# ABSTRAK

Penentuan jalur pengujian (*path testing*) merupakan tahapan krusial dalam metode pengujian *White Box* untuk menjamin seluruh alur logika dalam kode program telah terverifikasi secara menyeluruh. Namun, proses identifikasi jalur independen (*basis path*) dan pembangkitan data uji secara konvensional atau manual memiliki kelemahan signifikan, yaitu risiko kesalahan manusia (*human error*) yang tinggi serta kebutuhan waktu yang besar, terutama pada perangkat lunak dengan struktur kontrol yang kompleks. Penelitian ini bertujuan untuk merancang dan membangun sebuah sistem otomatisasi pengujian *white box* yang mampu mengekstraksi jalur independen dan membangkitkan rencana pengujian (*test plan*) secara otomatis.

Metode yang digunakan dalam penelitian ini berbasis pada pemanfaatan *Abstract Syntax Tree* (AST) untuk membedah struktur kode sumber Python menjadi representasi hirarki data. Representasi tersebut kemudian ditransformasikan menjadi objek *Control Flow Graph* (CFG) yang merepresentasikan hubungan antar titik keputusan (*decision points*) dalam program. Sistem menghitung nilai kompleksitas siklomatis menggunakan standar McCabe untuk menentukan jumlah jalur independen minimum yang diperlukan. Penelusuran jalur dilakukan dengan mengimplementasikan algoritma *Depth-First Search* (DFS) yang dilengkapi dengan teknik *backtracking* guna memastikan pencapaian *path coverage* sebesar 100%.

Sistem ini memiliki keunggulan pada fitur *Automated Test Case Generation*, di mana sistem menggunakan teknik pemindaian ekspresi reguler (*regex extraction*) untuk menganalisis kondisi pada setiap *node* dan memberikan saran data input yang akurat agar jalur tersebut dapat dieksekusi. Selain itu, sistem bertindak sebagai *test oracle* dengan memprediksi ekspektasi *output* melalui mekanisme substitusi nilai variabel secara dinamis.

Efektivitas sistem diuji melalui dua skenario utama: logika aritmatika sederhana dan sistem analisis risiko investasi yang melibatkan struktur perulangan (*while loop*), operator logika majemuk (*and/or*), serta percabangan bersarang (*nested if*). Hasil pengujian menunjukkan bahwa sistem mampu mengidentifikasi seluruh jalur independen—termasuk jalur pengecualian seperti pembagian nol dan validasi data—secara presisi dan identik dengan hasil analisis manual, namun dengan efisiensi waktu pemrosesan yang jauh lebih tinggi. Luaran penelitian ini berupa laporan analisis komprehensif dalam format HTML yang menyajikan visualisasi grafis alur program, rincian metrika kompleksitas, serta tabel rencana pengujian yang siap digunakan oleh pengembang perangkat lunak.



**Kata Kunci:** *White Box Testing, Basis Path, Abstract Syntax Tree, Cyclomatic Complexity, Automated Test Case Generation, Control Flow Graph.*

## ABSTRACT

*Independent path identification is a critical stage in the White Box Testing method to ensure that all logical flows within a program's source code are comprehensively verified. However, conventional or manual processes for identifying independent paths (basis paths) and generating test cases possess significant weaknesses, including a high risk of human error and substantial time requirements, particularly for software with complex control structures. This research aims to design and develop an automated white box testing system capable of extracting independent paths and generating comprehensive test plans automatically.*

*The methodology employed in this study is based on utilizing the Abstract Syntax Tree (AST) to decompose Python source code into a hierarchical data representation. This representation is then transformed into a Control Flow Graph (CFG) object, illustrating the relationships between decision points within the program. The system calculates Cyclomatic Complexity using McCabe's standard to determine the minimum number of independent paths required for full coverage. Path traversal is executed by implementing the Depth-First Search (DFS) algorithm, integrated with backtracking techniques to guarantee 100% path coverage.*

*The system features an advanced Automated Test Case Generation capability, utilizing regular expression (regex) scanning techniques to analyze conditions at each node and provide accurate input data suggestions to trigger the execution of specific paths. Furthermore, the system serves as a test oracle by predicting expected outputs through a dynamic variable substitution mechanism.*

*The system's effectiveness was validated through two primary scenarios: simple arithmetic logic and a complex investment risk analysis system involving while loops, compound logical operators (and/or), and nested branching. The results demonstrate that the system can precisely identify all independent paths—including exception paths such as division by zero and data validation errors—with results identical to manual analysis while achieving significantly higher processing efficiency. The final output of this research is a comprehensive analysis report in HTML format, providing graphical visualizations of program flows, detailed complexity metrics, and ready-to-use test plan tables for software developers.*



***Keywords:*** *White Box Testing, Basis Path, Abstract Syntax Tree, Cyclomatic Complexity, Automated Test Case Generation, Control Flow Graph.*