Studi Terhadap Reverse Engineering Goal Models from Legacy Code

Jan Peter Alexander, 0906503805

Fakultas Ilmu Komputer, Universitas Indonesia
Indonesia
jp@ui.ac.id

***Abstrak*—Metodologi Reverse Engineering dari kode legacy memliki banyak tantangan . Dari segi aplikasi, kode sumber yang tak berdokumen dan aplikasi yang tak memiliki kode sumber memerlukan penanganan ekstra agar dapat diolah. Paradigma pemrograman sistem legacy dengan saat ini juga menjadi gap yang harus diatasi. Ada bagian-bagian yang terhilang sehingga goal model tidak lengkap bahkan ada goal yang saling konflik.**

I. Pendahuluan

Yu et. al. mengembangkan sebuah metodologi untuk melakukan *reverse engineering* dari sebuah kode *legacy* untuk mendapatkan kembali *goal models* dari *stakeholders*. Metodologi ini terinspirasi dari GSP framework. GSP menghasilkan *requirements* dengan memodelkan dan menganalisis *user **g**oals*, ***s**kill*, dan ***p**references*. Tujuan dari metodologi ini adalah menghasilkan produk baru yang telah terfaktorisasi.

Proses dalam metodologi tersebut digambarkan seperti tapal kuda, setengah proses pertama yang menanjak adalah proses *reverse engineering*, sedangkan setengah proses selanjutnya menggunakan *forward engineering*. Ada pun langkah-langkahnya: 1) *Refactoring* dengan mengekstrak method-method dari komentar yang ada; 2) Mengubah kode-kode tersebut menjadi bentuk abstrak melalui *refactoring* diagram *state* dan membangun diagram Hammock; 3) Ekstrak goal model dari pohon struktur program yang abstrak; 4) Mengidentifikasi non-functional requirements dan menurunkan softgoals berdasarkan traceability antara kode sumber dan *goal model*.

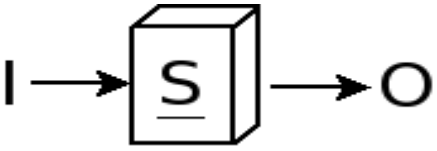
Sebagai bahan percobaan digunakan dua aplikasi perangkat lunak bebas, yakni SquirrelMail dan Columba. SquirrelMail adalah sebuah aplikasi webmail client yang ditulis dengan bahasa pemrograman PHP. Columba adalah sebuah aplikasi mail client yang dibuat dengan menggunakan Java. Untuk dapat mengolah keduanya, metodologi ini mengembangkan perkakas dengan menggunakan teknologi Eclipse.

II. Identifikasi Kode untuk *Refactoring*

Proses refactoring yang digunakan dalam [1] menggunakan komentar-komentar yang sudah ada dalam aplikasi. Hal ini menyebabkan adanya kondisi yang membuat metodologi ini tidak dapat digunakan, yakni: 1) Aplikasi tanpa kode sumber; 2) Aplikasi dengan kode sumber yang tidak berkomentar.

Untuk dapat menghasilkan *refactored representation* dari kode legacy, diperlukan langkah-langkah lebih lanjut. Proses dari kode legacy ke kode yang telah refactored memiliki karakteristik unik, yakni tanpa mengorbankan fungsionalitas. Proses refactoring berusaha menjadikan kode method lama S ditransformasikan menjadi sebuah method baru S'. Method S dan S' memiliki karakter yang sama: memiliki himpunan input dan himpunan output yang sama. Dengan ini dapat dikatakan bahwa S dan S' adalah sebuah sebuah *black box* yang sama. Oleh sebab itu, dapat digunakan sebuah metode black box untuk mendapatkan kode *refactored*.

Metode ini dilakukan dengan menganggap method-method yang terlihat oleh pengguna sebagai sebuah *black box* seperti Gambar 1. Metode ini berusaha mengenali himpunan *input* dan himpunan *output* yang mungkin ada pada sebuah fungsionalitas. Dari proses identifikasi ini, kemudian ditentukan sebuah method yang sudah terfaktorisasi. Proses identifikasi ini dapat menggunakan metode yang biasa digunakan dalam *unit testing*.



Gambar 1: Proses Input Output

Untuk dapat mengidentifikasi himpunan input dan himpunan output, diperlukan pendataan method yang ada. Pada aplikasi GUI, hal ini dapat dimulai dengan mendata fungsi-fungsi yang paling dekat yang membutuhkan interaksi manusia, misalnya menu, tombol, dan sebagainya. Pada aplikasi CLI, fungsionalitas dapat dikenali pada bagian-bagian program yang membutuhkan *input*.

Kode sumber yang tak berdokumentasi sebenarnya memiliki solusi. Platform Eclipse yang digunakan sebagai perkakas menyediakan fungsi pendataan method-method yang tersedia. Dengan memanfaatkan fungsi pada platform ini, setiap method dapat diteliti dan dikenali fungsionalitasnya.

Setelah mengenali method-method, langkah selanjutnya yang diperlukan adalah proses *refinement*. Proses Extract Method yang dilakukan dalam metodologi ini menggabungkan beberapa method menjadi satu. Dengan demikian, metode *black box* yang ditambahkan dapat lebih dibersihkan.

Terkadang, ada fungsionalitas yang tak terdokumentasi yang masuk dalam sebuah aplikasi. Fungsionalitas tak terdokumentasi itu biasanya sebuah kode tambal sulam yang berisi workaround dan improvisasi dari programmer. Sayangnya metodologi ini tidak memperhatikan hal tersebut.

Padahal, kode-kode ini bisa jadi sebuah momok dalam proses *decoupling* karena kode itu menyebabkan ketergantungan antar satu method dengan method yang lain. Hal ini bisa menghambat proses *refactoring*.

Selain kode tambal sulam, perlu diperhatikan perbedaan konstrain pengembangan sistem. Sistem legacy hidup pada masa di mana sumber daya tidak sebanyak sekarang. Algoritma-algoritma yang dahulu dianggap tidak efisien sekarang telah kembali dipelajari. Perkembangan keilmuan dan perangkat keras membuat beberapa paradigma berubah.

Perbedaan paradigma ini mengakibatkan pendekatan yang berbeda. Saat ini orang cenderung menggunakan OOP dibandingkan prosedural. Dengan perbedaan paradigma pemrograman, sebuah obyektif yang hendak dicapai bisa jadi berbeda penerapannya saat ini dibandingkan dengan saat dulu.

Selain menggunakan *Extract Method* dan *Extract States and Transitions*, metodologi ini perlu memperhatikan hal perbedaan paradigma ini dengan menyediakan sebuah langkah transformasi. Transformasi yang dimaksudkan tidak mengubah fungsionalitas, tetapi lebih cenderung kepada penyesuaian dengan pola pemrograman saat ini. Proses transformasi ini bisa menjawab masalah S*tatechart structuring* pada pemrograman antik seperti Fortran.

III. Identifikasi Goal untuk Requirement

Reverse Engineering ibarat seperti mendulang emas. Setelah mendapatkan bongkahan emas, pemurnian lebih lanjut dibutuhkan untuk memisahkan kandungan emas dari kandungan logam, kendati logam lain itu juga berharga. Logam tersebut belum tentu dibuang, tetapi bisa jadi diproses di tempat lain.

Goal-goal yang telah terbentuk juga perlu diklasifikasikan. Namun, berbeda dengan pendekatan forward engineering, goal yang tidak memiliki nilai yang tinggi tidak seharusnya dibuang. Bagaimana pun juga, restorasi requirement akan ada bagian-bagian yang terhilang dari maksud mula-mula. Oleh sebab itu, pendataan seluruh goal diperlukan untuk melihat kembali apa bila ada penemuan baru.

Pada sistem yang kompleks bisa jadi goal-goal yang telah teridentifikasi ada yang bertentangan. Hal ini bisa terjadi akibat adanya informasi yang hilang dari niat mula-mula. Oleh sebab itu, selain penggunaan *soft goals*, metodologi ini bisa diperkaya dengan memperkenalkan metode identifikasi *conflicting goals*.

Dengan mengenali goal-goal yang bertentangan, didapatkan suatu gambaran bagaimana goal-goal ini harus ditata (refined) lagi sebelum menjadi requirement. Dengan dikenalinya conflicting goals, metodologi ini bisa menghasilkan goal lain yang bisa menutupi konflik tersebut. Solusi lain dengan membuat goal baru yang merupakan jalan tengah dari goal-goal yang saling konflik. Pada akhirnya, proses ini memperlengkapi goal model menjadi sebuah solusi yang komprehensif dan bersih.

IV. Kesimpulan

Produk akhir dari reverse engineering menghasilkan goal model yang cukup lengkap dan bersih. Suatu goal model yang bersih memiliki level decoupling yang tinggi. Dengan mudahnya proses decoupling, forward engineering dapat mendisain arsitektur yang lebih efisien berdasarkan komponen-komponen.

Karena bersifat modular, maka setiap komponen tersebut dapat ditingkatkan. Fungsionalitas baru dapat diperkenalkan dengan baik. Sehingga, pada akhirnya sistem yang baru dapat menjadi sebuah sistem yang lebih baik dari sistem *legacy*.

References

[1] Yu, Y., Wang, Y., Mylopoulos, J., Liaskos, S., Lapouchnian, A., and Prado Leite, J. C. 2005. Reverse Engineering Goal Models from Legacy Code. In Proceedings of the 13th IEEE international Conference on Requirements Engineering (August 29 - September 02, 2005). RE. IEEE Computer Society, Washington, DC, 363-372. DOI= http://dx.doi.org/10.1109/RE.2005.61