SNESTIK

Seminar Nasional Teknik Elektro, Sistem Informasi, dan Teknik Informatika

https://ejurnal.itats.ac.id/snestik dan https://snestik.itats.ac.id

Informasi Pelaksanaan :
SNESTIK IV - Surabaya, 27 April 2024
Ruang Seminar Gedung A, Kampus Institut Teknologi Adhi Tama Surabaya

Informasi Artikel:
DOI : 10.31284/p.snestik.2024.5871

Prosiding ISSN 2775-5126

Fakultas Teknik Elektro dan Teknologi Informasi-Institut Teknologi Adhi Tama Surabaya
Gedung A-ITATS, Jl. Arief Rachman Hakim 100 Surabaya 60117 Telp. (031) 5945043
Email : snestik@itats.ac.id

Multi-Threading Reverse Engineering untuk Membangun Class Diagram

Faris Simorangkir, Dika Rizky Yunianto, Wilda Imama Sabila

Politeknik Negeri Malang
e-mail: muhfarishalyo@gmail.com

ABSTRACT

In the realm of software development, inadequate documentation often poses a significant challenge, particularly in projects involving multiple developers. Previous research indicates that insufficient documentation hinders source code comprehension and may disrupt efficiency and collaboration. This paper proposes the utilization of reverse engineering technique coupled with multi-threading to construct class diagrams from existing source code. The method is tested on three applications, revealing that increasing the number of threads significantly impacts extraction time and application performance. Overall, augmenting the thread count can enhance application performance, especially for parallelizable applications, but adding more than 50 threads may incur overhead and diminish performance. Thus, this technique holds promise for improving efficiency and understanding of poorly-documented software.

Keywords : *Multi-threads, class diagram, reverse engineering, documentation.*

ABSTRAK

Dalam pengembangan perangkat lunak, kurangnya dokumentasi seringkali menjadi hambatan, terutama dalam proyek-proyek aplikasi yang melibatkan banyak pengembang. Penelitian sebelumnya menunjukkan bahwa kurangnya dokumentasi mempersulit pemahaman kode sumber dan berpotensi mengganggu efisiensi dan kolaborasi. Artikel ini mengusulkan penggunaan teknik reverse engineering dengan memanfaatkan multi-threading untuk membangun diagram kelas dari kode sumber yang ada. Metode ini diuji pada tiga aplikasi dengan hasil menunjukkan bahwa peningkatan jumlah threads berdampak signifikan pada waktu ekstraksi dan performa aplikasi. Kesimpulannya, peningkatan jumlah threads dapat meningkatkan kinerja aplikasi, terutama pada aplikasi yang dapat diparalelkan, namun penambahan lebih dari 50 threads dapat menyebabkan overhead dan penurunan performa. Dengan demikian, teknik ini memiliki potensi untuk

meningkatkan efisiensi dan pemahaman terhadap perangkat lunak yang sudah ada yang seringkali kurang dokumentasi.
Kata kunci: *Multi-Thread,* diagram kelas, rekayasa balik, dokumentasi.

PENDAHULUAN

Dalam konteks pengembangan perangkat lunak yang terus berkembang, proyek-proyek seperti Aplikasi permainan memainkan peran penting dalam mendukung efisiensi dan produktivitas di lingkungan perkantoran. Namun, terlepas dari manfaatnya, proyek aplikasi semacam ini seringkali menghadapi masalah yang sama dengan banyak proyek perangkat lunak lainnya: kekurangan dokumentasi yang memadai, sehingga programmer lebih menyukai membaca *source code* dari pada membaca dokumentasi [1].

Dalam jurnal "Software Engineering for Computational Science:" yang diterbitkan pada tahun 2018, penelitian tersebut menegaskan bahwa kurangnya dokumentasi yang memadai akan menyulitkan pengembang baru dalam membaca dan memahami kode [2]. Pengembang cenderung mengandalkan cara-cara informal dan kolaboratif dalam mentransfer pengetahuan, serta lebih memilih berinteraksi secara langsung dengan pengembang di bagian tertentu untuk mendiskusikan permasalahan terkait kode. Hasil penelitian ini menyoroti pentingnya dokumentasi yang kuat dalam mendukung efisiensi dan kolaborasi dalam pengembangan perangkat lunak.

Dokumentasi pada pengembangan perangkat lunak dapat diterima secara luas bahwa peran dokumentasi memiliki signifikansi yang besar. Dokumentasi berperan sebagai alat komunikasi yang krusial dalam menyampaikan persyaratan sistem, rancangan, implementasi, dan informasi pemeliharaan antara berbagai pemangku kepentingan dan anggota tim pengembang [3]. Dengan adanya dokumentasi yang tepat dan komprehensif, pemahaman yang lebih baik tercipta di antara mereka, memfasilitasi kolaborasi yang efektif, serta mengurangi kemungkinan kesalahpahaman.

Penyebab utama dari kekurangan dokumentasi ini adalah karena sistem yang menjadi objek penelitian telah dikerjakan oleh beberapa programmer yang berbeda dalam periode waktu yang berbeda pula. Dalam jurnal "Software Engineering for Computational Science:" yang diterbitkan pada tahun 2018, penelitian ini menunjukkan bahwa tidak adanya dokumentasi akan menyulitkan pengembang baru dalam membaca dan memahami kode.Pengembang cenderung bergantung dengan cara-cara informal dan kolaboratif dalam mentransfer pengetahuan, serta lebih memilih menghubungi pengembang pada bagian tertentu dan berdiskusi dengan mereka [2].

Jurnal yang diterbitkan pada tahun 2017 dengan judul "Do UML object diagrams affect design comprehensibility? Results from a family of four controlled experiments" menjelaskan bahwa penggunaan diagram objek UML dapat meningkatkan pemahaman desain perangkat lunak, namun hanya untuk pengguna tertentu dengan pengalaman dan pengetahuan UML yang tepat [4].

Dalam konteks pembangunan dokumentasi, teknik *reverse engineering* menjadi pendekatan yang efektif. Pendekatan ini melibatkan proses analisis dan ekstraksi informasi dari perangkat lunak yang sudah ada dengan cara mengekstrak arsitektur dan desain *source code* [5]. Dengan demikian, teknik *reverse engineering* bukan hanya berperan sebagai alat untuk mengatasi permasalahan dokumentasi, melainkan juga sebagai metode esensial yang membantu meningkatkan pemahaman dan pemeliharaan perangkat lunak, sehingga berkontribusi pada pengembangan perangkat lunak yang lebih handal dan berkelanjutan.

Reverse Engineering merupakan suatu teknik analisis untuk memahami secara komprehensif cara kerja sistem kompleks yang mungkin tertutup atau tidak terdokumentasi [6]. *Reverse Engineering* memungkinkan analisis dan pemahaman yang mendalam terhadap logika dan mekanisme internal suatu sistem atau produk yang sebelumnya tidak jelas. Proses analisis dilakukan terhadap desain, komponen, dan fungsionalitas suatu sistem atau produk yang sudah ada, dengan niat untuk mengekstrak pengetahuan dan informasi dari suatu aplikasi.

Multi-threading adalah suatu teknik yang memungkinkan beberapa thread eksekusi dijalankan secara bersamaan dalam satu program [7]. Thread ini merupakan aliran instruksi independen yang dapat dijadwalkan untuk berjalan secara paralel dengan thread lainnya. Tujuan dari implementasi multi-threading adalah untuk meningkatkan kinerja dan daya tanggap program. manfaat dari penerapan multi-threading mencakup respons yang lebih cepat, *throughput* yang lebih tinggi, dan pemanfaatan sumber daya perangkat keras yang lebih efisien. Oleh karena itu, teknik ini menjadi sangat berharga untuk berbagai aplikasi, termasuk desktop, server, dan aplikasi tertanam, dengan tujuan mencapai tingkat paralelisme yang optimal [8].

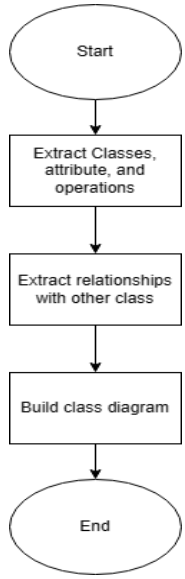
Oleh karena itu, pada jurnal ini diusulkan *reverse engineering* untuk membangun class diagram menggunakan multi-threading. Diharapkan *reverse engineering* dapat mengatasi tantangan dalam memahami perangkat lunak yang sudah ada, yang sering kali tidak memiliki dokumentasi yang memadai. Dengan pemahaman yang lebih baik tentang struktur perangkat lunak, pengembang dapat menghindari risiko kesalahan dalam perancangan sistem yang dapat mengganggu efisiensi dan keberlanjutan sistem perangkat lunak [9].Pentingnya *reverse engineerin*g dalam menggambarkan *class diagram* memerlukan proses yang cukup kompleks, sehingga penggunaan multi-threading dapat menjadi salah satu teknik untuk menyederhanakan pemrosesan yang rumit tersebut

METODE

Diagram kelas adalah alat yang penting bagi pengembang perangkat lunak, karena diagram ini memberikan representasi visual dari sistem yang sedang dikembangkan yang dapat digunakan untuk berkomunikasi dengan orang lain dalam tim dan untuk mengidentifikasi masalah sebelum terjadi. Diagram ini juga dapat digunakan untuk menguji dan memvalidasi desain sistem dan memastikan bahwa sistem tersebut memenuhi persyaratan para pemangku kepentingan sistem [10].

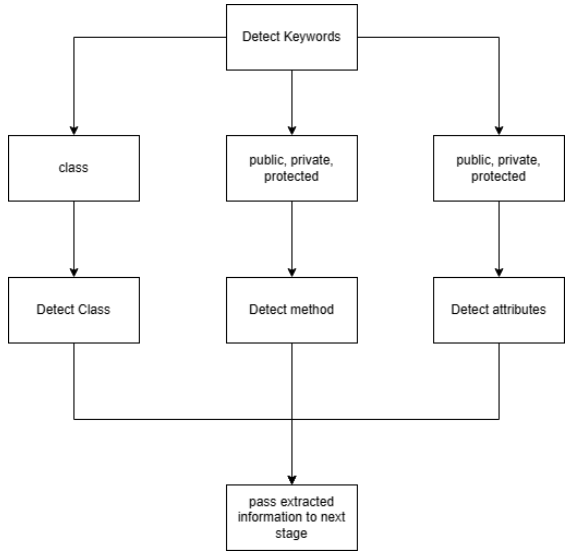
Reverse engineering adalah proses untuk mengevaluasi desain yang telah ada. Dalam pengembangan aplikasi yang konvensional, tahapan umumnya mengikuti Model Pengembangan Perangkat Lunak (SDLC), yang terdiri dari tahapan analisis kebutuhan (*requirement*), perancangan (*design*), implementasi, pengujian (*testing*), pengiriman (*delivery*), namun tidak dapat disangkal bahwa banyak pengembang tidak mengikuti SDLC secara ketat dalam proses pengembangan mereka. Oleh karena itu, *reverse engineering* digunakan untuk membantu pengembang dalam memeriksa kembali kode sumber yang terbentuk akibat tidak mengikuti SDLC secara konvensional.

Pengolahan data pada *source code Controller* untuk mendapatkan *class diagram* dilakukan dengan menggunakan metode Multi-Threading. Proses pengolahan data dilakukan dengan tiga tahapan, yaitu Ekstraksi relasi dengan kelas lain, Ekstraksi kelas, atribut, dan operasi, serta pembuatan class diagram. Alur tersebut dapat dilihat pada Gambar 1.



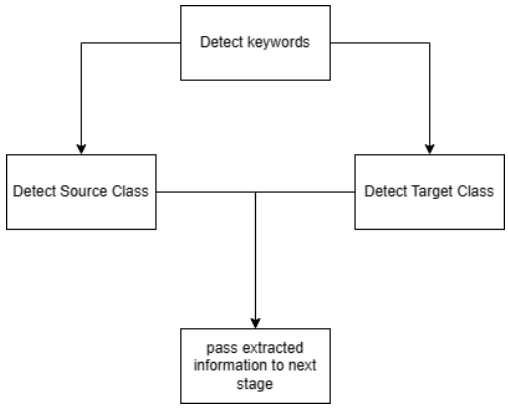
Gambar 1. Alur proses reverse engineering

Setelah proses ekstraksi untuk menentukan relasi antar kelas maka proses selanjutnya adalah melakukan ekstraksi untuk menentukan atribut dan operasi pada kelas. Proses ini ditujukan untuk mengetahui atribut yang terdapat pada kelas. Proses ekstraksi tersebut dapat dilihat pada Gambar 2.



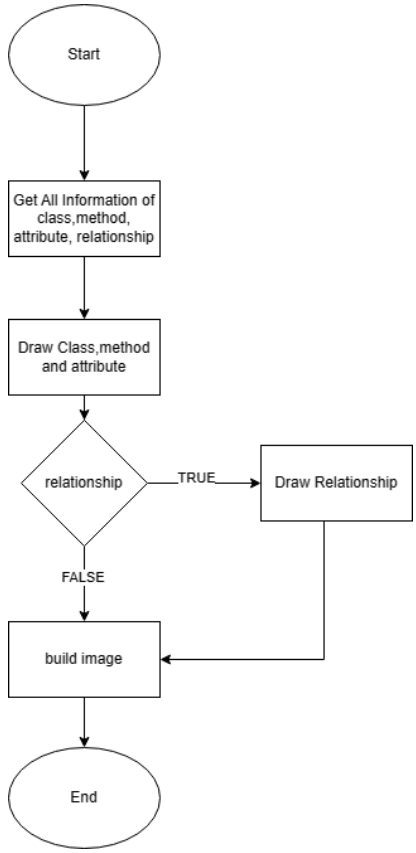
Gambar 2. Proses ekstraksi

Data yang telah dipilih akan dilakukan ekstraksi hubungan dengan kelas yang lain. Tahap ini bertujuan untuk mendapatkan informasi terlebih dahulu apakah kelas yang akan dilakukan ekstraksi memiliki hubungan dengan kelas lain. Proses tersebut dapat dilihat pada Gambar 3.



Gambar 3. Proses ekstraksi relasi

Tahap terakhir adalah membangun kelas diagram yang datanya dihasilkan dari proses ekstraksi sebelumnya. Data yang didapat pada ekstraksi sebelumnya berguna untuk mengetahui nama kelas beserta atribut yang dimilikinya. Setelah melalui proses pembangunan diagram kelas, hasil akhirnya biasanya berupa gambar diagram. Format umum yang digunakan untuk menyimpan gambar diagram ini adalah dalam bentuk file JPEG atau PNG. Gambar tersebut memberikan representasi visual yang jelas dan mudah dipahami mengenai struktur kelas dalam sistem, serta bagaimana kelas-kelas tersebut saling berinteraksi. Proses tersebut dapat dilihat pada Gambar 4.



Gambar 4. Proses penggambaran class diagram

HASIL DAN PEMBAHASAN

Data yang digunakan untuk melakukan pengujian memiliki detail seperti yang tercantum pada Tabel 1, yang mencakup ukuran aplikasi dan jumlah kelas.

Tabel 1. Data pengujian

	Aplikasi 1	**Aplikasi 2**	**Aplikasi 3**
Jumlah File	36	9	12
Total Size (KB)	301	37.6	35.5

Pengujian dilakukan dengan penggunaan threads yang berbeda untuk setiap aplikasi aplikasi yang diujikan akan diinputkan ke dalam sistem *reverse engineering* yang akan dibangun menggunakan bahasa pemrograman C#, dalam sistem *reverse engineering* yang telah dibangun telah disediakan pilihan penggunaan *multi-thread* untuk proses pembentukan class diagram. Sistem *reverse enginee*ring yang dibangun dapat dilihat pada gambar. Data hasil pengujian tersebut terdapat pada Tabel 2. Contoh hasil *class diagram* dapat dilihat pada Gambar 5.

Seperti yang terlihat dari hasil yang disajikan, waktu yang dibutuhkan, untuk melakukan ekstraksi kode sumber dipengaruhi oleh jumlah *threads* terhadap perform ketiga aplikasi yang diuji. Analisis performa aplikasi dalam hubungannya dengan jumlah threads menunjukkan adanya pengaruh yang signifikan pada tiga aplikasi yang diuji, yakni Aplikasi 1, Aplikasi 2, Aplikasi 3.

Pada aplikasi 1, teramati bahwa waktu pengolahan (runtime) rata-rata mengalami penurunan yang signifikan seiring dengan peningkatan jumlah threads dari 1 hingga 50. Penurunan runtime yang paling mencolok terjadi pada 3 threads awal, dengan runtime menurun dari 4.89 detik menjadi 1.32 detik. Meskipun terjadi peningkatan jumlah threads, runtime kembali naik sedikit pada 100 threads, kemungkinan disebabkan oleh overhead yang timbul dari pengelolaan thread yang semakin banyak. Namun, pada 150 threads, terjadi penurunan runtime kembali menjadi 0.11 detik.

Tabel 2. Hasil Pengujian

Jumlah Threads	**Waktu (s)**		
	Aplikasi 1	**Aplikasi 2**	**Aplikasi 3**
1	4.89	0.93	1.42
3	1.32	0.33	0.44
5	0.87	0.23	0.33
7	0.66	**0.10**	0.22
9	0.44	0.12	0.22
50	**0.11**	0.11	**0.10**
100	0.16	0.11	0.11
150	0.11	0.11	0.11

Pada aplikasi 1, terlihat bahwa semakin banyak *thread* yang digunakan, waktu eksekusi cenderung semakin cepat. Namun, terdapat titik optimal di mana penambahan jumlah *thread* tidak lagi memberikan peningkatan signifikan dalam waktu eksekusi. Hal ini dapat dilihat pada uji coba dengan 50 thread pada aplikasi 1. Fenomena serupa juga terjadi pada aplikasi 2 dan aplikasi 3.

Selain itu, jumlah *file* yang menggambarkan kelas juga mempengaruhi performa waktu. Semakin sedikit jumlah *file*, waktu eksekusi cenderung semakin singkat. Begitu pula dengan ukuran file, semakin besar ukuran *file*, waktu yang diperlukan juga semakin lama.

Dari percobaan di atas, terlihat bahwa rata-rata waktu maksimum yang dihasilkan dari multi-threading sebesar 0.1 detik. Secara keseluruhan, peningkatan jumlah *threads* secara umum dapat meningkatkan performa aplikasi, terutama pada aplikasi dengan beban kerja yang dapat diparalelkan. Penambahan hingga 50 threads memberikan hasil optimal, sementara peningkatan lebih lanjut dapat menghasilkan *overhead* yang mengakibatkan penurunan performa.

Gambar 5. Contoh hasil class diagram reverse engineering

KESIMPULAN

Berdasarkan hasil penelitian yang disajikan, disimpulkan bahwa waktu ekstraksi kode sumber dipengaruhi oleh jumlah *thread* dan berdampak pada aplikasi yang diuji. Penambahan hingga 50 *thread* memberikan hasil optimal, sementara peningkatan lebih lanjut dapat menyebabkan *overhead* yang mengakibatkan penurunan performa.

DAFTAR PUSTAKA

[1] E. Aghajani *et al.*, "Software Documentation Issues Unveiled," in *Proceedings - International Conference on Software Engineering*, IEEE Computer Society, May 2019, pp. 1199–1210. doi: 10.1109/ICSE.2019.00122.

[2] A. N. Johanson and W. Hasselbring, "Software Engineering for Computational Science:," *Comput Sci Eng*, vol. 20, no. 2, pp. 90–109, Mar. 2018, doi: 10.1109/MCSE.2018.108162940.

[3] P. Mäder and A. Egyed, "Gesellschaft für Informatik," 2016.

[4] M. Torchiano, G. Scanniello, F. Ricca, G. Reggio, and M. Leotta, "Do UML object diagrams affect design comprehensibility? Results from a family of four controlled experiments," *J Vis Lang Comput*, vol. 41, pp. 10–21, Aug. 2017, doi: 10.1016/j.jvlc.2017.06.002.

[5] A. Alalawi and M. Hammed, "Reverse Engineering Approach for Classes' Representations and Interactions in Software Projects," 2020.

[6] M. K. Sarkar and T. Chaterjee, "Reverse Engineering: An Analysis of Dynamic Behavior of Object Oriented Programs by Extracting UML Interaction Diagram," 2013. [Online]. Available: www.ijcta.com

[7] S. Khalid, R. Ibrahim, and H. Onn Malaysia, "GENERATING UML CLASS DIAGRAM FROM SOURCE CODES USING MULTI-THREADING TECHNIQUE," vol. 11, no. 18, 2016, [Online]. Available: www.arpnjournals.com

[8] S. M. S. S. V Chandra, "Multi-thread sequencing," 2019.

[9] M. J. Decker, K. Swartz, M. L. Collard, and J. I. Maletic, "A tool for efficiently reverse engineering accurate UML class diagrams," in *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, Institute of Electrical and Electronics Engineers Inc., Jan. 2017, pp. 607–609. doi: 10.1109/ICSME.2016.37.

[10] F. Chen, L. Zhang, X. Lian, and N. Niu, "Automatically recognizing the semantic elements from UML class diagram images," *Journal of Systems and Software*, vol. 193, Nov. 2022, doi: 10.1016/j.jss.2022.111431.