BAB 2
TINJAUAN PUSTAKA

2.1 Landasan Teori

Beberapa teori dasar yang mendukung penelitian ini meliputi konsep-konsep utama yang digunakan untuk mengembangkan aplikasi.

2.1.1 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses evaluasi terhadap aplikasi atau sistem untuk memastikan bahwa perangkat lunak tersebut memenuhi kebutuhan pengguna dan bebas dari bug (Nur Syihab & Sugiarti, 2024). Tujuan dari pengujian adalah untuk meningkatkan kualitas perangkat lunak dengan mendeteksi kesalahan atau masalah pada tahap pengembangan sebelum perangkat lunak digunakan. Dengan pengujian yang tepat, perangkat lunak dapat memberikan pengalaman pengguna yang optimal serta mengurangi risiko kegagalan saat digunakan pengguna nantinya.

Pengujian dapat dibedakan menjadi 2 yaitu:

1. White-box testing, yaitu pengujian fokus pada bagian dalam kode program, seperti alur logika, jalur kode, atau fungsi tertentu. Metode ini biasanya digunakan untuk memastikan bahwa semua alur dalam kode dapat berjalan dengan benar.
2. Black-box testing, yaitu fokus pada pengujian fungsionalitas perangkat lunak tanpa melihat bagaimana kode ditulis. Penguji hanya memeriksa apakah input yang dimasukkan menghasilkan output yang sesuai, tanpa perlu tahu cara kerja kode di balik layar.

Salah satu metode pengujian white-box adalah unit test, metode pengujian di mana bagian terkecil dari perangkat lunak dalam hal ini adalah method atau function. Tujuannya untuk memastikan program berfungsi sesuai dengan yang diharapkan, dalam unit test dikenal istilah test case, yaitu skenario uji yang dirancang untuk menguji suatu fungsi atau metode tertentu. test case sendiri terdiri atas 2 yaitu input dan expected output, test case membantu memastikan bahwa semua kondisi dari unit yang diuji berfungsi dengan benar.

2.1.2 Alur Kontrol

Dalam sebuah program, alur kontrol menentukan instruksi yang dieksekusi terlebih dahulu . Kondisi tertentu mempengaruhi bagaimana eksekusi berjalan dan berpindah antar instruksi. Alur kontrol juga memberikan struktur serta menentukan logika pemrograman, terutama dalam pengambilan keputusan dan perulangan tugas.

Alur kontrol mencakup beberapa konsep utama yang dapat mengubah arah eksekusi program dimulai dari percabangan, yaitu mekanisme yang memungkinkan program untuk memilih jalur eksekusi yang berbeda berdasarkan kondisi tertentu. Struktur percabangan yang paling umum adalah if-else. Dengan percabangan, program dapat membuat keputusan berdasarkan nilai variabel atau hasil evaluasi ekspresi, Gambar 2.1 merupakan contoh dari alur kontrol untuk percabangan.

```
if x > 0:
    print("Positif")
else:
    print("Negatif atau Nol")
```

Gambar 2. 1 Contoh alur kode percabangan
Sumber: Dokumentasi pribadi

Berikutnya adalah alur pengulangan, memungkinkan eksekusi suatu blok kode berulang kali berdasarkan kondisi tertentu. Ada beberapa jenis pengulangan yang umum digunakan, seperti for, while, dan do-while. Pengulangan sangat penting dalam pemrograman untuk memproses koleksi data, menjalankan algoritma, atau melakukan perhitungan berulang, Gambar 2.2 merupakan contoh dari kode alur kontrol perulangan.

```
# perulangan for
for i in range(5):
    print("Nilai i:", i)
```

(a)

```
# perulangan while
i = 0
while i < 5:
    print("Nilai i:", i)
    i += 1
```

(b)

Gambar 2. 2 (a) Contoh perulangan for, (b) Contoh perulangan while
Sumber: Dokumentasi pribadi

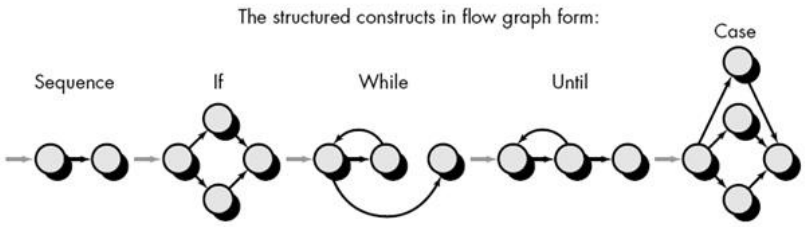
Dalam pengujian perangkat lunak alur kontrol sering menjadi target pengujian, bagian ini sering digunakan untuk memastikan bahwa program dapat

menangani berbagai skenario kondisi dan perulangan dengan benar (Setyawan dkk., 2020). Pengujian alur kontrol dilakukan dengan menguji apakah percabangan dan pengulangan bekerja sesuai dengan yang diharapkan, baik dalam kondisi normal maupun dalam kondisi ekstrem. Misalnya, pengujian dapat memeriksa apakah program memilih jalur yang tepat dalam percabangan berdasarkan nilai variabel atau apakah pengulangan berhenti pada kondisi yang benar.

Alur kontrol yang baik membantu memastikan bahwa program memiliki alur yang jelas dan dapat menangani semua kemungkinan kondisi yang muncul selama eksekusi. Oleh karena itu, penting untuk memahami berbagai jenis alur kontrol, serta cara menanganinya dengan benar dalam program. Dengan demikian, alur kontrol menjadi bagian yang sangat penting dalam pemrograman untuk mengatur logika dan jalannya aplikasi yang stabil dan dapat diandalkan.

2.1.3 Control Flow Graph

Control Flow Graph (CFG) adalah representasi grafis yang menunjukkan alur eksekusi suatu program dalam bentuk graf (Shiddiq, 2022). Dalam CFG, setiap simpul (node) menggambarkan suatu blok instruksi atau pernyataan, sedangkan setiap garis penghubung (edge) menggambarkan transisi atau aliran kontrol dari satu blok ke blok lainnya. Pada Gambar 2.3 diperlihatkan bentuk alur untuk setiap struktur dasar program, seperti sequence, condition (if-else), dan iteration (loop), jika digambarkan dalam bentuk CFG.



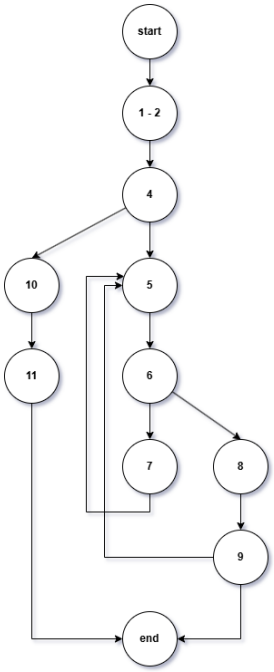
Gambar 2. 3 Struktur dari Control Flow Graph
Sumber: (Binus University School Of Computer Science, 2016)

Pada Gambar 2.4 diperlihatkan salah satu contoh bentuk penggambaran dari Control Flow Graph berdasarkan kode program python.

```python
angka_awal = 1
angka_akhir = 5

if angka_awal < angka_akhir:
    for angka in range(angka_awal, angka_akhir + 1):
        if angka % 2 == 0:
            print(f"{angka} adalah angka genap")
        else:
            print(f"{angka} adalah angka ganjil")
else:
    print("Rentang angka tidak valid")
```

(a)

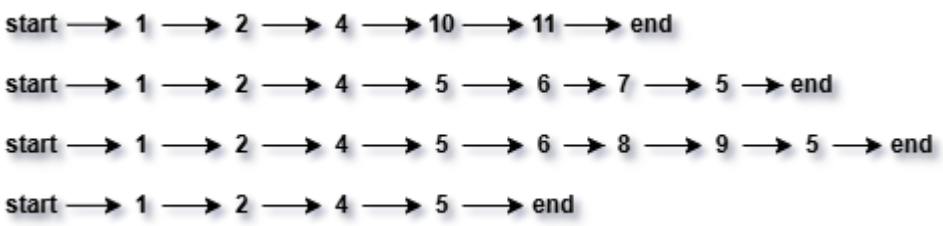


(b)

Gambar 2. 4 (a) Contoh kode transformasi ke CFG, (b) Control Flow Graph
Sumber: Dokumentasi pribadi

Hasil dari CFG digunakan untuk membuat jalur-jalur yang mungkin dilewati program untuk path testing. Path Testing merupakan salah satu metode pengujian yang menggunakan kode program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang test case. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Asri dkk., 2018).

Sebagai contoh Gambar 2.5 adalah hasil generate path untuk CFG yang ada pada Gambar 2.4 (b) sebelumnya, dari program tersebut didapatkan 4 jalur.



Gambar 2. 5 Hasil Generate Path CFG
Sumber: Dokumentasi Pribadi

Kemudian dari jalur tersebut akan ditentukan test case yang berkemungkinan

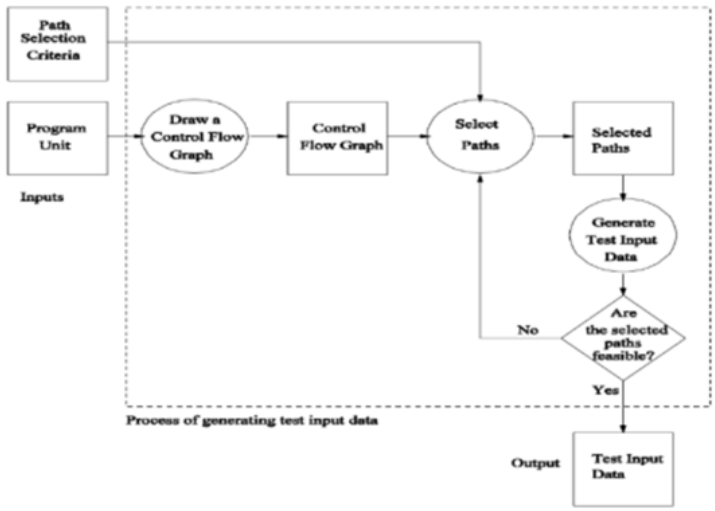
akan melewati jalur tersebut atau tidak. Untuk mengetahui jumlah jalur yang ingin diuji bisa menggunakan metode penghitungan cyclomatic complexity dengan persamaan 2-1, yang mana jumlah edge dengan node yang ada pada CFG akan dikurangkan kemudian akan dijumlahkan dengan angka konstanta yaitu dua.

$$V(G) = E - N + 2 \qquad 2\text{-}1$$

dimana

V(G) : Nilai cylomatic complexity yang dicari
E : Jumlah Edge (sisi) CFG
N : Jumlah Node (simpul) CFG

Gambar 2.6 adalah langkah atau alur pembuatan test case menggunakan Control flow graph.



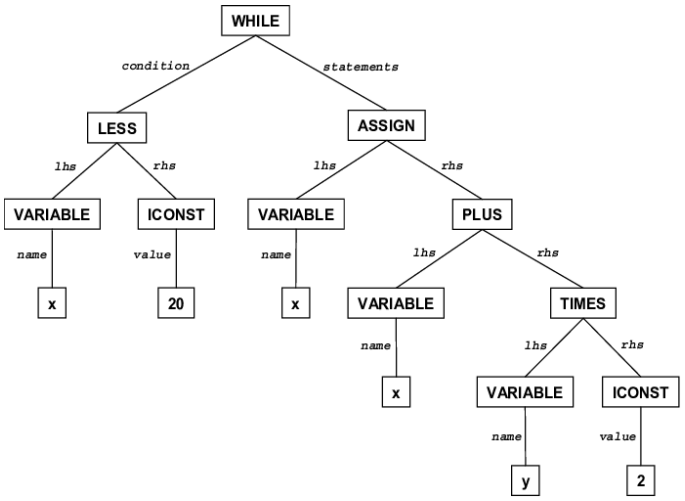
Gambar 2. 6 Alur Pembuatan Test Case menggunakan CFG
Sumber: (Naik & Tripathy, University of Waterloo)

Dari proses pembuatan test case pada Gambar 2.6, dibagian penggambaran Control Flow Graph dan generate path memakan waktu cukup lama dan rawan akan kesalahan, karena perlu ketelitian dan pemahaman mendalam mengenai kode program yang ditulis.

2.1.4 Abstract Syntax Tree (AST)

Abstract Syntax Tree (AST) adalah sebuah representasi struktur sebuah kode dengan bahasa pemrograman tertentu (Rusdianto & Chaniago, 2019). Setiap simpul dalam AST merepresentasikan elemen dalam kode, seperti ekspresi, pernyataan,

atau blok kode. AST membantu dalam analisis kode karena menyediakan struktur hierarki yang dapat dimanfaatkan untuk membangun CFG.



Gambar 2. 7 Contoh struktur AST dari kode program
Sumber: StackExchange

Pada gambar 2.8 (b) diperlihatkan bagaimana bentuk AST dari kode program yang sudah di konversi menggunakan library AST milik python. Jika diperhatikan dapat lihat bahwa setiap syntax akan disusun berdasarkan jenis dan urutan nya mengikuti kode program, mulai dari type, operator, ekspresi hingga ke value nya.

```
if a % 2 == 0:
```

(a)

```
{
  "_type": "If",
  "test": {
    "_type": "Compare",
    "left": {
      "_type": "BinOp",
      "left": { "_type": "Name", "id": "a" },
      "op": { "_type": "Mod" },
      "right": { "_type": "Constant", "value": 2 }
    },
    "ops": [{ "_type": "Eq" }],
    "comparators": [{ "_type": "Constant", "value": 0 }]
  }
}
```

(b)

Gambar 2. 8 (a) Kode program, (b) Hasil AST
Sumber: Dokumentasi pribadi

Dari beberapa penelitian yang pernah dilakukan AST dapat digunakan sebagai dasar untuk membuat visualisasi CFG, dengan alasan berikut:

1) Representasi struktur program

   AST menyusun kode program dalam bentuk tingkatan, di mana setiap node mengungkapkan hubungan antara elemen-elemen dalam kode, sehingga

dapat menelusuri urutan eksekusi yang membantu memahami bagaimana alur berpindah dari satu bagian kode ke bagian lainnya.

2) Identifikasi blok dalam program

   Untuk membangun CFG, maka perlu memetakan blok-blok dasar dalam program, seperti pernyataan kontrol (if, while, for), menentukan cabang alur logika, juga menentukan jalur berurutan dalam alur eksekusi. AST memudahkan proses ini karena simpul-simpulnya menyimpan informasi jenis pernyataan atau instruksi yang digunakan.

3) Kemampuan penelusuran yang baik

   AST memungkinkan penelusuran menggunakan algoritma seperti Depth First Search sehingga setiap simpul dan cabang dapat dianalisis secara sistematis.

Berikut ini adalah proses yang akan digunakan untuk melakukan transformasi dari AST ke CFG yang akan melibatkan beberapa langkah berikut:

1) Identifikasi Kode dari AST

   Menganalisis struktur AST untuk mengidentifikasi elemen-elemen utama program. Proses ini melibatkan pengenalan tipe simpul seperti kondisi, perulangan, atau pernyataan biasa, serta memahami hubungan antar simpul. Informasi ini membantu menentukan bagian mana dari kode yang merupakan titik awal, percabangan, atau penggabungan dalam alur kontrol program, yang menjadi dasar pembuatan CFG.

2) Pemisahan Blok Dasar

   Setelah elemen-elemen AST diidentifikasi, program dipisahkan menjadi kumpulan pernyataan yang dieksekusi secara berurutan tanpa adanya percabangan. Setiap blok dasar mencakup pernyataan yang saling terhubung hingga terdapat kondisi, perulangan, atau akhir program. Blok dasar ini nantinya menjadi simpul awal yang merepresentasikan elemen-elemen dalam CFG.

3) Pembuatan Simpul CFG dari Simpul AST

   Setiap simpul dalam AST dikonversi menjadi simpul dalam CFG berdasarkan jenis dan fungsinya. Simpul kondisi seperti if atau while menghasilkan percabangan dengan dua jalur (true dan false), sedangkan

simpul pernyataan biasa dihubungkan secara lurus ke simpul berikutnya.

4) Penambahan Hubungan Antar Simpul

   Setelah semua simpul dibuat, hubungan antar simpul atau edges ditambahkan untuk merepresentasikan alur kontrol program. Jalur lurus menghubungkan simpul yang dieksekusi secara berurutan, percabangan menghubungkan simpul kondisi ke jalur true dan false, sementara loop dihubungkan kembali ke simpul awal perulangan.

5) Visualisasi CFG

   Langkah terakhir adalah membuat visualisasi grafis dari CFG, visualisasi ini bisa menggunakan library seperti React Flow, dimana setiap simpul dan hubungan diberi label sesuai jenis blok eksekusi.

2.1.5 Research and Development (R&D)

Metode penelitian Research and Development (R&D) adalah metode penelitian yang digunakan untuk menghasilkan produk tertentu dan menguji keefektifan produk tersebut (Sugiyono, 2020). Sugiyono menjelaskan bahwa metode ini memiliki 4 level yaitu:

Meneliti untuk menghasilkan rancangan produk.

1) Level ini berfokus pada penelitian yang dilakukan untuk memahami kebutuhan dan merancang produk yang belum ada sebelumnya, hasil nya hanya berupa rancangan.
2) Tanpa meneliti hanya menguji produk yang telah ada.

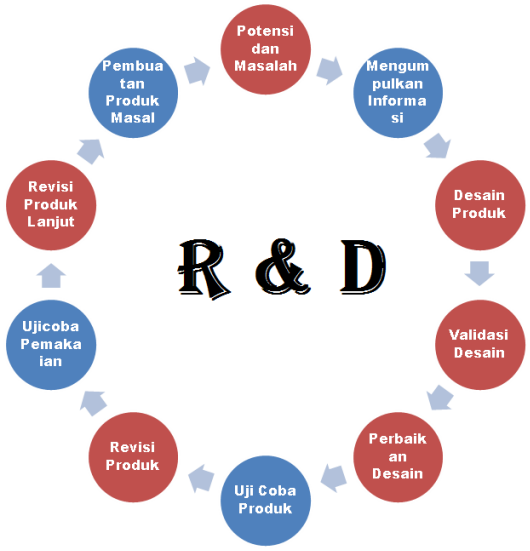
   Peneliti hanya menguji keefektifan atau kinerja dari produk yang sudah ada tanpa pengembangan lebih lanjut.
3) Meneliti dan mengembangkan produk yang telah ada dan mengujinya.

   Produk yang sudah ada diteliti lebih lanjut untuk menemukan kekurangannya, lalu dikembangkan dengan inovasi baru dan diuji kembali.
4) Meneliti dan menciptakan produk baru dan mengujinya.

   Produk yang benar-benar baru diciptakan berdasarkan hasil penelitian mendalam, kemudian diuji untuk memastikan kualitas dan keberfungsiannya.

Gambar 2. 9 Tingkatan metode R&D
Sumber: (Sugiyono, 2020)

Penelitian ini berfokus ke tingkatan 3 yaitu meneliti dan mengembangkan produk yang telah ada dan mengujinya. Dalam penelitian ini, metode R&D diterapkan menggunakan model R&D yang sudah di kembangkan oleh Sugiyono. Pada gambar 2.10 diperlihatkan bagaimana bentuk alur dari metode R&D.



Gambar 2. 10 Tahapan metode R&D
Sumber: (Sugiyono, 2020)

Metode ini terdiri menjadi 2 yaitu tahapan riset dan pengembangan, untuk riset berakhir pada langkah validasi desain yang mana dihasilkan sebuah spesifikasi dari produk yang akan di kembangkan, tahapan ini cenderung didasarkan dengan data kualitatif. Sedangkan untuk tahapan berikutnya adalah pengembangan yang terdiri dari membangun produk dan mengujinya, biasanya menggunakan pengukuran kuantitatif. Berikut adalah penjelasan dari setiap langkah-langkah yang

ada pada metode pengembangan R&D.

1) Potensi dan Masalah

   Langkah pertama dalam R&D adalah mengidentifikasi potensi dan masalah yang ada. Proses ini bertujuan untuk memahami kebutuhan yang belum terpenuhi atau permasalahan yang dihadapi terkait produk yang serupa, dari tahapan ini akan kita rumuskan inovasi ataupun evaluasi apa yang dilakukan sehingga harapannya adalah adanya peningkatan dari aplikasi yang akan dibangun.

2) Pengumpulan Informasi

   Pada tahap ini, data yang relevan dikumpulkan untuk memperoleh wawasan mendalam terkait produk yang akan dikembangkan. Informasi tersebut dapat diperoleh melalui dua sumber utama, yaitu penelitian lapangan dan studi literatur yang berkaitan dengan pengembangan aplikasi serupa. Studi literatur dapat berupa jurnal atau publikasi lain yang relevan dengan topik yang digunakan dalam merancang aplikasi. Selain itu, penelitian lapangan juga dilakukan, yang dapat menggunakan pendekatan kualitatif. Salah satu contoh metode kualitatif yang dapat diterapkan adalah analisis terhadap aplikasi-aplikasi yang sudah ada atau kompetitor, melalui percobaan dalam skala kecil. Langkah ini bertujuan untuk mengidentifikasi kelebihan dan kekurangan dari aplikasi-aplikasi yang telah ada, sehingga hasilnya dapat menjadi acuan dalam pembuatan desain produk yang akan dibangun.

3) Desain Produk

   Berdasarkan hasil analisis dari tahap pengumpulan informasi, produk awal dirancang dengan mempertimbangkan kebutuhan pengguna dan tujuan penelitian. Desain atau rancangan akan menjadi spesifikasi untuk mengembangkan produk, sehingga peneliti memiliki panduan selama tahapan pengembangan dan pengujian yang dilakukan.

4) Validasi Desain

   Pada tahapan ini desain rancangan aplikasi divalidasi oleh ahli ataupun pakar, seperti dosen atau pengguna yang memahami kebutuhan teknis. Proses validasi bertujuan untuk memastikan bahwa desain yang dibuat telah sesuai dengan kebutuhan, sehingga didapatkan sebuah desain yang sudah

teruji. Masukan yang diperoleh dari ahli digunakan untuk mengevaluasi kelayakan desain sebelum dilanjutkan ke tahap pengembangan menjadi sebuah produk.

5) Perbaikan atau Revisi Desain

   Berdasarkan masukan dari proses validasi, desain produk diperbaiki atau direvisi untuk mengatasi kekurangan atau ketidaksesuaian yang teridentifikasi. Selain revisi desain disini juga akan dilakukan tahapan untuk mulai menjadikan desain atau rancangan tersebut menjadi sebuah aplikasi awal atau prototipe.

6) Uji Coba Produk

   Produk yang telah divalidasi selanjutnya dilakukan pengujian internal terhadap aplikasi, untuk mengevaluasi fungsionalitas dasar serta kesesuaiannya dengan desain yang sudah dibuat.

7) Revisi Produk

   Apabila terdapat kekurangan atau potensi perbaikan yang ditemukan selama uji coba tahap pertama, revisi dilakukan untuk meningkatkan kualitas produk. Tahap ini memastikan produk semakin mendekati kebutuhan pengguna. Revisi dilakukan dengan tetap mengacu pada desain atau rancangan yang sudah dibuat.

8) Uji Coba Pemakaian Produk

   Setelah revisi awal selesai, produk kembali diuji dalam konteks penggunaan yang lebih mendekati kondisi nyata, yaitu pengujian lapangan. Dengan melakukan pengujian untuk mengevaluasi performa, kestabilan dan kemudahan penggunaan produk. Tahap ini bertujuan untuk memastikan produk dapat memenuhi kebutuhan pengguna dengan baik.

9) Revisi Produk lanjut

   Apabila ditemukan kekurangan atau potensi perbaikan selama uji coba pemakaian oleh pengguna, revisi lanjutan dilakukan untuk menyempurnakan produk.

10) Pembuatan Produk akhir

    Setelah melalui serangkaian proses revisi dan uji coba, produk disempurnakan menjadi versi final yang siap diimplementasikan. Versi

akhir ini diharapkan dapat memenuhi kebutuhan pengguna secara keseluruhan.

2.2 Penelitian Terdahulu

Penelitian terdahulu terkait dengan Visualisasi CFG telah pernah dilakukan. Penelitian oleh Asri dkk. (2018) mengembangkan aplikasi yang mampu melakukan instrumentasi otomatis pada kode program, menghasilkan CFG, menghitung cyclomatic complexity, dan mengidentifikasi jalur-jalur yang ada untuk pengujian. Penelitian ini difokuskan pada kode program Matlab, di mana aplikasi yang dikembangkan dapat membantu pemrogram dalam menganalisis struktur kode serta memahami kompleksitas dari sebuah alur program.

Penelitian oleh Putra dkk. (2021) yang bertujuan untuk membangun AST dan CFG dalam notasi algoritmik sebagai alat bantu untuk merepresentasikan alur eksekusi kode. AST dan CFG digunakan untuk menunjukkan bagaimana struktur algoritma dan alur kontrol suatu program bekerja. Penelitian ini ditujukan untuk pembelajaran, di mana mahasiswa atau pengembang dapat menggunakan graf untuk memahami bagaimana kode beroperasi, sehingga meningkatkan proses pembelajaran algoritma dan analisis kode.

Penelitian lain juga dilakukan oleh Sirait (2021) yang memanfaatkan teori graf, khususnya *Control Flow Graph*, untuk menganalisis cyclomatic complexity dari suatu algoritma. Dengan memanfaatkan *Contol Flow Graph*, penelitian ini memberikan cara untuk menilai dan membandingkan kompleksitas berbagai algoritma, sehingga dapat digunakan untuk memilih atau mengoptimalkan algoritma dengan kompleksitas yang lebih rendah.

Penelitian selanjutnya oleh Putri dkk. (2024) mengembangkan perangkat lunak berbasis web yang dapat menghitung Cyclomatic Complexity dari kode program dengan memanfaatkan CFG. Perangkat lunak ini membantu pemrogram mengidentifikasi bagian kode yang kompleks, sehingga memungkinkan perbaikan yang dapat meningkatkan efisiensi pengembangan perangkat lunak.

Tabel 2. 1 Penelitian Terdahulu

Judul	Fokus	Hasil	Gap
Instrumentasi Kode Program	Mengembangkan aplikasiuntuk instrumentasi	Membantu pemrogram dalam menganalisis struktur	Fokus pada kode program Matlab belum mencakup

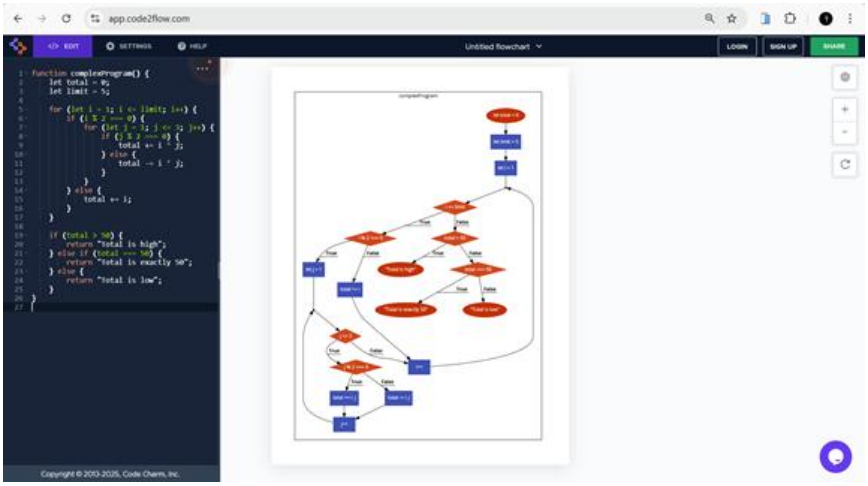
Judul	Fokus	Hasil	Gap
Secara Otomatis untuk Path Testing	otomatis, pembangkitan CFG, dan perhitungan cyclomatic complexity pada kode Matlab	kode serta memahami kompleksitas jalur control	Bahasa pemrograman lain atau platform yang lebih umum
Abstract Syntax Tree (AST) and Control Flow Graph (CFG) Construction of Algorithmic Notation	Membangun AST dan CFG untuk Representasi Alur control kode dalam notasi algoritmik untuk tujuan pembelajaran	Alat bantu untuk memahami struktur algoritma dan alur kontrol,meningkatkan pembelajaran algoritma dan analisis kode	Fokus pada visualisasi dari kode program saja, belum mencakup anlisa jalur.
Analisis kompleksitas Siklomatik Suatu Algoritma dengan memanfaatkan teori graf	Menganalisis cyclomatic complexity dari algoritma dengan teori graf	Menilai dan membandingkan kompleksitas algoritma untuk memilih atau mengoptimalkan algoritma yang lebih baik	Tidak ada aplikasi atau implementasi praktis
Rancang Bangun Perangkat Lunak Perhitungan Metrik Cyclomatic Complexity Berdasarkan Control Flow Graph Berbasis Web	Mengembangkan perangkat lunak berbasis web untuk menghitung cyclomatic complexity menggunakan CFG	Membantu pemrogram mengidentifikasi bagian kode yang kompleks untuk meningkatkan efisiensi dari sebuah kode program	Fokus pada perhitungan cyclomatic complexity saja, belum mencakup visualisasi atau analisis jalur kontrol lebih lanjut

Tabel 2. 2 Penelitian Saat Ini

Judul	Fokus	Hasil diharapkan	Gap diperbaiki
Pengembangan Aplikasi Visualisasi CFG untuk Analisis Alur Kontrol pada Kode Program	Mengembang Kan aplikasi visualisasi CFG serta jalur yang dilewati untuk analisis alur kontrol dengan fitur pengujian test case untuk keperluan unit test pada kode program python	Menghasilkan Control Flow Graph beserta semua jalurnya,dan fitur pengujian test case yang bisa digunakan untuk mendukung pembuatan test case dari unit test.	Menyediakan visualisasi langsung yang dapat membangkitkan seluruh jalur serta membantu pengujian dengan menguji jalur menggunakan input test case berupa nilai untuk parameter dari method yang diuji.

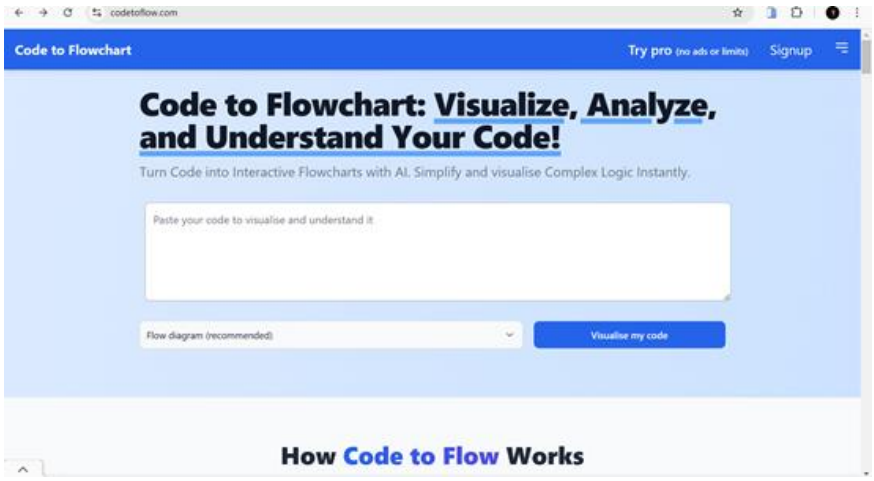
Selain beberapa penelitian tersebut, juga terdapat beberapa aplikasi yang

sudah pernah dibuat sebelumnya, yang pertama adalah code2flow yaitu aplikasi visualisasi kode program yang menggunakan notasi flowchart untuk menyajikan alur kode. Aplikasi ini memungkinkan visualisasi secara interaktif, di mana pengguna hanya perlu menuliskan kode, dan alur visualisasinya akan terbentuk secara otomatis. Berdasarkan dokumentasi resminya, proses visualisasi tidak dilakukan langsung pada bahasa pemrograman, melainkan hanya mengambil sintaks atau operator yang umum digunakan. Contoh sintaks tersebut adalah if, else, while, switch dan operator serupa lainnya. Dengan demikian, yang diterjemahkan ke dalam visualisasi hanyalah bagian-bagian tertentu dari kode.



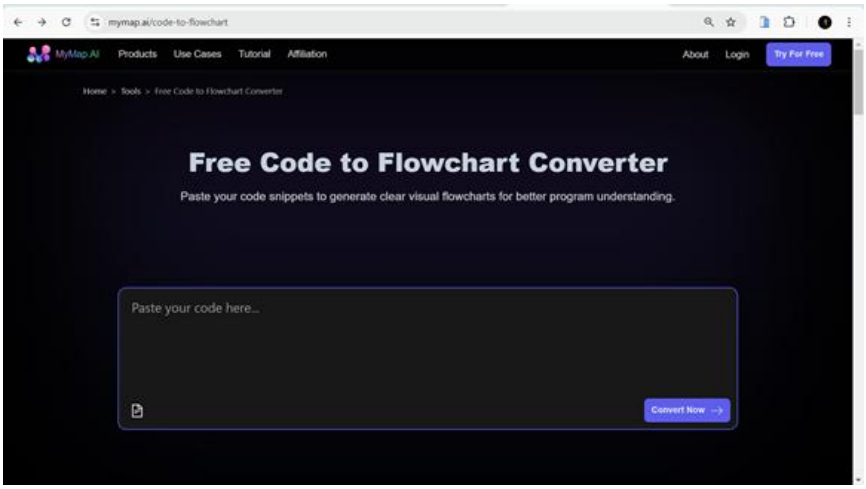
Gambar 2. 11 Tampilan Aplikasi Code2Flow
Sumber: (Code2flow.com, 2024)

Aplikasi selanjutnya adalah CodeToFlow, aplikasi yang memanfaatkan Generative AI untuk mengubah kode menjadi notasi yang dapat divisualisasikan. Berdasarkan dokumentasinya, aplikasi ini menghasilkan flowchart dengan menggunakan notasi yang sesuai dengan pustaka Mermaid.js. Hal ini bisa dilihat ketika pengguna mencoba fitur edit, aplikasi akan menampilkan notasi yang digunakan untuk membuat diagram tersebut, notasi ini berfungsi sebagai dasar bagi library untuk membangun visualisasi flowchart secara otomatis. Oleh karena itu, input kode harus sesuai dengan notasi yang didukung oleh pustaka Mermaid.js agar dapat divisualisasikan dengan baik.



Gambar 2. 12 Tampilan Aplikasi CodeToFlow
Sumber: (CodeToFlow.com, 2024)

MyMap.ai memiliki pendekatan yang mirip dengan CodeToFlow, baik dalam konsep maupun teknologi yang digunakan. Aplikasi ini juga memanfaatkan Generative AI untuk menerjemahkan kode menjadi notasi diagram yang sesuai dengan pustaka Mermaid.js. Pustaka ini kemudian digunakan untuk membangun flowchart secara otomatis berdasarkan notasi yang dihasilkan. Dengan demikian, MyMap.ai memberikan kemudahan bagi pengguna untuk memahami alur kode melalui visualisasi yang sederhana.



Gambar 2. 13 Tampilan Aplikasi MyMap.Ai
Sumber: (MyMap.Ai, 2024)

Pada tabel 2.3 telah diperlihatkan beberapa aplikasi yang sudah ada.

Tabel 2. 3 Aplikasi Kompetitor

Aplikasi	Platform	Teknologi digunakan	Output
Code2Flow	Web	Penggunaan algoritma dengan basis nya adalah aturan yang sudah ditentukan	Flow Graph dengan notasi Flowchart secara interaktif.
CodeToFlow	Web	Menggunakan teknologi Generative AI untuk mengonversi kode ke flow graph	Menghasilkan flow graph
MyMap.Ai	Web	Menggunakan Teknologi Generative AI untuk mengonversi kode ke flow graph	Menghasilkan flow graph beserta penjelasannya