Manuscript Submitted	: 11 Mac 2017
Accepted	: 3 April 2017
Published	: 22 June 2017

Penggunaan Graf Aliran Pergantungan Untuk Persembahan Program Berorientasikan Objek

Syarbaini Ahmad, Che Wan Shamsul Che Wan Ahmad,
Syakirah Mohd Sofi & Helyawati Baharudin
Kolej Universiti Islam Antarabangsa Selangor (KUIS), 43000 Kajang Selangor Malaysia
syarbaini, cwshamsul, syakirah, helyawati@kuis.edu.my

Abstrak

Pengaturcaraan berorientasikan objek digunakan secara meluas dalam kaedah pengaturcaraan pada hari ini. Ini kerana kaedah ini menjadikan proses pembangunan perisian lebih mudah untuk dilaksanakan dan lebih berkesan. Namun begitu, ciri-ciri seperti pergantungan, pewarisan, polimorfisma dan ikatan dinamik yang ada dalam pengaturcaraan berorientasikan objek menyumbang kepada berlakunya kerumitan dalam kerja-kerja analisa program untuk tujuan debugging, pengujian dan penyelenggaraan perisian. Artikel ini mencadangkan penggunaan graf aliran pergantungan sebagai kaedah menganalisis program berorientasikan objek yang meggabungkan antara graf aliran kawalan dan graf pergantungan di dalam satu graf yang dinamakan graf aliran pergantungan. Ujian ke atas kod telah dibuat untuk menilai keberkesanan graf yang digunakan ini kepada sepuluh kod yang dibangunkan dengan menggunakan kaedah berorientasikan objek. Hasilnya didapati ianya mampu menjana graf dengan tepat dan boleh berfungsi dengan baik. Untuk memantapkan lagi hasil kajian, teknik yang dicadangkan ini diaplikasi dalam bentuk perisian dan diuji kepada dua puluh buah syarikat pembangun perisian. hasil yang diperolehi menunjukkan ianya membantu dalam proses penyelenggaraan menjadi lebih efektif.

Kata carian *:persembahan, aliran, graf aliran kawalan, graf pergantungan, dependence flow graph*

Abstract

Object-oriented programming is widely used in the method of programming today. It is because of it make the software development process easier and more effective in practice. However, the features such as dependencies, inheritances, polymorphisms and dynamic bindings are contribute to the occurrence of the complexity in the program for the purpose of analysis, debugging, testing and maintenance of software. This article proposes the use of dependence flow graph as one of the technique to analyst the object-oriented program. It is a hybrid of the control flow graph and dependence graph into a single graph called dependence flow graph. The experiment has been made in order to evaluate the effectiveness of the graph onto ten JAVA code developed using object-oriented methods. As a result it is able to generate graphs accurately and successfully. To strengthen the results of the study, the proposed technique is applied to twenty software development companies. The results showed that it helps in the maintenance activity and becoming more effective.

1. Pengenalan

Kaedah pengaturcaraan berorientasikan objek (OO) sangat di perlukan dalam bidang sains komputer. Kaedah ini menjadikan bidang sains komputer dan teknologi maklumat berkembang dengan pesat dengan wujudnya berbagai sistem yang dibangunkan. Pada masa ini OO di ajar secara meluas sebagai sebagai kurikulum wajib dalam bidang asas pengaturcaraan pendidikan sains komputer. Dalam kajian Dale [1] [2] menunjukkan bahawa 65% daripada institut pengajian tinggi menjadikan pengaturcaraan berorientasi objek sebagai sebahagian daripada modul wajib dalam pendidikan sains komputer.

OO adalah kaedah pengaturcaraan yang merealisasikan sandaran kepada objek dalam mewujudkan modul-modul sistem, bukannya tingkah laku dalam usaha untuk menjalankan tindakan. Berbanding kaedah prosedur yang sebelumnya, kaedah OO mengenalpasti masalah dalam sistem pengurusan rutin dan cuba untuk menyelesaikannya. ia kemudian, dimuatkan beberapa jenis data yang telah ditetapkan:

integer, terapung, *Strings*, dan *array*. Penekanan tumpuan penyelesaian adalah pada objek yang ingin diubah atau sedang dibangunkan itu sendiri bukan pada pelaksanaan tugas-tugas yang bertindak bagi objek. Dalam perkataan yang mudah, struktur pengaturcaraan berorientasikan objek tidak menganggap keputusan yang dibuat mengenai kaedah untuk menggunakan logik, tetapi kepada definisi data yang akan digunakan dalam pengaturcaraan.

Dalam bidang kejuruteraan perisian, teknologi OO sememangnya digunakan secara meluas dalam membangunkan sistem seawal peringkat perancangan sistem hingga kepada penyelenggaraannya. Ia juga termasuk konsep dan ciri-ciri yang membuat penciptaan dan penggunaan objek lebih mudah dan lebih fleksibel [3]. Sepertimana lumrah yang berlaku pada setiap ciptaan manusia, perlu diakui bahawa tiada yang sempurna, bahkan sentiasa ada sahaja yang perlukan nilai tambah untuk menjadikan ianya sesuai dengan keperluan semasa. Dalam kejuruteraan perisian, sama ada proses, bahasa pengaturcaraan, teknik, alat atau platform yang baik perlu disesuaikan dengan setiap situasi. Ini menjadikan ianya masalah yang besar kaedah terutama dalam fasa penyelenggaraan. Pusat Penyelidikan Xerox Palo Alto telah menentukan bahawa terdapat banyak masalah pengaturcaraan terutama yang berkaitan dengan penggunaan kaedah OO. Semakin besar saiz sistem yang ingin dibangunkan, semakin rumit sistem tersebut dan semakin sukar untuk kerja-kerja penyelenggaraan dilaksanakan. Bahkan ada yang berpandangan yang kaedah pengaturcaraan OO tidak relevan jika di lihat dari aspek tersebut [4] [5]. Permasalahan yang dihadapi ini juga turut terkena pada aktiviti analisa sistem sama ada untuk tujuan penyelenggaraan, pengujian, kejuruteraan terbalik atau penyusunan.

Kaedah analisis kod yang biasa digunakan dalam pembangunan perisian hari ini adalah kebiasaannya menggunakan graf aliran pergantungan dan graf aliran kawalan yang menjadi antara komponen perisian dalam struktur program. Ini adalah salah satu aktiviti asas yang digunakan oleh penyelenggara untuk mengenal pasti pelbagai hubungan antara unsur-unsur program [6] [7] [8]. Penyelenggaraan program berorientasikan objek boleh menimbulkan masalah jika ia tidak dilakukan secara sistematik. Ciri-ciri utama teknik berorientasikan objek seperti polimorfisma, pewarisan, pengkapsulan dan dinamik mengikat menjadi sebab-sebab utama berlakunya banyak masalah penyelenggaraan [9].

Kebanyakan pembangun perisian hari ini, jika tiada cara yang lebih baik, akan menganalisis kod dengan menggunakan kaedah graf. Graf yang biasa digunakan pula kebiasaannya adalah, graf panggilan, graf kawalan atau graf pergantungan. Jika sistem yang dibangunkan itu direka dengan baik dan dibina, kod mungkin memerlukan perubahan dalam hanya beberapa tempat. Jika sistem adalah berorientasikan objek, kita mungkin membina kelas yang dibuat dalam bentuk blok-blok metod yang sesuai dan menghubungkannya dengan blok-blok metod yang lain sama ada dalam kelas yang sama atau kelas yang berlainan. keadaan ini menimbulkan kesukaran pada mana-mana juru analisa sistem untuk menganalisa sistem diperingkat penyelenggaraan. Kita mungkin memerlukan hirarki kelas untuk mengendalikan fail yang berbeza dan pangkalan data yang banyak. Untuk memahami perkaitan antara setiap nod kelas, metod dan pernyataan dalam program yang berorientasikan objek bukanlah suatu perkara yang remeh. Bahkan ia tidak akan menjadi mudah untuk memastikan bahawa kita akan membuat analisa pada kod-kod yang penting. Walaupun kita mempunyai kaedah pengaturcaraan berorientasikan objek yang baik dalam pembangunan perisian, namun masalah yang sama tetap berlaku [12]. Sistem yang dibangunkan dengan menggunakan kaedah berorientasikan objek biasanya menghasilkan kelas yang sukar untuk berubah, dan kod yang tidak boleh diguna semula dan sukar untuk dikesan kerana masalah perkaitan antara satu kod dengan kod yang lain yang banyak serta ikatan dinamik yang berlaku dalam bentuk yang rumit untuk difahami perkaitannya.

Kertas kerja ini menghuraikan tentang pembangunan graf aliran pergantungan secara teknikal. Ianya adalah berdasarkan latar belakang masalah yang ditemui dalam menggunakan program berasaskan OO. Kandungan perbincangannya adalah seperti, Metodologi kajian yang menghuraikan penggambungan antara graf aliran kawalan dengan graf pergantungan data. Seksyen 3 adalah proses

pengujian yang dibuat dan hasil yang diperolehi. Kemudian terdapat sedikit kajian literatur dan lapangan. Akhirnya kesimpulan yang diperolehi dari kajian yang dibuat.

2. Kajian Literatur

Ray et al.[17] memperkenalkan AOSG (Sistem Aspek Kebergantungan Graph). AOSG merupakan graf yang dibangunkan menggunakan graf pergantungan. Ia digunakan untuk mengira penghirisan dinamik program berorientasikan aspek dengan menandakan dan menyahtandakan tepi graf semasa perlaksanaan. Algoritma yang dicadangkan mengambil lebih banyak masa dalam penandaan dan menyahtandakan graf tersebut secara langsung. Namun begitu kaedah ini lebih sesuai digunakan untuk program kaedah aspek sahaja.

Tamrawi et. al. [18] membangunkan *SYMake* iaitu infrastruktur dan alat-alat untuk analisis kod bina dalam GNU. *SYMake* adalah bahasa skrip di mana fail dibina (dipanggil *Makefile*) digunakan untuk menentukan kebergantungan binaan antara fail konfigurasi dalam projek melalui entiti program *SYMake* . *SYMake* menyediakan algoritma simbolik penilaian yang memproses *Makefile* dan menghasilkan satu simbolik graf pergantungan yang mewakili kebergantungan bina (iaitu peraturan) di antara fail melalui arahan. Semasa penilaian simbolik, setiap nilai rentetan mewakili sebahagian jika nama fail atau arahan dalam peraturan, *SYMake* menyediakan juga graf *acyclic* (dipanggil-T model) untuk mewakili penilaian kesan simbolik. Digunakan *SYMake* untuk membangunkan algorithma dan alat untuk mengesan beberapa jenis kod yang silap dalam *Makefiles.* Ia juga menyokong kod untuk membina proses pemfaktoran semula. Penilaian empirikal untuk penamaan semula *SYMake* pada beberapa sistem dunia sebenar menunjukkan ketepatan yang tinggi dalam entiti penamaan semula.

Isong [13] mencadangkan penggunaan graf pergantungan dalam perwakilan perantaraan yang dinamakan sebagai *OO Component Dependency Networks* (OOComDN) yang jelas mewakili perisian dan membolehkan struktur yang rumit diukur menggunakan rangkaian yang kompleks. Objektifnya adalah untuk meningkatkan statik Analisa Impak Perubahan (Change Impact Analysis - CIA) dan memudahkan pemahaman aturcara. Eksperimen terkawal telah dijalankan untuk menilai keberkesanannya dengan menggunakan projek pelajar dalam tempoh penyelenggaraan dan pembetulan. Hasil yang diperoleh menunjukkan bahawa OOComDN adalah praktikal untuk analisis impak tersebut. Tidak seperti graf kebergantungan yang lain, OOComDN adalah mudah dan tidak begitu kompleks serta melibatkan komponen yang kecil sahaja. Fokus pengguna OOComDN adalah untuk pelajar pada peringkat permulaan dan sesuai untuk membuat analisa program yang berskala kecil dan sederhana.

Kanade [14] memperkenalkan aturcara pelanggan yang menerima struktur data sebagai input yang mempunyai perwakilan pergantungan jika kelakuannya berbeza untuk nilai-nilai input yang sama secara logik. Kanade mencadangkan kaedah dan alat untuk ujian automatik pada klien struktur data bagi perwakilan pergantungan. Penyelesaiannya adalah bergantung kepada sintesis secara automatik fungsi songsangan f: satu kepada banyak. Jika diberi nilai input x, kita boleh menjana pelbagai input ujian kesetaraan logik x dengan melaksanakan songsangan dengan nilai berkanun f(x) sebagai input berulang kali. Kajian ini menghasilkan algoritma penyongsangan untuk kelas terhad bagi program pemulihan, termasuklah program pemetaan tatasusunan untuk tatasusunan secara lelaran biasa. Dalam kajian ini, pelanggan diuji dengan berkesan menggunakan pendekatan ini. Namun, pendekatan yang dikemukakan mempunyai beberapa batasan, iaitu tidak tepat untuk operasi *join* dan sintaksis terhad ke atas program input.

Duboscq et. al [15] mencadangkan pengoptimuman spekulatif (Speculative optimizations) digunakan untuk meminimumkan fungsi penyusunan dalam pengkompil dengan menggunakan kaedah perwakilan perantaraan (IR). Ia adalah peta yang dioptimumkan berdasarkan graf dan jurubahasa. Ia digunakan untuk bergerak dan memasukkan nod pengawal tanpa sebarang kekangan dan membantu dalam penyusunan semula. Kelebihan alat ini adalah pelaksanaan pengendaliannya mengawal semakan berlebihan pada pengkompil yang boleh menyebabkan gangguan pada komputer.

Denaro et. al [16] memperkenalkan aliran data yang khusus digunakan untuk ujian program berorientasikan objek seperti ujian antara prosedur. Ia dikenali sebagai Perwakilan *DynaFlow*. Ia dibangunkan untuk mengeksploitasi maklumat aliran data yang berguna untuk diekstrak secara dinamik dari kod sumber untuk menjana kes-kes pengujian yang baru bagi sistem yang sedang diuji. Hasil daripada pendekatan ini adalah kegagalan dapat dikenal pasti yang tidak dapat dikesan semasa ujian awal. Ianya berkesan dan tidak bergantung kepada saiz kes-kes pengujian yang lain.

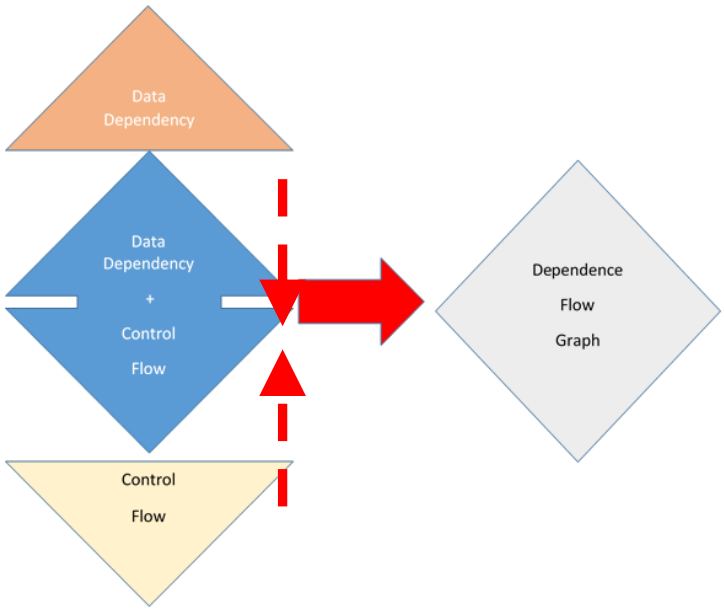
3. Metodologi Penyelidikan

Persembahan kod menggunakan graf aliran pergantungan (GAP) adalah untuk mewakili graf kawalan dan pergantungan aliran kod secara serentak. Ia adalah satu teknik untuk menunjukkan hubungan dan pergerakan kawalan aktiviti dan pergantungan data dengan hanya menggunakan satu graf sahaja berbanding dua graf berasingan. Inisiatif untuk membangunkan GAP adalah berpunca dari kerumitan program yang dibangunkan dengan menggunakan kaedah berorientasikan objek.

Konsep asas GAP boleh dirujuk pada Rajah 1. Ia menunjukkan bahawa terdapat dua sasaran yang berbeza yang ditonjolkan dalam konsep asas pembangunan graf ini. Yang pertama adalah aliran kawalan dan yang kedua pula ialah kebergantungan data. Dalam kebanyakan kajian yang ada, [8, 15, 16, 17] pembentangan aliran kawalan program ditunjukkan dengan menggunakan graf aliran kawalan. Ia memberikan maklumat tentang kenyataan atau pembolehubah aliran peristiwa dari satu nod kepada nod yang lain dalam program. Satu lagi sasaran adalah untuk mengetahui kebergantungan antara nod dalam program ini. Dengan memahami hubungan antara nod akan membantu untuk menunjukkan hubungan kebergantungan antara satu data dengan data yang lain. Apa yang membezakan antara aliran data dan kawalan dalam satu struktur kod adalah graf kawalan memberikan maklumat tentang aliran kawalan pembolehubah atau kenyataan dalam satu-satu program. Manakala, graf pergantungan pula memberikan maklumat tentang hubung kait antara satu nod data dengan nod data yang lain.

Dalam menerangkan kaedah pergantuntungan antara kod-kod dalam analisa program, kebiasaannya asas "*du-ud chain"* digunakan. Begitu juga dalam menganalisa data untuk membangunkan graf kawalan aliran dan graf pergantungan.

Ia adalah lebih mudah untuk menentukan secara langsung bagaimana menghubungkan label penyataan yang menghasilkan nilai kepada label penyataan yang menggunakannya. Bagi setiap penggunaan berubah-ubah, mengaitkan semua tugasan yang mencapai penggunaan yang dipanggil rantaian penggunaan definisi atau *"ud-chain"*. Bagi setiap tugasan, mengaitkan semua penggunaan dipanggil rantaian Definisi-penggunaan atau *"du-chain"*. Takrif taraf *du-chain* dan *ud-chain* adalah seperti dalam definisi 1 dan 2.



Rajah 1. Lakaran Konsep Pembentukan GAP

Definisi 1. Takrifan nod x

dikatakan mencapai 'penggunaan' x jika terdapat laluan aliran kawalan daripada takrifan untuk kegunaan yang tidak melalui mana-mana definisi selain dari x.

***du-chain** merupakan pembolehubah x dengan sepasang nod (n1, n2) dan apa-apa yang mentakrifkan n1 x, menggunakan n2 x, dan mentakrifkan x di n1 mencapai penggunaan x di n2.*

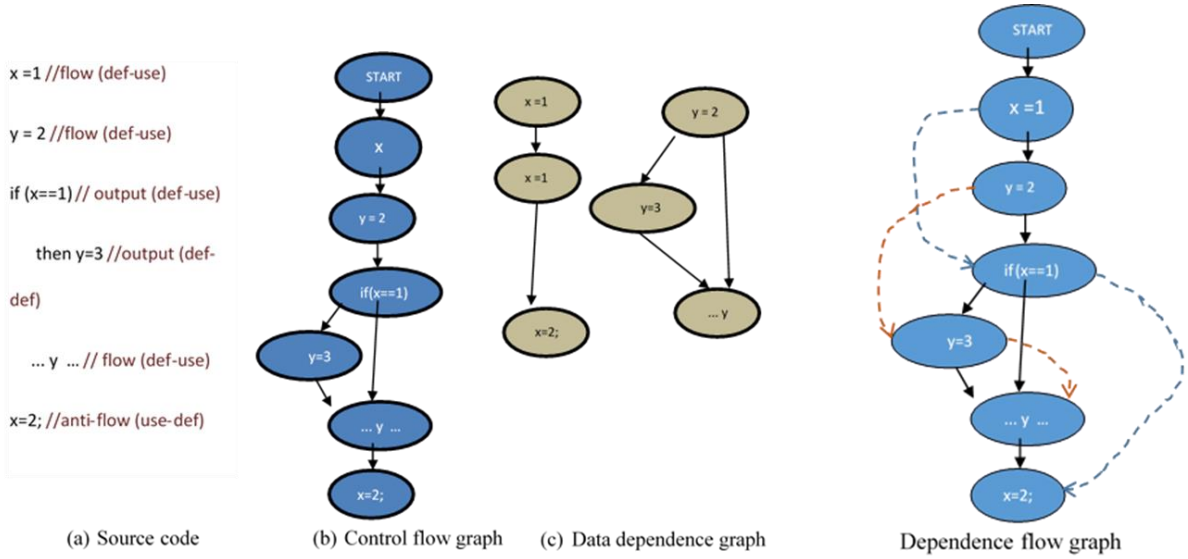
Definisi 2. Takrifan pembolehubah x

x dikatakan mencapai definisi x jika terdapat laluan aliran kawalan daripada penggunaan untuk mentakrifkan bahawa tidak melalui sebarang takrifan yang selain x.

***ud-chain** merupakan* pembolehubah x jika terdapat sepasang nod (n1, n2) yang menggunakan n1 x, n2 dan menentukan x, serta kegunaan x di n1 mencapai pentakrifan x di n2.

Rajah 2 menunjukkan contoh pelaksanaan kedua-dua definisi tersebut. Rajah 2(a) adalah contoh kodnya. Disebelahnya adalah komen tentang penggunaan *def-use*. Rajah 2 (b) adalah bentuk graf kawalan aliran, dan 2(c) adalah graf pergantungan. Rajah graf yang di sebelah 2(c) adalah gabungan antara graf kawalan aliran dengan graf pergantungan yang menghasilkan graf aliran pergantungan. Nod mewakili sama ada kenyataan kombinasi antara aliran data atau aliran kawalan. Garisan bersambung adalah aliran kawalan dan garisan putus-putus menunjukkan aliran pergantungan. Nod tugasan mempunyai pengganti tunggal, sementara nod bersyarat mempunyai dua pengganti yang mewakili cawangan kemungkinan kawalan.

Def-use adalah rantaian untuk graf pergantungan yang mempunyai nod yang sama seperti graf aliran kawalan, tetapi ianya bersambungan dengan setiap pembolehubah yang digunakn dalam definisi yang diberikan di atas. Dalam Rajah 2(c) garisan dalam graf mewakili kebergantungan yang dikelaskan sebagai aliran (def-use), anti (use-def), atau hasil pergantungan (def-def). Perlu difahami bahawa graf pergantungan tidak mewakili perlaksanaan (execution) program dan juga bukan menggabungkan maklumat mengenai aliran kawalan.



Rajah 2. Contoh program dan grafnya

4. Pembinaan Gap

Seksyen ini menerangkan pembentukan GAP secara keseluruhannya. Sintesis dan gambaran bagaimana GAP dibangunkan bermula dari menggabungkan graf aliran kawalan dengan graf perwakilan. Ini menunjukkan bahawa GAP adalah turutan hubungan hibrid yang terdiri daripada nod yang mempunya aliran kawalan dan data pergantungan antara kelas, metod atau pernyataan di dalam satu graf sahaja.

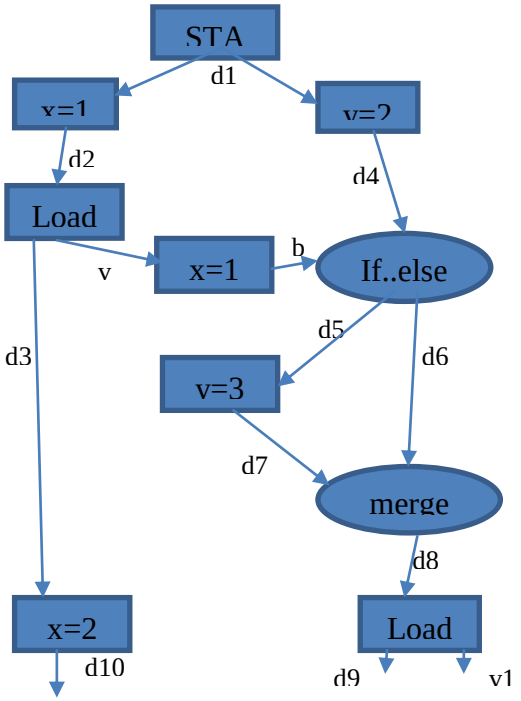
Graf aliran kawalan adalah graf yang membentuk nod menjadi penghubung antara satu blok dan kepada satu blok yang lain di dalam program [3]. Garisan lurus digunakan untuk menggambarkan aliran kawalan yang berlaku. Aliran kawalan bermaksud susunan kod yang dikompilkan dan dilaksanakan secara turutan bermula dari kod barisan pertama dalam program hingga ke barisan terakhir. Di dalam graf aliran kawan, sekiranya nod *x* berhubung dengan nod *y*, nod *x* dinamakan sebagai *successor* dan nod *y* dinamakan sebagai *predecessor*.

Pergantungan data adalah kod pernyataan yang berkaitan dengan pernyataan yang lain di dalam satu program. Graf pergantungan adalah persembahan pernyataan yang mengandungi nod dan alirannya [2]. Nod adalah simbolik kepada sama ada metod atau pernyataan di dalam program. Aliran pula merupakan simbolik kepada transisi data dari satu nod kepada nod yang lain. Ianya menjadi sama ada input atau output kepada, sama ada metod atau pernyataan. Hubungan antara satu nod dengan nod lain menggunakan dua jenis hubungan, sama ada menggunakan rantaian *du* atau *ud*.

Ramai penyelidik [10, 14, 6, 8] menggunakan graf aliran kawalan sebagai teknik persembahan graf pengkomputeran selain graf pergantungan. Dalam penyelenggaraan perisian, kedua-dua teknik boleh digunakan, bahkan sangat digalakkan. Tetapi kedua-dua graf tersebut memberikan maklumat yang berbeza dan tertakluk kepada apa maklumat yang diperlukan oleh jurutera perisian. Sekiranya maklumat tentang aliran kawalan program yang diperlukan maka graf aliran kawalan yang sesuai digunakan. sekiranya maklumat yang diperlukan adalah berkenaan dengan aturan pergantungan data dalam program yang diperlukan, maka graf sesuai digunakan ialah graf pergantungan. Aliran kawalan dan data tidak bebas. Ianya dibina mengikut aturan dan berkaitan dari satu baris ke baris yang lain. Ini memerlukan sedikit analisa untuk memahami penggunaan dalam definisi *def-use* dan *use-def* mengikut definisi yang disebutkan sebelum ini.

Masalah yang dihadapi oleh graf pergantungan adalah mewarisi pergantungan data seperti perkembangan yang berterusan yang memerlukan pelaksanaan untuk melaksanakan transformasi program ini. Masalah ini akan berkait dengan graf aliran pergantungan (GAP). GAP mempunyai keupayaan untuk mewakili pelaksanaan semantik dan kebergantungan dengan nod data dan juga boleh melihat struktur data dan mudah untuk mengesahkan aliran pergantungan. Selain itu, GAP juga boleh dilaksanakan dan semantiknya dalam bentuk yang general.

Untuk lebih memahami GAP, rajah 3 boleh dijadikan contoh untuk dilaksanakan. Perlaksanaan graf ini bermula dari pada nod START dengan menghantar satu tanda ke stor operasi x=1 dan y=2. Bergantung kepada tanda itu, sama ada diterima melalui saluran b TRUE atau FALSE. Lebih senang untuk difahami, GAP adalah satu graf yang membentangkan maklumat tentang aliran kawalan bersama dengan pergantungan data secara lengkap. Ia merupakan graf hibrid yang menggabungkan dua graf yang berbeza yang digunakan oleh kebanyakan penganalisa program yang digabungkan menjadi satu graf.



Rajah 3. Contoh GAP pada satu program

Di dalam program berorientasikan objek terdapat beberapa ciri yang tidak sama dengan program berorientasikan kaedah tradisional. Ini kerana di dalam OO terdapat ciri-ciri seperti pewarisan, pengkapsulan, polimorfisma, pengabstrakan dan lain-lain. OO memberi faedah besar dalam projek besar di mana kaedah pengaturcaraan yang digunakan menghasilkan banyak gelung bersyarat serta cabang-cabang yang rumit, sehingga sukar hendak difahami dan diselenggara. Ciri-ciri pengaturcaraan berorientasi objek telah dimuatkan dalam pelbagai Bahasa Pengaturcaraan. Ini termasuklah Ada, BASIC, Lisp, Pascal, dan banyak lagi. Disebabkan bahasa-bahasa ini pada dasarnya bukan berkonsepkan OO, maka timbulnya masalah keserasian dan penyelenggaraan kod. Bahasa pengaturcaraan berorientasi objek yang "tulen" pula, kurang ciri-ciri yang sering digunakan oleh Juruaturcara.

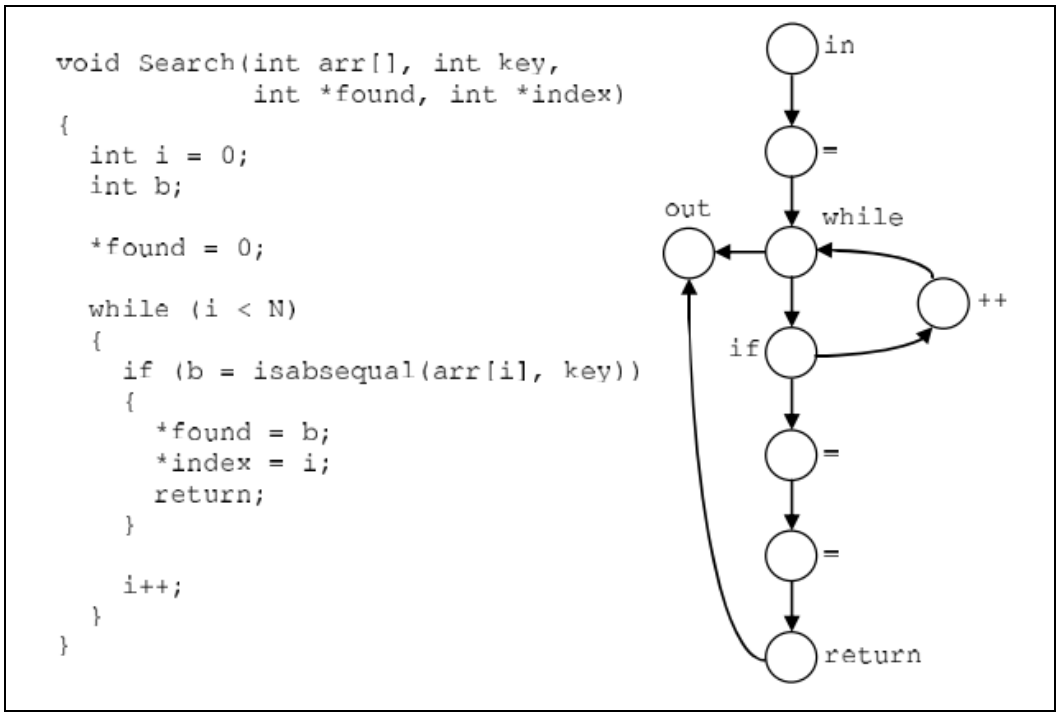
3.1 Aliran Kawalan Dalam GAP

Definisi 1. Karakter dalam dasar analisis;

Char1*: Laluan utama xThe entry point of the routine x.*

Char2*: Sasaran cawangan y atau*

Char3*:* Arahan berikutan y cawangan atau kembali ke x.



Rajah 4. Pembentukan graf aliran kawalan dalam GAP

3.2 Data Pergantungan Dalam GAP

Pergantungan Data ditakrifkan sebagai nod yang mewakili kenyataan program yang mewakili kebergantungan data antara pernyataan. Nod adalah data bergantung kepada yang lain jika ia merujuk kepada keadaan yang berubah-ubah (komponen atau sifat) yang ditakrifkan atau dikemaskinikan oleh nod lain.

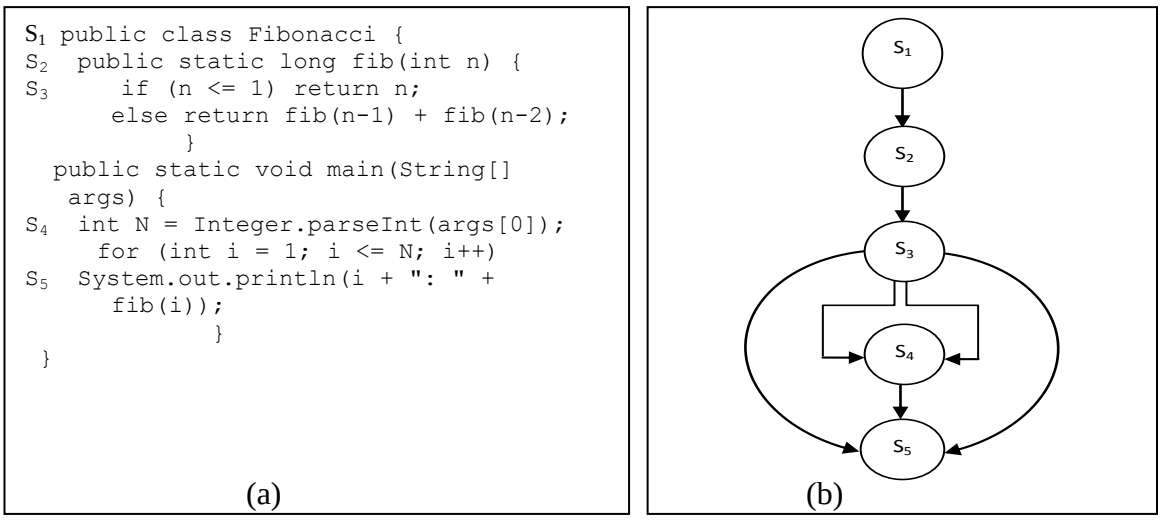
Dalam usaha membantu memudahkan pembangun perisian untuk memhami kaedah OO ini, GAP dibangunkan. Dari segi konsep pembangunannya, graf GAP terbentuk antara kenyataan S1 dan S2 di mana S1 adalah definisi v yang dinamik dan S2 adalah kegunaan v dinamik. Terdapat laluan dari S1 ke S2 dan v yang statik. Takrif v dalam S1 mencapai penggunaan v di S2. Jika kenyataan S2 adalah aliran bergantung kepada kenyataan S1, hubungan S1 dan S2 dikenali sebagai *def-use*.

Berbanding graf aliran kawalan, GAP boleh diaplikasikan kepada setiap aras dalam program. Ini kerana sumber analisa pergantungannya akan melaksanakan berdasarkan arahan pelaksanaan S yang mewakili pernyataan. Jika S1 dan S2 mendahului (S1 ·S2) pelaksanaan, maka S2 adalah bergantung kepada S1. Terdapat 4 jenis pergantungan data pada GAP[10]:

Definisi 2. Jenis kebergantungan

- ***Jenis 1****:* Aliran pergantungan / pergantungan TRUE; Jika $S_1 \prec S_2$ dan bekas set nilai yang digunakan oleh set yang kemudian.
- ***Jenis 2****: Anti pergantungan; If* $S_1 \prec S_2$ *,* S_1 *menggunakan beberapa nilai pembolehubah dan* S_2 *yang menetapkannya.*
- ***Jenis 3****: Output pergantungan; If* $S_1 \prec S_2$ *dan kedua-dua pernyataan menentukan nilai pembolehubah yang lain.*
- ***Jenis 4****: Input pegantungan; If* $S_1 \prec S_2$ *kedua-dua pernyataan membaca nilai pembolehubah yang lain.*

Rajah 5 adalah contoh empat jenis karakter yang disebutkan dalam Definisi 2. Ia merupakan ringkasan analisis kod untuk memberikan gambaran jelas tentang apa yang dihuraikan dalam definisi. 4(a) adalah contoh kod ringkas dan 4(b) adalah penggunaan GAPnya. S3 menggunakan nilai beberapa pembolehubah ini (e) dan S4 menetapkan ia sebagai jenis 2. S3 dan S5 ditetapkan nilai beberapa pembolehubah yang disebutkan dalam jenis 3. Dan akhirnya jenis 4 adalah bergantung antara S3 dan S5 kerana kedua-dua membaca nilai e.



Rajah 5. Contoh Pernyataan S Dalam GAP

5. Dapatan

Satu set soal selidik telah digunakan sebagai instrumen untuk mengumpul pendapat 20 pembangun perisian senior dari 20 buah syarikat pembangun perisian yang berbeza. Semua syarikat yang dimaksud ini terlibat dalam pembangunan perisian dengan pengalaman lebih dari sepuluh tahun. Sebelum itu, ujian rintis (pilot test) telah dibuat terlebih dahulu kepada pembangun perisian di syarikat-syarikat berkenaan. Fokus kajian adalah untuk mengenalpasti sejauh manakah kaedah GAP ini membantu dalam urusan penyelenggaraan perisian yang telah dibangunkan oleh syarikat tersebut. Semua soalan yang ditanya adalah berkisar pada persoalan sejauh mana keberkesanan kaedah GAP ini. Keberkesanan adalah keupayaan GAP dalam membantu pembangun perisian yang bertanggungjawab untuk mendapatkan pandangan yang jelas dan mudah mengenal pasti rantau yang perlu kepada pengubahsuaian berdasarkan petunjuk daripada graf GAP.

Jadual 1: Hasil Kajian

Soalan	Pilihan	Kekerapan	%
a.Apakah GAP memberi kesan yang baikdan berkesan untuk mempersembahkan kod?	Ya	16	80.0
	Tidak	4	20.0
b. Adakah maklumat yang GAP bekalkan cukup untuk aktiviti penyelenggaraan	Ya	17	85.0
	Tidak	3	15.0
c. Adakah GAP boleh membantu dalam kerja penyelenggaraan perisian?	Ya	17	85.0
	Tidak	3	15.0
d. Adakah GAP boleh meminimumkan masa penyelenggaraan anda?	Ya	18	90.0
	Tidak	2	10.0
e. Adakah GAP boleh membantu jurutera perisian untuk menyelesaikan kerja penyelenggaraan terutama untuk perisian yang kompleks complexity problem in software maintenance?	Ya	17	85.0
	Tidak	3	15.0

Soalan a – e adalah satu set soalan untuk mendapatkan pendapat dari responden mengenai keberkesanan GAP dalam membantu kerja penyelenggaran perisian. Hasilnya adalah seperti di Jadual 1. Keputusan adalah 80% bersetuju bahawa GAP berkesan mewakili struktur kod dan 85% bersetuju bahawa GAP membekalkan maklumat yang cukup untuk membantu kerja penyelenggaraan. 85% bersetuju bahawa GAP boleh membantu dalam kerja penyelenggaraan perisian dan 90% memaklumkan bahawa GAP membantu mereka menjimatkan masa mereka dalam mengubah suai program. Akhir sekali, soalan e yang diajukan, sama ada GAP boleh membantu penyelenggara perisian untuk menyelesaikan masalah kerumitan dalam penyelenggaraan perisian atau tidak. Maklum balas yang diperolehi adalah 85% bersetuju bahawa ia adalah berguna dalam penyelenggaraan perisian.

Dari makumbalas daripada dua puluh responden dari dua puluh syarikat pembangun perisian yang mempunyai pengalaman yang lama dalam pembangunan perisian, kita boleh mengatakan bahawa mereka berpuas hati dengan GAP sebagai salah satu alat sokongan pembangunan perisian khususnya dalam aspek penyelenggaraan. Ia berkesan dalam menyediakan maklumat, menjimatkan masa, dan menyelesaikan kerumitan dalam mengesan hubungan kod terutamanya dalam program yang bersaiz besar dan sederhana. GAP boleh membantu penyelenggara perisian bukan sahaja untuk mengenal pasti struktur keseluruhan program, tetapi juga boleh membantu mereka untuk mengurangkan masa melakukan aktiviti analisa hubungan keseluruhan program yang dibangunkan dengan kaedah OO dengan usaha yang minimum.

6. Kesimpulan

Kajian ini merupakan kajian tentang pembentangan segera (intermediate representation) kod dengan menggunakan teknik graf aliran pergantungan berbanding teknik-teknik lain seperti graf aliran kawalan, graf pergantungan, graf panggilan dan sebagainya. Graf aliran kawalan adalah gabungan antara dua graf popular yang biasa digunakan iaitu graf pergantungan dan graf aliran kawalan yang digabungkan menjadi satu graf. Ianya memberikan maklumat yang terdapat pada dua-dua graf tersebut di dalam satu graf yang lebih efektif dan mudah difahami. Ini memudahkan juruanalisa

sistem untuk menganalisa program dan membantu dalam proses penyelenggaraan agar menjadi lebih mudah dan cepat.

Rujukan

1. Dale, N. 2005a. SIGCSE members survey. http://www.cs.utexas.edu/users/ndale/ContentResults.html
2. Dale, N. 2005b. Non SIGCSE members survey. http://www.cs.utexas.edu/users/ndale/ContentResults2.html
3. Zhao, J., & Rinard, M. (2003). System Dependence Graph Construction for Aspect-Oriented Programs. *Laboratory for Computer Science*.
4. Masuhara, H., Kiczales, G., & Dutchyn, C. (2002). Compilation Semantics of Aspect-Oriented Programs, 17-26.
5. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J., … Lopes, C. (1997). Aspect-Oriented Programming Aspect-Oriented Programming, (June).
6. Elrad, T. (2001). ASPECT-ORIENTED PROGRAMMING, *44*(10), 28–32. ACM
7. P. E. Livadas andS.Croll, (2005) *"System Dependence Graphs Based on Parse Treesand their Use in Software Maintenance"*, IEEE Transactions.
8. Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011). Using Program Slicing To Improve the Efficiency and Effectiveness of Cluster Test Selection. *International Journal of Software Engineering and Knowledge Engineering*, *21*(06), 759–777.
9. Lin, Y. L. Y., Zhang, S. Z. S., & Zhao, J. Z. J. (2009). Incremental call graph reanalysis for AspectJ software. *2009 IEEE International Conference on Software Maintenance*.
10. Syarbaini Ahmad, (2003) *Development of Computerize Maintenance Management System for SE Project* (Thesis), *CASE-UTM Kuala Lumpur.*
11. Lientz B; Swanson E, Software Maintenance Management, Reading, Mass., Addison-Westley, 2008, p.214.
12. Shi, H. (2004). *Inspection of OO Software with Incomplete Documentation Using A Document Driven Approach*.
13. Isong, B. (2014). Explicit Object-Oriented Program Representation for Effective Software Maintenance. *Advances in Computer Science: an International Journal*, 113-123.
14. Kanade, A. A. (2010). Representation dependence testing using program inversion. *In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering* (pp. 277-286). ACM.
15. Duboscq, G., Würthinger, T., Stadler, L., Wimmer, C., Simon, D., & Mössenböck, H. (2013). An Intermediate Representation for Speculative Optimizations in a Dynamic Compiler. *Proceedings of the 7th Workshop on Virtual Machines and Intermediate Languages (VMIL '13)*, None. http://doi.org/10.1145/2542142.2542143
16. Denaro, G., & Margara, A. (2015). Dynamic Data Flow Testing of Object Oriented Systems. *... Conference on Software ...*. Retrieved from http://www.people.usi.ch/vivantim/papers/dynamicdf.pdf
17. Ray, A., Mishra, S., Mohapatra, D., P., (2014) "A Novel Approach for Com- puting Dynamic Slices of Aspect-Oriented Programs", *arXiv preprint arXiv:1403.0100*.
18. Tamrawi, A., Nguyen, H. A., Nguyen, H. V., & Nguyen, T. N. (2012). Build code analysis with symbolic evaluation. *Proceedings - International Conference on Software Engineering*, 650–660. http://doi.org/10.1109/ICSE.2012.6227152