JURNAL SAINS DAN INFORMATIKA
RESEARCH OF SCIENCE AND INFORMATIC V8.I1

Vol.8 No.01(2022)68-74 p-issn : 2459-9549
http://publikasi.lldikti10.id/index.php/jsi e-issn : 2502-096X

Pengembangan Aplikasi SOLID-*Calculator* untuk Pengukuran Kualitas Desain Diagram Kelas

Intan Oktafiani[a], Muhammad Fauzi Arda Saputra[b]
[a]Program Studi Ilmu Komputer, Fakultas Sain dan Ilmu Komputer, Universitas Pertamina, intan.oktafiani@universitaspertamina.ac.id
[b]Program Studi Ilmu Komputer, Fakultas Sain dan Ilmu Komputer, Universitas Pertamina, fauziardasaputra@gmail.com

Submitted: 25-04-2022, Reviewed: 25-04-2022, Accepted 30-04-2022
http://doi.org/10.22216/jsi.v8i1.959

Abstract

Today's development of information technology makes the competition for software products increasingly tight. Developers should release immediately and gain the trust of users. One crucial strategy that needs to be done is improving the software's quality. The application of software engineering principles in every development process is how to deliver quality software. The SOLID principles are the standard for developers to build well-designed software. Using this principle will make the software easier to understand, more flexible, easy to maintain, and testable. Implementation of the SOLID Principles can be started from the design of the class diagram. In this research, a software development called SOLID-Calculator was carried out to speed up measuring the quality of class diagrams. When developers can quickly find out the level of design quality, the faster errors can be detected and corrected. It is believed that the product will soon enter the market share and even excel in user ratings.

Keywords: Software, Quality, Design, Class Diagram, SOLID Principles

Abstrak

Perkembangan teknologi informasi saat ini membuat persaingan produk perangkat lunak menjadi semakin ketat. Pengembang harus segera merilis dan memperoleh kepercayaan pengguna. Salah satu strategi penting yang perlu dilakukan adalah dengan meningkatkan kualitas perangkat lunak. Untuk dapat menghasilkan perangkat lunak yang berkualitas diperlukan penerapan kaidah-kaidah rekayasa perangkat lunak dalam setiap proses pengembangannya. Prinsip SOLID merupakan merupakan standar untuk pengembang agar dapat membangun perangkat lunak dengan desain yang baik. Dengan menerapkan prinsip ini, perangkat lunak akan lebih mudah dipahami, lebih fleksibel, mudah dipelihara dan dapat diuji. Implementasi Prinsip SOLID dapat dimulai dari desain diagram kelas. Pada penelitian ini dilakukan pengembangan perangkat lunak dengan nama SOLID-Calculator untuk mempercepat proses pengukuran kualitas diagram kelas. Ketika pengembang dapat dengan cepat mengetahui tingkat kualitas desainnya maka akan semakin cepat kesalahan dapat dideteksi dan diperbaiki. Dengan itu, produk tersebut diyakini akan dengan cepat masuk dalam pangsa pasar bahkan dapat unggul dalam peringkat penggunaan.

Kata kunci: Perangkat lunak, kualitas, desain, diagram kelas, prinsip SOLID.

1. Pendahuluan

Kecepatan kebutuhan teknologi informasi dan jumlah pesaing yang semakin meningkat, menuntut para pengembang untuk dapat merilis produk ke pasar dengan cepat dan menghasilkan perangkat lunak yang dapat menarik perhatian pengguna untuk menjadi pengguna tetapnya. Untuk menjawab tantangan tersebut perlu dilakukan pengelolaan kualitas perangkat lunak. Pada dasarnya tingkat kualitas suatu artefak ini hanya dapat diketahui dengan melakukan pengukuran. Pengukuran harus dilakukan pada setiap proses pengembangan perangkat lunak, tetapi pada kenyataannya pengukuran tersebut cenderung hanya dilakukan pada tahap pengujian kode implementasi, dimana biaya perbaikan kesalahan pada tahap pengujian adalah 10 kali lipat lebih besar dibandingkan apabila kesalahan tersebut diperbaiki pada tahap desain

[1]. Terlebih lagi apabila produk telah dirilis, biaya perbaikan dapat mencapai hingga 30 kali lipat lebih besar. Hal ini terjadi karena kesalahan tersebut terjadi berlarut-larut dalam setiap tahap pengembangan. Besar biaya ini berbanding lurus dengan waktu yang dibutuhkan. Untuk melakukan perbaikan, kesalahan harus dicari hingga kesumbernya hal ini pastinya juga menyita waktu dan biaya. Bukan hanya itu, merilis perangkat lunak yang mengandung bugs atau kesalahan berpotensi kerusakan reputasi dan menghilangkan kepercayaan pelanggan. Hal ini menyebabkan jumlah pengguna dapat yang juga diiringi penurunan jumlah pendapatan.

Diagram kelas merupakan salah satu artefak desain yang harus mempunyai kualitas yang baik karena diagram ini menjadi pedoman bagi para pemrogram untuk membuat kelas, atribut, fungsi, serta hubungan antar kelasnya. Prinsip SOLID [2] adalah salah satu pedoman desain perangkat lunak berorientasi objek yang terdiri dari 5 prinsip: *Single Responsibility Principle (SRP), Open Closed Principle (OCP), Liskov Substitution Principle, (LSP), Interface Segregation Principle (ISP)* dan *Dependency Inversion Principle (DIP)*. Penerapan prinsip ini mampu menghasilkan perangkat lunak yang lebih mudah dipahami, lebih fleksibel, mudah dipelihara dan dapat diuji.

Metriks untuk mengukur kesesuaian desain diagram kelas terhadap Prinsip SOLID telah dirumuskan oleh Intan Oktafiani [3], namun pengukuran masih harus dilakukan secara manual. Setiap komponen diagram kelas harus dianalisis, kemudian dihitung satu persatu. Hal ini pastinya memakan waktu dan kemungkinan besar dapat terjadi kesalahan apabila persepsi pengukur berbeda dengan persepsi metrik. Aplikasi *SOLID-Calculator* ini dibuat untuk mengautomasi proses pengukuran tersebut. Desain diagram kelas hanya perlu disimpan dalam file berformat .mdj hasil keluaran aplikasi StarUML, untuk kemudian aplikasi melakukan parsing, perhitungan, kemudian menampilkan laporan mengenai nilai kepatuhan desain tersebut beserta letak kesalahannya. Dengan harapan dapat membantu para pengembang untuk mengetahui nilai kepatuhan, letak kesalahan, kemudian dapat menjadi pertimbangkan para pengembang, perbaikan apa saja yang perlu dilakukan untuk menghasilkan desain yang kualitas.

2. Tinjauan Pustaka/Penelitian Sebelumnya

Berikut ini adalah tinjauan pustaka mengenai beberapa hal penting berkaitan dengan penelitian ini yaitu tentang Prinsip SOLID beserta metriksnya, teknik parsing, diagram kelas dan penelitian serupa.

2.1 Prinsip SOLID

Prinsip desain SOLID [2] adalah prinsip desain berorientasi objek yang memungkinkan pengelolaan sebagian besar masalah terkait kualitas desain perangkat lunak. Sekumpulan prinsip ini dapat memberikan pemahaman tentang desain untuk menghindari gejala desain yang buruk atau yang dikenal dengan "*Bad Design*", membantu untuk mengurangi kompleksitas kode, meningkatkan keterbacaan, ekstensibilitas, kemudahan pemeliharaan, mengurangi kesalahan, dapat digunakan kembali, mudah untuk diuji, dan mengurangi ketergantungan yang ketat pada perangkat lunak berorientasi objek. Jika tidak mengikuti Prinsip SOLID, perangkat lunak yang akan dihasilkan akan cenderung memiliki tingkat ketergantungan kode yang erat dengan banyak modul lain. Ketergantungan yang erat menghasilkan kode yang sulit untuk diuji, terdapat banyak duplikasi kode, munculnya *bug* baru ketika memperbaiki bug lain, dan banyak potensi masalah dengan berbagai sebab dalam siklus pengembangan aplikasi.

SOLID merupakan singkatan dari lima prinsipnya. Penjelasan singkat mengenai prinsip-prinsip dan metriksnya sebagai berikut:

1. SRP – *The Single Responsibilities Principle*

"*A class should have only one reason to change*." Prinsip ini mengenai kohesi suatu kelas, komponen kelas harus saling berkaitan. Adapun pengukuran untuk prinsip ini dapat dilakukan mengikuti Persamaan berikut :

$$VSRP = \frac{\sum CSRP}{Jumlah\ Kelas\ (NC)} \quad (1)$$

VSRP : *Value of SRP*, yaitu nilai pengukuran SRP dengan rentang nilai diantara 0 sampai 1.
nCSRP : *n Conform to SRP*, yaitu banyaknya kelas yang memenuni prinsip SRP.
NC : *Number of Classes*, banyak kelas kelas dalam sebuah digram kelas yang diukur

2. OCP – *The Open Close Principle*

"*A module should be open for extension but closed for modification*.". prinsip ini berkaitan dengan pewarisan (*inheritance*) dan abstraksi (*abstraction*). Berikut adalah metriks pengukurannya:

$$VOCP = \frac{\sum COCP}{Jumlah\ Baseclass(NSUP)} \quad (2)$$

VOCP : *Value of OCP*, nilai pengukuran OCP dengan rentang nilai dari 0 sampai 1.
nCOCP : *n Conform to OCP*, banyaknya kelas dalam diagaram kelas yang memenuhi prinsip OCP.
NSUP : *Number of Superclass*, banyaknya kelas parent dalam diagram kelas yang diukur.

3. LSP – *The Liskov Subtitution Principle*

"*Subclasses should be substitutable for their base classes*". Kunci utama prinsip ini adalah pengelolaan hirarki pewarisan. Metriksnya sebagai berikut:

$$VLSP = \frac{\sum CLSP}{NOH} \quad (3)$$

VLSP : *Value of LSP*, nilai pengukuran LSP dengan rentang nilai dari 0 sampai 1.
nCLSP : *n Conform to LSP*, banyaknya kelas yang memenuhi prinsip LSP.
NOH : *Number of Hirearchy*, banyaknya set hirarki inheritance pada diagram kelas.

4. ISP – *The Interface Segregation Principle*

"*Many client specific interfaces are better than one general purpose interface*". Prinsip ini menekankan bahwa sebuah kelas seharusnya tidak bergantung kepada interface atau abstraksi yang tidak ada hubungannya dengan kelas tersebut. Adapun metriksnya dapat dilihat dibawah ini:

$$VISP = \frac{\sum CISP}{NOI} \quad (4)$$

VISP : *Value of ISP*, yaitu nilai pengukuran ISP dengan rentang nilai diantara 0 sampai 1.
nCISP : *n Conform to ISP*, yaitu banyaknya kelas yang memenuni prinsip ISP.
NOI : *Number of Interface*, banyak kelas interface dalam sebuah digram kelas yang diukur.

5. DIP – *The Dependency Inversion Principle*

"*Depend upon Abstractions. Do not depend upon concretions*". Prinsip ini menegaskan bahwa suatu kelas hanya boleh bergantung pada kelas abstrak atau interface. Berikut ini metriksnya:

$$VDIP = \frac{\sum CDIP}{NDep} \quad (5)$$

VDIP : *Value of DIP*, yaitu nilai pengukuran DIP dengan rentang nilai diantara 0 sampai 1.
nDIP : *n Conform to DIP*, yaitu banyaknya kelas yang memenuni prinsip DIP.
NDep : *Number of Dependency*, banyaknya hubungan dependency atau ketergantungan dalam sebuah digram kelas yang diukur.

2.2 Parsing

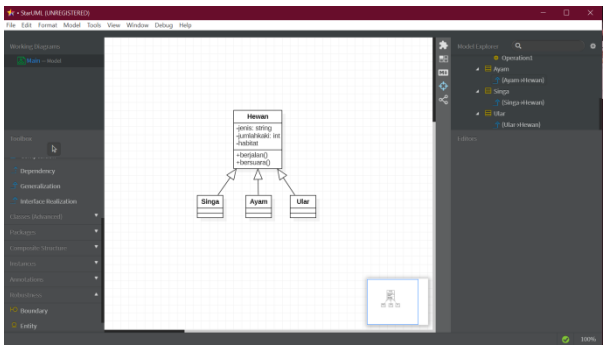
Parsing merupakan proses menerima masukan berupa kalimat (*string*), kemudian melakukan pembacaan dari kiri ke kanan dan dari atas kebawah, guna membentuk suatu susunan data yang terstruktur dari string tersebut sehingga mudah untuk digunakan kembali [4]. Pada kasus pengembangan aplikasi pengukuran prinsip SOLID ini, parsing dilakukan untuk membangun sebuah data yang terstruktur yang digunakan dalam perhitungan pengukuran prinsip SOLID. Parsing dilakukan dengan menerima masukan berupa text dari diagram kelas hasil keluaran aplikasi *StarUML* yang memiliki format data ".mdj". Adapun hasil yang diharapkan dari proses parsing ini adalah sekumpulan data UML Class yang akan digunakan dalam pengukuran.

2.3 Class Diagram

Diagram kelas merupakan suatu model desain yang dapat membantu dalam memvisualisasikan struktur kelas dari suatu sistem dan merupakan tipe diagram yang paling sering ditemui dalam pemodelan sistem berbasis objek. Diagram Kelas adalah salah satu dari *Unified Modeling Language* (UML) yang merupakan diagram statis yang menggambarkan struktur suatu desain dari suatu sistem dengan menunjukkan kelas, atribut, method dan hubungan antar objek [5]. Selain itu diagram kelas juga dapat mendeskripsikan, mendokumentasikan aspek sistem, serta membangun kode yang dapat dieksekusi dari aplikasi perangkat lunak. Kelas adalah unit dasar sistem berorientasi objek, konsep-konsep yang sudah dikenal adalah mengenai abstraksi, warisan, agregasi, ketergantungan dan Polimorfisme.

2.4 Star UML

StarUML merupakan perangkat lunak sumber terbuka (*open-source*) untuk merancang UML secara cepat, flexibel, dan fitur yang beragam, serta dapat diakses secara bebas diberbagai platform seperti Windows, MacOS, dan Linux (Staruml, 2020). Proyek yang dihasilkan dari perkakas ini disimpan sebagai satu file (.mdj). Pemodelan sistem perangkat lunak yang dapat dilakukan meliputi Use-Case Model, Design Model, Component Model, Deployment Model, atau lainnya [6]. Tampilan Aplikasi StarUML dapat dilihat pada Gambar 1.



Gambar 1 Tampilan Aplikasi StarUML

Berikut ini pada Gambar 2 adalah contoh isi file .mdj dari desain diagram kelas diatas.

```
{
    "_type": "UMLClass",
    "_id": "AAAAAAGAQLcOCVEzTEc=",
    "_parent": {
        "$ref": "AAAAAAFF+qBWK6M3Z8Y="
    },
    "name": "Hewan",
    "attributes": [
        {
            "_type": "UMLAttribute",
            "_id": "AAAAAAGAQLdUNVFdT8M=",
            "_parent": {
                "$ref": "AAAAAAGAQLcOCVEzTEc="
            },
            "name": "jenis",
            "visibility": "private",
            "type": "string"
        },
        {
            "_type": "UMLAttribute",
            "_id": "AAAAAAGAQLiZ+lFkQ/U=",
            "_parent": {
                "$ref": "AAAAAAGAQLcOCVEzTEc="
            },
            "name": "jumlahkaki",
            "visibility": "private",
            "type": "int"
        },
```


Gambar 2 Contoh isi file .mdj

2.5 Flutter

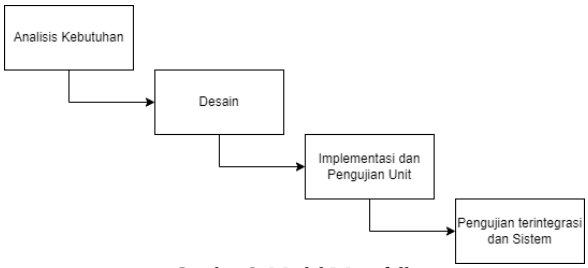
Flutter merupakan kerangka kerja antarmuka (*UI-framework*) pengembangan aplikasi antar-platform yang dikembangkan oleh *Google* [7]. *Flutter* digunakan bersamaan dengan bahasa pemograman utama yang digunakan adalah bahasa pemrograman *Dart* yang juga dikembangkan oleh *Google*. *Flutter* digunakan untuk mengembangkan aplikasi ini dikarenakan Flutter mampu membantu pengembangan aplikasi ini tanpa harus menggunakan tools tambahan lainnya, sehingga dapat mempermudah dan mempercepat proses pengembangan aplikasi.

2.6 Penelitian Serupa

Penelitian serupa sebelumnya telah dilakukan oleh Muhammad Dzaky [8] dimana hasil dari penelitian tersebut berupa aplikasi berbasis web yang dapat menghitung kualitas desain diagram kelas dengan cara memasukkan data komponen-komponen diagram kelas satu persatu ke dalam form yang disediakan. Dengan proses tersebut, waktu dan usaha yang dibutuhkan masih cukup besar. Perancang tidak dapat langsung mengunggah hasil desainnya tetapi harus secara manual memasukkan data satu persatu. Hal inilah yang ingin diperbaiki pada penelitian ini.

3. Metodologi Penelitian

Metodologi penelitian yang dilakukan mengikuti model proses pengembangan perangkat lunak *Waterfall* yang dapat dilihat pada Gambar 3.



Gambar 3. Model Waterfall

Model *waterfall* merupakan model pengembangan perangkat lunak yang menekankan tahapan-tahapan yang berurutan dan sistematis. Model ini dipilih karena daftar kebutuhan dari aplikasi yang akan dibangun sudah pasti, tidak rentan berubah-ubah. Tahapan pada proses model ini dimodifikasi dari model yang ada pada buku *Software Engineering* [9] oleh Ian Sommerville. Modifikasi yang dilakukan adalah penghapusan tahap akhir yaitu operasi dan pemeliharaan karena perangkat lunak ini belum sampai pada tahap tersebut.

3.1 Analisis Kebutuhan

Pada tahap ini, dilakukan wawancara sebagai bentuk studi kelayakan pengembangan aplikasi pengukuran kelas diagram terhadap Prinsip SOLID ini, studi literatur, serta observasi terhadap aplikasi sejenis. Dari hasil pengumpulan kebutuhan tersebut, kemudian dianalis sedemikian rupa hingga dihasilkan daftar kebutuhan sebagai berikut:

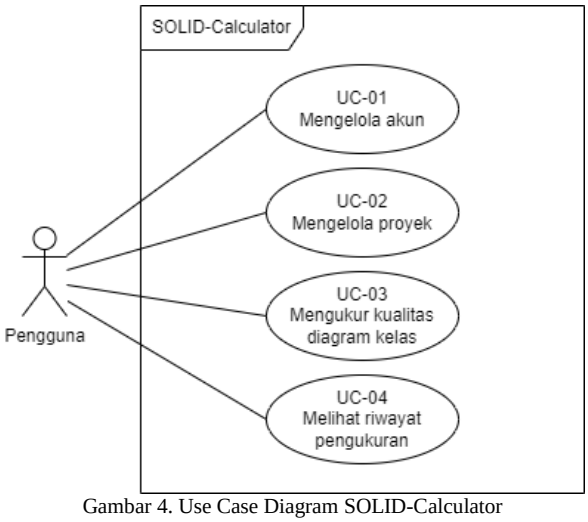
Kebutuhan fungsional

1. Aplikasi mampu mengelola akun
2. Aplikasi mampu mengelola proyek
3. Aplikasi mampu menyediakan fitur unggah diagram kelas dalam dokumen dengan ekstensi .mdj
4. Aplikasi mampu menerima unggahan dokumen secara berulang
5. Aplikasi mampu menyimpan data hasil pengukuran
6. Aplikasi mampu menampilkan hasil pengukuran dalam bentuk angka dan grafik progress bar
7. Aplikasi mampu memperbarui hasil pengukuran
8. Aplikasi mampu menampilkan riwayat pengukuran Prinsip SOLID yang pernah dilakukan dalam bentuk grafik garis

Kebutuhan non-fungsional

Performa: Aplikasi mampu memuat hasil pengukuran tidak lebih dari 5 detik.

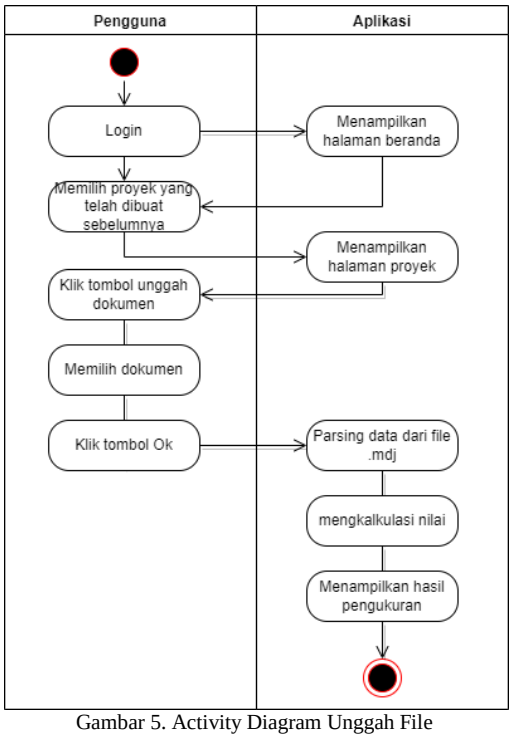
Dari daftar kebutuhan tersebut, dibuatlah *Use Case Diagram* yang dapat dilihat pada Gambar 4. Terdapat 4 buah *use case* yang merepresentasikan jumlah *end to end* yang ada dalam aplikasi yaitu mengelola akun, mengelola proyek, mengukur kualitas diagram kelas, dan melihat riwayat pengukuran.



Gambar 4. Use Case Diagram SOLID-Calculator

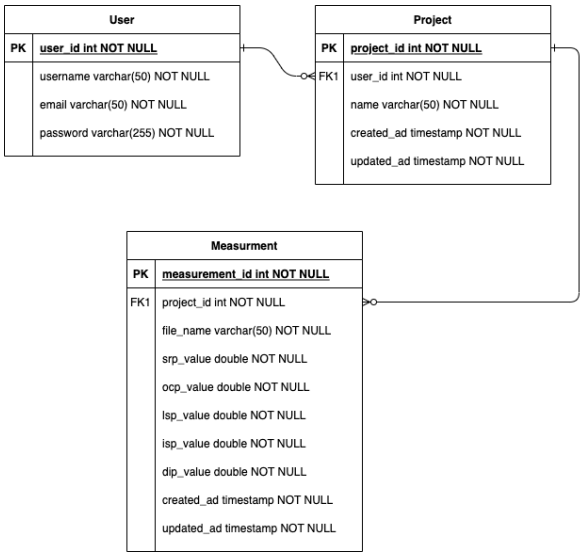
3.2 Desain

Pada tahap ini, dibuat beberapa artefak desain diantaranya *Activity Diagram*, *Physical Data Model* (PDM), *Class Diagram*, dan antarmuka *Low Fidelity Prototype*. Desain-desain tersebut dibuat untuk memberikan gambaran bagaimana cetak biru aplikasi yang akan dibangun. Selain itu dibuat pula dokumen kasus uji yang digunakan sebagai skenario pengujian aplikasi.



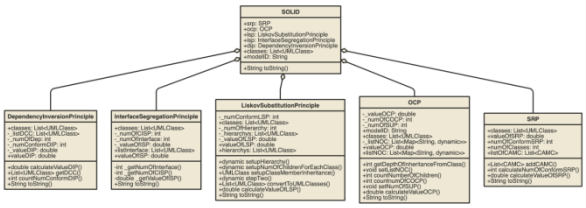
Gambar 5. Activity Diagram Unggah File

Gambar 5 merupakan salah satu Activity Diagram yang merupakan pendefinisian detail alur *end to end* yang ada pada Use Case Diagram. Pada diagram tersebut dijelaskan alur perilaku aplikasi mulai dari halaman utama hingga berhasil mengunggah dan mendapatkan hasil pengukuran.



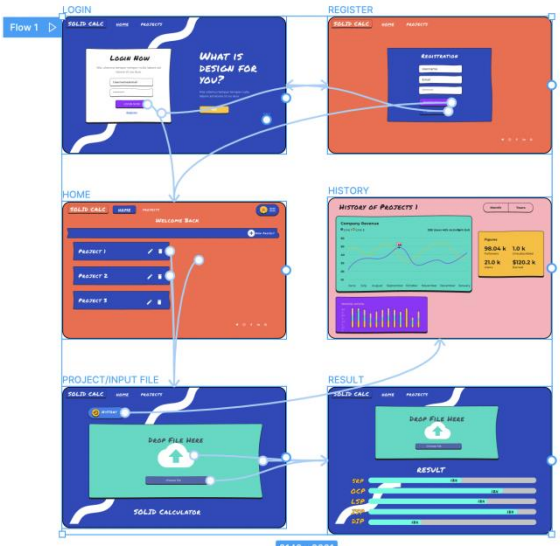
Gambar 6. *Physical Data Model* (PDM) SOLID-Calculator

Gambar 6 adalah desain fisik menyimpanan data yang akan disimpan dalam 3 tabel yaitu tabel user, tabel proyek dan tabel measurement. Setiap user dapat membuat lebih dari satu proyek dan setiap proyek dapat menyimpan lebih dari satu hasil pengukuran.



Gambar 7. Diagram Kelas

Diagram kelas pada Gambar 7 mengikuti model arsitektur pengembangan yang disebut dengan *Clean Architecture*. Hal ini bertujuan agar aplikasi yang dibangun mempunyai kerangka pengembangan yang dapat dengan mudah akan dipahami dan digunakan kembali ketika diperlukan pengembangan lebih lanjut.



Gambar 8. Desain Antarmuka

Gambar 8 menggambarkan alur tampilan penggunaan aplikasi oleh pengguna nantinya. Desain antarmuka ini terdiri dari 6 halaman yaitu halaman login, register, home, history, input file, dan result.

3.3 Implementasi dan Pengujian Unit

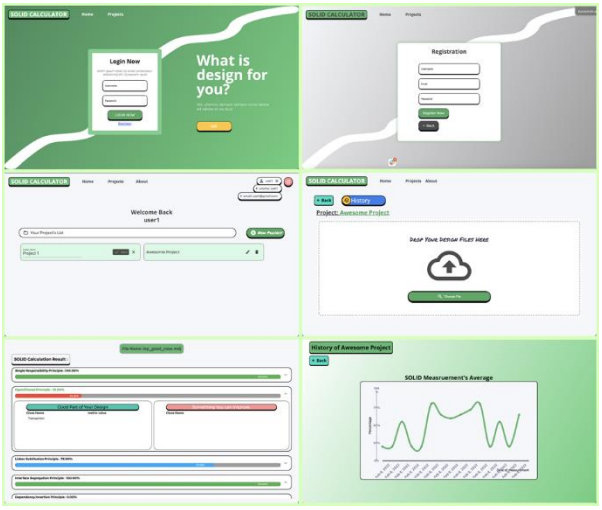
Pemrograman dilakukan dengan menggunakan bahasa *Dart* didukung dengan perkakas editor teks *Visual Code*. Pada tahap pemrograman, beberapa prinsip pemrograman, prinsip desain, serta *clean-code* diterapkan. Setiap unit yang selesai dibuat dipastikan dapat berjalan baik dibuktikan dengan pengujian unit secara langsung.

3.4 Pengujian terintegrasi dan pengujian sistem

Pada tahap ini, pengujian terintegrasi dilakukan yaitu menguji setiap unit yang saling berhubungan, memastikan bahwa hubungan tersebut dapat berjalan dengan baik. Setelah semua selesai maka dilakukan pengujian sistem secara *black box* mengikuti dokumen kasus uji yang sebelumnya telah dibuat. Semua pengujian dilakukan oleh pemrogram.

4. Hasil dan Pembahasan

Berikut ini adalah beberapa tangkapan layar dari tampilan hasil pengembangan Aplikasi *SOLID-Calculator*. Aplikasi ini berjalan pada platform web dan dapat digunakan oleh siapa saja terutama untuk orang bertugas sebagai perancang perangkat lunak. Adapun halaman utama dari aplikasi ini adalah halaman home, unggah dokumen desain, hasil pengukuran, riwayat pengukuran, dan profil. Tampilan dapat dilihat pada Gambar 9.



Gambar 9 Tampilan Aplikasi SOLID-Calculator

Setelah aplikasi selesai dibangun, dilakukan pengujian dengan kasus uji yang sudah dipersiapkan, hasilnya terbukti bahwa semua kebutuhan sudah dapat berjalan dengan baik. Hasil pengujian tersebut dapat dilihat pada Tabel 1.

Tabel 1 Test Case Hasil Pengujian

Test Scenario	Test Step	Test Data	Actual Result	Link	Status
Login berhasil dengan data yang valid	1. Buka aplikasi 2. Isi form login dengan data yang valid. 3. Pilih tombol ”Login Now”	username: user1 password: user123	Berhasil login dengan data yang valid, dan diarahkan ke halaman utama dari aplikasi.	https://bit.ly/37wUzFv https://bit.ly/37oYBjs	✓
Login gagal dengan data yang tidak valid	1. Buka aplikasi 2. Isi form login dengan data yang tidak valid, atau tanpa data sama sekali 3. Pilih tombol ”Login Now”	username: random password: random	Login gagal, lalu muncul alert pemberitahuan bahwa data yang dimasukkan salah atau tidak valid	https://bit.ly/3Ok4hM1	✓
Registrasi berhasil dengan data valid	1. Buka Aplikasi 2. Pilih menu “Register” 3. Isi form registrasi dengan data yang valid 4. Pilih tombol Register	email: email@gmail.com username: user1 password: user123	Berhasil registrasi dan pengguna diarahkan ke halaman utama aplikasi.	https://bit.ly/38UvIM9	✓
Registrasi gagal dengan data yang tidak balid	1. Buka aplikasi 2. Pilih menu“Register” 3. Isi form registrasi dengan data yang tidak balid 4. Pilih tombol Register	email: email#m username: user1 password: user123	Gagal registrasi dan pengguna diberikan alert peringatan bahwa data yang dimasukkan tidak valid.	https://bit.ly/3L0u2is https://bit.ly/3Ok4SgJ	✓

Test Scenario	Test Step	Test Data	Actual Result	Link	Status
Berhasil membuat proyek baru	1. Buka Aplikasi 2. Pilih tombol "New Project" 3. Isi form dengan nama proyek 4. Pilih tombol save	project name : myproject	Berhasil membuat proyek baru	https://bit.ly/3EqHZ6J https://bit.ly/37YuyyS	✓
Gagal membuat proyek baru	1. Buka Aplikasi 2. Pilih tombol "New Project" 3. Kosongkan form nama proyek 4. Pilih tombol save	project name :-	gagal membuat proyek baru	https://bit.ly/3rBM6rn https://bit.ly/391kCFn	✓

5. Kesimpulan

Setelah pengembangan aplikasi dilakukan mulai dari tahap pengumpulan kebutuhan hingga pengujian, maka dihasilkanlah Aplikasi berbasis web bernama SOLID-*Calculator*.

Penelitian ini telah menghasilkan perangkat lunak siap pakai untuk mengukur tingkat kesesuaian desain diagram kelas dengan prinsip SOLID. Perangkat lunak yang dihasilkan mampu melakukan pengukuran dengan tepat dan cepat, selain itu perangkat lunak juga dapat mendeteksi potensi kesalahan dengan menampilkannya dalam bentuk daftar kelas yang perlu diperbaiki. Dengan informasi tersebut diharapkan dapat menjadi pertimbangan pendesain untuk memperbaiki desainnya, sehingga dapat mengurangi kesalahan sejak tahap desain. Diharapkan dengan kesalahan yang minimal dan kualitas yang baik maka perangkat lunak dapat dengan cepat dirilis dan mendapatkan sambutan yang antusias dari pengguna.

6. Daftar Rujukan

[1] K. A. Briski *et al.*, "Minimizing code defects to improve software quality and lower development costs .," *Dev. Solut.*, no. October, p. 12, 2008.

[2] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices.pdf*. .

[3] I. Oktafiani, "Software Metrics Proposal for Conformity Checking of Class Diagram to SOLID Design Principles," *Int. Conf. Data Softw. Eng.*, vol. 5, 2019, [Online]. Available: https://ieeexplore.ieee.org/document/8705857.

[4] G. Langdale and D. Lemire, "Parsing gigabytes of JSON per second," *VLDB J.*, vol. 28, no. 6, pp. 941–960, 2019, doi: 10.1007/s00778-019-00578-5.

[5] H. Eriksson, M. Penker, B. Lyons, and D. Fado, *UML 2 toolkit*. 2003.

[6] StarUML, "StarUML Documentation." https://docs.staruml.io/ (accessed Apr. 19, 2022).

[7] Flutter, "Flutter documentation." https://docs.flutter.dev/ (accessed Apr. 19, 2022).

[8] M. D. Normansyah and I. Oktafiani, "Pengembangan Perangkat Lunak Pengukuran Kepatuhan Desain Diagram Kelas Terhadap Prinsip SOLID," Universitas Pertamina, 2021.

[9] I. Sommerville, *Software Engineering*. 2013.