PROPOSAL TUGAS AKHIR

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

Disusun Oleh:

Mujahid Ansori Majid 1197050093

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS ISLAM NEGERI SUNAN GUNUNG DJATI

BANDUNG

2023

DAFTAR TABEL

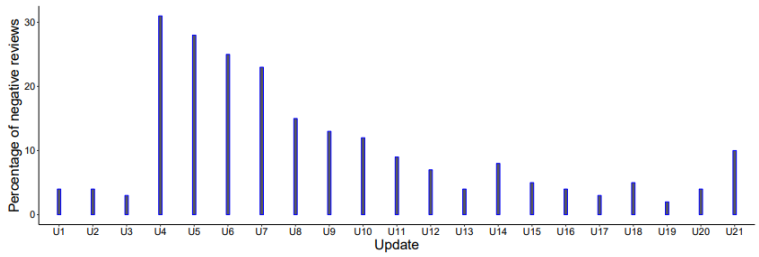
DAFTAR GAMBAR

DAFTAR ISI

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 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 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

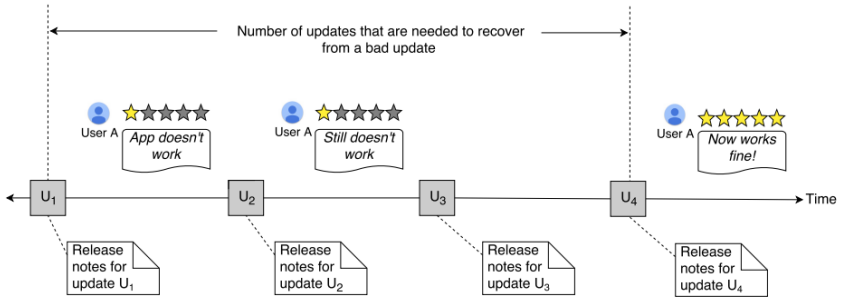


Gambar 1 Negative Review for each updates

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*. (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 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 consumen'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 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 (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 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 (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 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 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 (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)

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 fleksibilitas sebuah perangkat lunak jika menerapkan SOLID *design principle*

3. Batasan Masalah

Batasan masalah dari penelitian ini adalah sebagai berikut:

- Bahan uji dalam penelitian ini hanya perangkat lunak media pembelajaran pemrograman
- Metode untuk menguji *maintainability* pada penilitian ini yaitu *C&K metrics*

4. Tujuan Penelitian

Tujuan dari penelitian ini sebagai berikut:

- Mengimplementasikan *C&K metrics* dalam mengukur *maintainability* perangkat lunak
- Menganalisis efektifitas penggunaan metode *C&K metrics* dalam mengukur *maintainability* sebuah perangkat lunak

5. Manfaat Penelitian

Manfaat yang diharapkan dari hasil penelitian ini adalah sebagai berikut:

- Mengetahui fleksibilitas dari perangkat lunak untuk pemeliharaan
- Mengetahui efektifitas penggunaan metode *C&K metrics* dalam mengukur fleksibilitas sebuah perangkat lunak

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. Dalam penelitian ini

menyertakan 5 jurnal international yang berhubungan mengenai *software maintainability*. Berikut jurnal-jurnal tersebut:

1) 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)

2) 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)

3) Penelitian yang berjudul "*Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects"* yang diperoleh dari *IEEE computer society* dilakukan oleh Ramanath Subramanyam and M.S. Krishnan pada tahun 2003 menjelaskan betapa pentingnya aspek desain dalam aplikasi yang beroritentasi objek khususya dalam fase awal pengembangan. Metrics desain berperan penting penting dalam membantu developer memahami aspek desain dari sebuah software dan karena memahami desian metrics dapat meningkatkan kualitas software dan

produktifitas dalam pengemabangan. Dalam penelitian ini menyediakan bukti empiris mengenai peran OO desain complexity metrics khususnya CK metrics dalam menentukan kecacatan sebuah software. Penelitian ini hanya menggunakan tiga dari enam metrics, hal tersebut dikarenakan ketiga yang dikecualikan tidak cocok dengan model yang akan diuji dan berpotensi akan menyulitkan dalam perhitungan. Fokus utama dari penelitian ini yaitu memahami peran beberapa pengukuran dalam CK metrics dalam menjelaskan kecacatan aplikasi berorientasi objek di level *class*. Hasilnya menunjukan bahwa beberapa CK metrics yang mengukur desain complexity aplikasi berbasis OO secara signifikan menjelaskan kecacatan dalam sebuah aplikasi yang diuji (Subramanyam & Krishnan, 2003).

4) Penelitian yang berjudul "Enhancing the fault prediction accuracy of CK metrics using high precision cohesion metrics" yang diterbitkan oleh *Int. J. Computer Applications in Technology*

   Yang ditulis oleh N. Kayarvizhy, S. Kanmani, dan V.Rhymend Uthariaraj pada tahun 2016 penelitian ini membahas mengenai salah satu object-oriented metrics yaitu The Chidamber and Kemerer (CK) metric, karena CK metric dianggap sebagai pioneer yang sudah lama digunakan dan dijadikan sebagai benchmarking untuk metric-metric yang baru. Penelitian ini mengevaluasi kekurangan kemampuan CK metric dalam memprediksi dan mengvalidasinya secara empiris. Secara spesifik penelitian ini mengganti cohesion metric (LCOM) dengan cohesion metric (high precision cohesion metrics) (Kayarvizhy, Kanmani, & Uthariaraj, 2016).

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	*Empirical Analysis of CK*	2003, India	Kuantitatif	*large B2C e-commerce*	Hasil dari penilitian ini

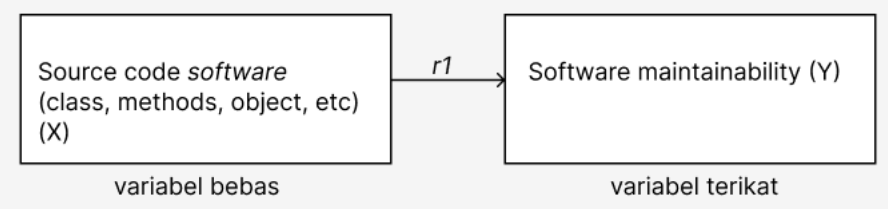
	Metrics for Object-Oriented Design Complexity: Implications for Software Defects Peneliti: Ramanath Subramanyam dan M.S. Krishnan			*application suite developed using C++ and Java as primary implementation languages.*	digunakan sebagai pedoman peneliti bahwasanya menggunakan beberapa CK metrics dapat mengidentifikasi beberapa kecacatan di dalam aplikasi sehingga dapat dijadikan sebagai
2	METRICS TO QUANTIFY SOLID SOFTWARE DESIGN PRINCIPLES Peneliti: Dr. Sunil Sikka	2018, India	Kuantitatif	Penelitian ini menggunakan hasil dari (Singh & Hassan, 2015) dan Jason Garmon	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.	A COMPREHENSIVE REVIEW AND ANALYSIS ON OBJECT-ORIENTED SOFTWARE METRICS IN SOFTWARE MEASUREMENT Penulis: K.P. Srinivasan Dr. T. Devi	2014, India	Kuantitatif	Software metrics	Penelitian ini menyatakan bahwa kebanyakan metrics yang ada kurang adanya eksperimen dan hanya sedikit saja yang diterima dan digunakan. Sehingga penilitian ini membandingkan software metrics yang ada, terdapat lima metrics yang diuji. Hasilnya menyatakan bahwa CK metrics salah satu metrics yang paling banyak digunakan.

					Meskipun salah satu metrics tertua yang ada CK metrics merupakan salah satu metrics yang masih dapat diandalkan.
4.	Enhancing the fault prediction accuracy of CK metrics using high precision cohesion metric Peneliti: N. Kayarvizhy, S. Kanmani, dan V.Rhymend Uthariaraj	2016, India	Kuantitaif	500 *classes* dari 12 project berbeda	Penelitian ini merupakan *tren* terbaru dari The chidamber and kemere metric. Dengan adanya perubahan peneliti akan menggunakan metode yang terbaru sehingga prediksi yang dihasilkan akan semakin baik.

7. Kerangka Pemikiran

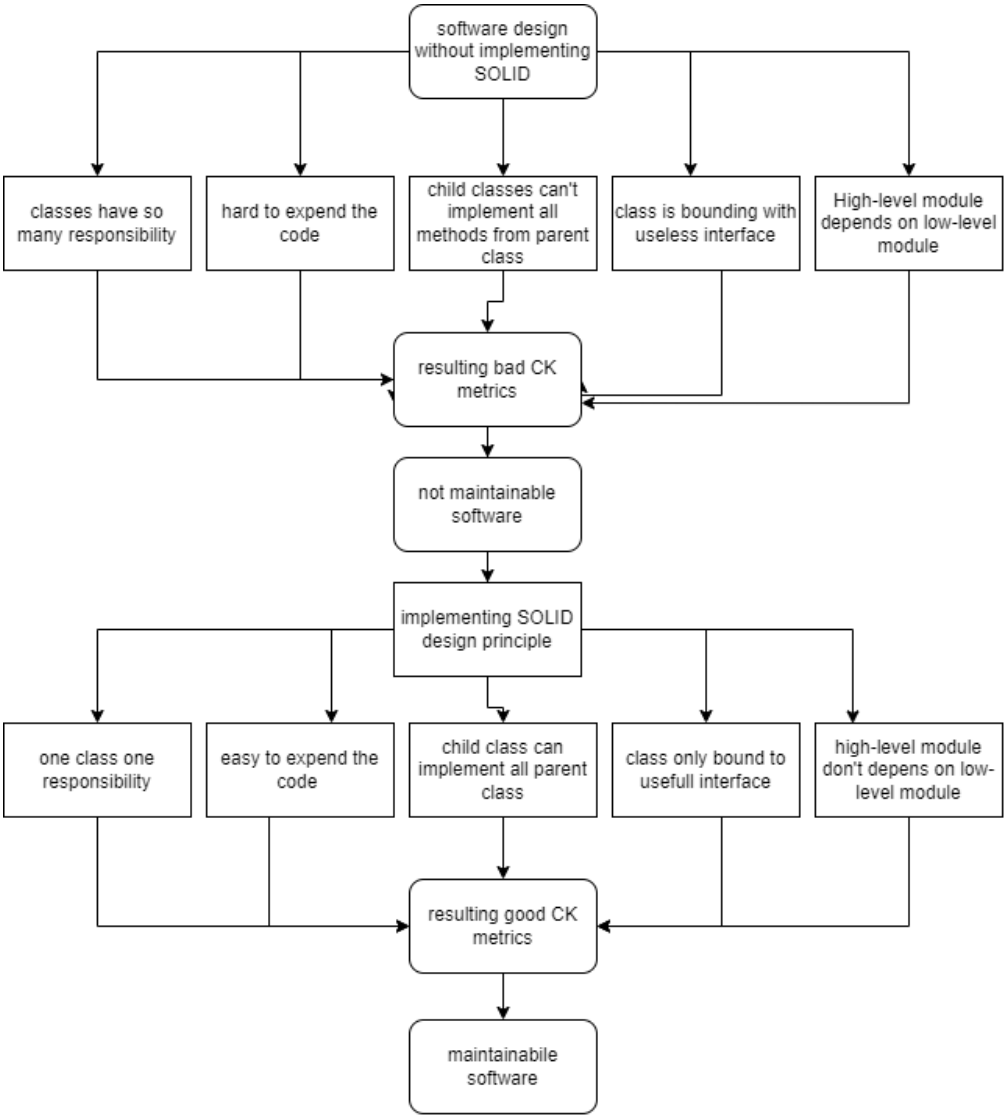
Variabel dependent atau variabel terikat dalam penelitian ini adalah software maintainability atau sebagaimana fleksibel sebuah software mudah untuk dirawata. Untuk variable independent atau variabel bebas dari penelitian ini adalah source code dari software yang dibuat.



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.

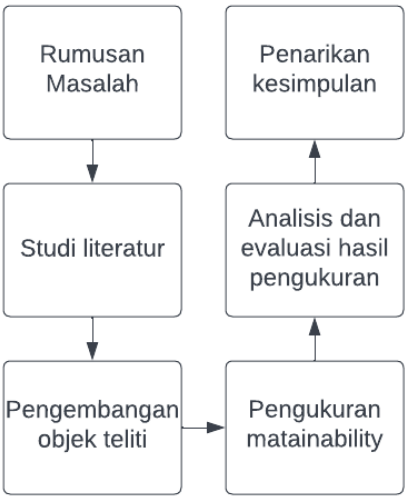
Oleh karena itu dalam tahap pengembangan awal sebuah software yang *maintainable* harus menerapkan suatu design principle. SOLID design principle merupakan salah satu design principle yang dapat membuat software design menjadi lebih baik dan juga membuat software lebih maintainable. Berdasarkan uraian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