membuat desain object oriented semakin dan membuat kualitas perangkat lunak semakin baik pula.</p>
3.	<p>Impact of design patterns on software quality: a systematic literature review</p> <p>Penulis: Fadi Wedyan dan Somia Abufakher</p>	2020, USA	Kuantitatif	<p>50 penelitian utama dari rentang waktu tahun 2000 - 2018</p>	<p>Penelitian ini mengamati penelitian dalam periode 2000 – 2018 mengenai design patterns. Penulis menyebutkan bahwa design pattern terbukti secara empiris mempunyai positive efek dalam maintainability</p>
4.	<p>Impact of Abstract Factory and Decorator Design Patterns on Software Maintainability: Empirical Evaluation using CK Metrics</p> <p>Peneliti:</p>	2022, Malaysia	Kuantitatif	<p>Sample problem scenario of car manufacturing</p>	<p>Penelitian ini membahas mengenai dua buah design principle yaitu abstract factory dan decorator dengan pengukuran menggunakan</p>

No	Judul Jurnal dan Peneliti	Tahun dan tempat penelitian	Metode Penelitian	Objek penelitian	Perbandingan yang dijadikan alasan tujuan penelitian
	Aisha Kurmangali, Muhammad Ehsan Rana, dan Wan Nurhayati Wan Ab Rahman				CKJM menghasilkan sebuah analisis yang menunjukkan design patterns secara positif mempengaruhi software maintainability

7. Kerangka Pemikiran

Variabel dependent atau variabel terikat dalam penelitian ini adalah software maintainability atau sebagaimana fleksibel sebuah software mudah untuk dirawat. Untuk variable independent atau variabel bebas dari penelitian ini adalah source code dari software yang dibuat.

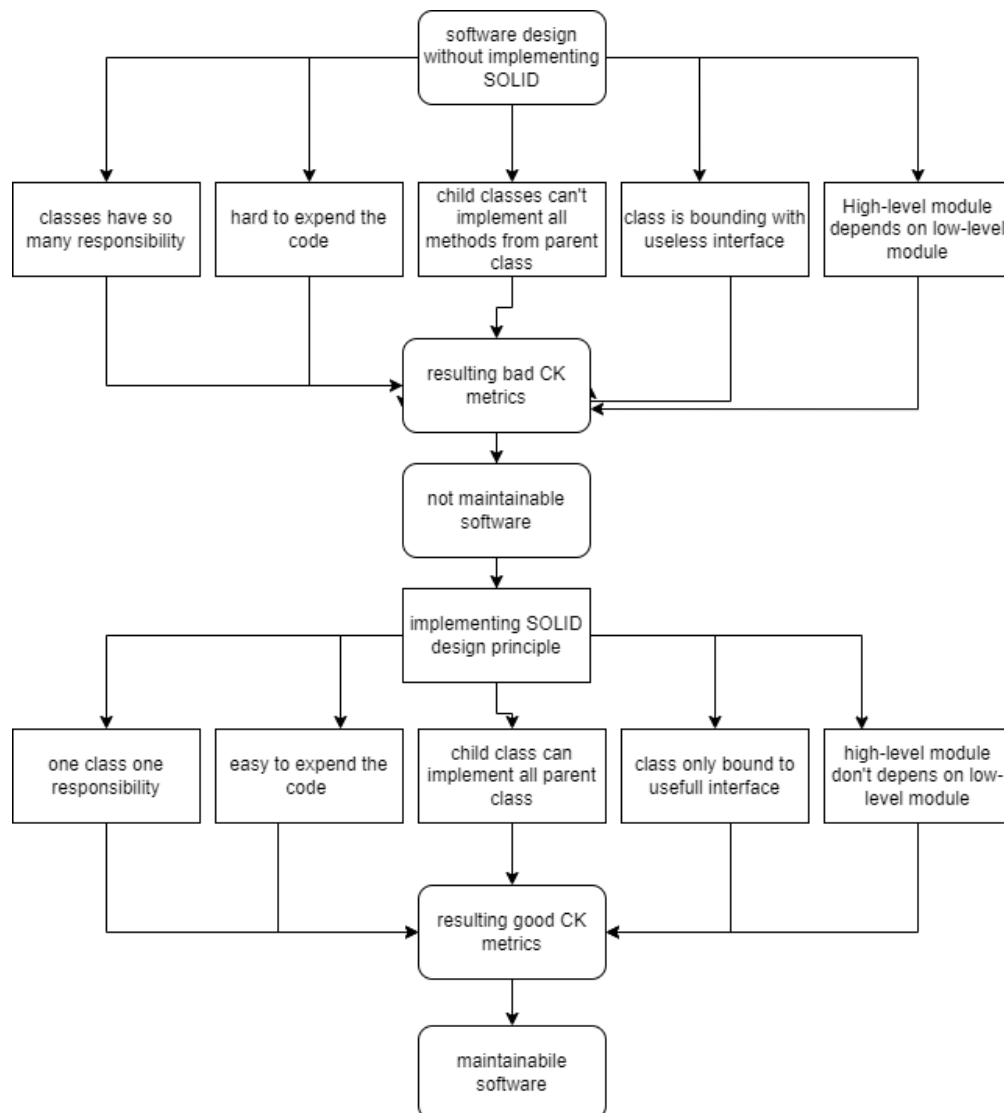


Gambar 3 variabel yang digunakan

Pengembangan *software* bukan hanya sekali *deploy* langsung selesai namun perlunya dilakukan perawatan. Aplikasi yang dapat dikembangkan secara terus menerus diperlukan software design yang matang, software design yang buruk menyebabkan aplikasi akan sulit dikembangkan atau lebih parah lagi tidak dapat dikembangkan lagi. Oleh karena itu diperlukannya sebuah arahan atau *guidelines* untuk menciptakan software design yang baik.

Oleh karena itu dalam tahap pengembangan awal sebuah software yang *maintainable* harus menerapkan suatu design principle. SOLID design principle merupakan salah satu design principle yang dapat membuat software design menjadi lebih baik dan juga membuat

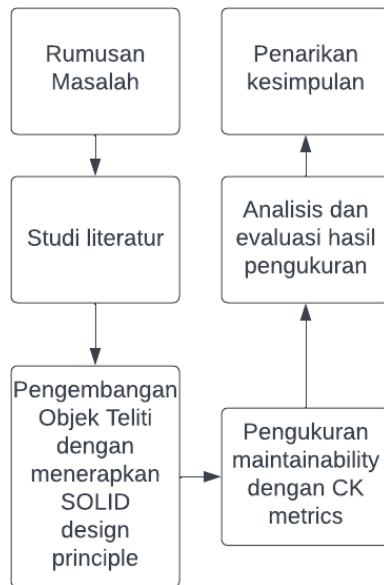
software lebih maintainable. Berdasarkan uraian di atas, maka kerangka berpikir dalam penelitian ini dapat dipetakan sebagai berikut:



Gambar 4 Kerangka Pemikiran

8. 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

1. 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.

2. 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.

3. Pengembangan objek teliti

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

4. Pengukuran maintainability

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

5. Analisis hasil penelitian

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

- a. WMC metric (weighted methods per class)
- b. DIT metric (Depth of inheritance Tree)
- c. NOC metric (Number of children)
- d. CBO metric (Coupling between object)
- e. RFC metric (Response for a class)
- f. LCOM metric (lack of cohesion in methods).

6. 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 pengukuran													Data evaluasi dari hasil C&K metrics yang dihasilkan.
5	Penarikan Kesimpulan													Kesimpulan dari keseluruhan proses

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.
- Kayarvizhy, N., Kanmani, S., & Uthariaraj, V. R. (2016). Enhancing the fault prediction accuracy of CK metrics using high precision cohesion metric. *International Journal of Computer Applications in Technology*, 54(4), 290–296.
- 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.

Subramanyam, R., & Krishnan, M. S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29(4), 297–310.

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