EVALUASI DAN PERBAIKAN KUALITAS DESAIN DIAGRAM KELAS

Arwin Halim

STMIK MIKROSKIL
Jl. Thamrin No. 112, 124, 140 Medan 20212

arwin@mikroskil.ac.id

Abstrak

Dalam proses pengembangan dan pemeliharaan proyek perangkat lunak, kualitas merupakan salah satu hal penting yang menjadi penentu keberhasilan perangkat lunak.Kesalahan yang tidak ditemukan pada awal pengembangan akan membutuhkan sumber daya, biaya, dan waktu perbaikan yang lebih tinggi. Salah satu tahapan yang dilakukan saat proses pengembangan perangkat lunak adalah pemodelan data. Pada perangkat lunak yang berorientasi objek, data biasanya dimodelkan dalam bentuk diagram kelas. Kualitas pada diagram kelas sangat tergantung pada pengetahuan dari perancang. Oleh karena itu, berbagai metrik telah dikembangkan untuk menilai kualitas desain dari berbagai aspek. Pada paper ini, Penulis mengusulkan sebuah pendekatan dan model untuk mengevaluasi, mendeteksi, dan memperbaiki desain kelas diagram, sehingga sesuai dengan kriteria kualitas yang diharapkan.

Kata kunci : *UML, metrik diagram kelas, kualitas desain, evaluasi desain, perbaikan desain*

1. Pendahuluan

Teknologi berorientasi objek menggambarkan pengembangan perangkat lunak yang terdiri dari beberapa tahapan proses dengan menggunakan notasi diagram [1]. *Unified Modelling Language* (UML) secara *de facto* telah digunakan sebagai bahasa untuk memodelkan arsitektur dan desain perangkat lunak berorientasi objek. Model UML dapat digunakan dalam beberapa tahapan *software engineering*, seperti analisis domain permasalahan, dokumentasi arsitektur, atau spesifikasi rincian desain pada sistem [2]. UML 2.0 telah menyediakan 13 notasi diagram yang digunakan untuk memodelkan sistem, seperti diagram sekuensial, diagram aktivitas, diagram kelas, dan lain-lain.

Pemodelan data merupakan langkah awal dalam melakukan desain database. Pada perangkat lunak berorientasi objek, data akan dimodelkan dalam notasi diagram UML yaitu diagram kelas. Diagram ini digambarkan dalam bentuk graf yang terdiri dari *node* sebagai *class* dan *interface*, dan *edge* sebagai *relationship* [1]. Dalam proses pengembangan dan pemeliharaan sistem, desain diagram kelas dapat berubah sesuai dengan spesifikasi. Perubahan tersebut dapat memicu munculnya kesalahan pada desain. Kesalahan yang tidak ditemukan pada awal tahapan pengembangan akan meningkatkan waktu, sumber daya, dan biaya untuk perbaikan kesalahan tersebut [3].

Kualitas dari desain ditunjukkan dengan model desain yang mudah diuji. Aliran kontrol desain yang berorientasi objek sudah tidak hirarki, tetapi telah tersebar dan terbagi dalam keseluruhan arsitektur desain [4]. Beberapa usulan dalam menentukan kualitas dari desain database, khususnya *Entity Relationship Diagram* (ERD) telah didiskusikan pada paper [3]. Pada paper ini, Penulis akan berfokus pada cara evaluasi dan meningkatkan kualitas desain diagram kelas. Proses tersebut dapat dilakukan melalui pendekatan dan model yang diusulkan, sehingga kelemahan pada desain (kesalahan) dapat dikurangi. Proses ini merupakan salah satu

langkah pencegahan terhadap kegagalan perangkat lunak dengan meningkatkan kualitas desain sebelum diimplementasikan dan diuji pada tahapan pengujian.

Pada bagian kedua paper akan memberikan gambaran mengenai penelitian yang telah pernah dilakukan sebelumnya. Pada bagian ketiga, Penulis mengusulkan sebuah pendekatan dan model untuk mengevaluasi dan meningkatkan kualitas desain, dan rancangan tersebut akan didiskusikan pada bagian keempat. Pada bagian terakhir, berisi kesimpulan dan menunjukkan arah penelitian yang akan dilakukan selanjutnya.

2. Kajian Pustaka

Bagian ini akan menunjukkan landasan teori dan berbagai penelitian sebelumnya yang menjadi landasan dalam penulisan paper ini.

2.1 Metrik untuk UML Diagram Kelas

Tidak ada sebuah metrik yang tersedia untuk mengukur kompleksitas dari perangkat lunak [5]. Fenton and Pfeeger mengemukakan, kompleksitas struktural yang rendah akan meningkatkan *reliability* dari perangkat lunak. Banyak metrik telah diperkenalkan untuk mengukur *complexity, understandability, modifyability, maintainability, reliability*, dan *testability* pada diagram kelas [6-9]. Hasil penelitian Genero et al [6] menunjukkan beberapa metrik yang telah dikembangkan pada paper [10] dapat digunakan untuk mengukur faktor kualitas *modifiability, understandability* dan *usability* dari desain. Metrik tersebut telah divalidasi dengan pendekatan berdasarkan *property*oleh Briand et al [11] dan menggolongkannya dalam ukuran, kompleksitas, panjang, dan coupling yang ditunjukkan pada tabel 1.

Tabel 1. Validasi Metrik yang dilakukan Briand et al.

	Size	*Complexity*	*Length*	*Coupling*
Class Diagram – scope metrics	NAggH, NGenH	NAssoc, NDep, NAgg, NGen	MaxHAgg, MaxDIT	
Class Scope Metrics	NDP, NP, NW		HAgg	NAssocC, NDepIN, NDepOUT

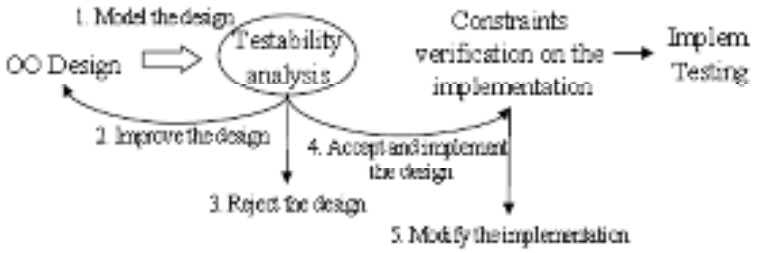
Benoit et al [4] mengukur kualitas desain dengan merepresentasikan diagram kelas ke dalam model abstrak yang disebut *Class Dependency Graph*. Perhitungan kompleksitas struktural model tersebut dilakukan berdasarkan *class interaction* dan *interaction complexity*. *Class interaction* dapat menunjukkan sumber potensi terjadinya kesalahan pada desain diagram kelas.

Tong Yi [12] mengusulkan pengukuran kompleksitas desain berdasarkan *relationship* antar kelas. Fokus utama dalam metrik ini adalah adanya pengaruh perbedaan jenis *relationship* terhadap kualitas desain. Model ini memanfaatkan algoritma *pagerank* untuk menghitung pengaruh dan kompleksitas antar class.

2.2 Metodologi Pengujian

Benoit et al [4] memperkenalkan sebuah metodologi pengujian kualitas untuk desain berorientasi objek yang ditunjukkan pada Gambar 1. Langkah-langkah pengujian kualitas desain yang diusulkan terdiri dari:

1. Inisialisasi desain berorientasi objek (1), dibentuk dengan menghitung interaksi antar kelas (class interaction) sebagai kompleksitas dari desain.
2. Jika nilai kompleksitas dirasakan masih terlalu tinggi, maka kita dapat memutuskan apakah desain akan diperbaiki (2) atau desain akan ditolak (3).
3. Perbaikan desain akan diterima jika telah berhasil menurunkan nilai *coupling* pada arsitektur atau menunjukkan batasan yang jelas, sehingga dapat mengurangi kesalahan saat implementasi.
4. Ketika perbaikan sudah sesuai dengan spesifikasi, desain akan diimplementasi (4), dan batasan yang telah ditambahkan pada desain harus diverifikasi sebelum dilakukan pengujian (5).



Gambar 1. Metodologi Pengujian Kualitas Desain Berorientasi Objek

2.3 Peningkatan Kualitas Perangkat Lunak

Kualitas perangkat lunak dapat diperbaiki dengan berbagai cara. Salah satu cara perbaikan kualitas adalah *refactoring*. Reddy and Rao [13] mengusulkan metode untuk mendeteksi cacat dari desain dan melakukan perbaikan kualitas dari perangkat lunak dengan melakukan refactoring. Metode yang diusulkan tersebut, menghitung nilai Dcoupling antar artifak pada kelas yang menjadi indikator tingkat ketergantungan kelas pada desain. Nilai Dcoupling digunakan untuk menghitung kompleksitas untuk keseluruhan struktur desain diagram kelas.

3. Metodologi Penelitian

Dalam menyelesaikan penulisan ini, Penulis melakukan proses pengumpulan data berupa informasi dan literatur yang berhubungan dengan metode, alat bantu, dan teknik secara teoritis dan praktis dalam mengukur kualitas perangkat lunak, khususnya perangkat lunak berorientasi objek. Penulis mempelajari literatur berupa penelitian yang sudah ada pada jurnal dan konferensi, serta landasan teori yang berasal dari buku. Informasi tersebut digunakan sebagai landasan utama Penulis untuk mengembangkan sebuah pendekatan kolaboratif yang dibentuk dari hasil modifikasi pendekatan dan metodologi yang telah ada serta ulasan dari penelitian terbaru (*state-of-the-art*).

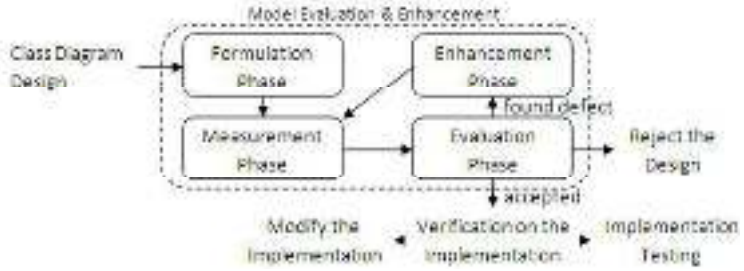
4. Usulan Pendekatan untuk Evaluasi dan Perbaikan Kualitas Desain

Perangkat lunak dapat diukur kualitas dan kompleksitasnya dengan menghitung nilai metrik. Metrik kualitas akan melakukan proses kuantifikasi faktor kualitas tertentu berdasarkan atribut perangkat lunak. Metrik akan menjadi salah satu bagian penting pada pendekatan yang diusulkan pada paper ini. Pendekatan yang diusulkan merupakan hasil modifikasi dari metodologi pengujian yang dikembangkan oleh Benoit et al [4] dan

dikembangkan dari metodologi testing untuk model spiral. Metodologi testing untuk model spiral terdiri dari*Planning, Evaluation, Risk analysis,*dan *Engineering.*

Teknik yang digunakan untuk melakukan pengukuran pada setiap langkah pada pendekatan usulan merupakan kolaboratif dari teknik pengukuran yang telah dikembangkan pada penelitian lain dan berfokus pada berbagai faktor kualitas, seperti Boudry dkk[4] yang berfokus pada pemgukuran kualitas berdasarkan *testability*, metric NA dan Ngen yang mampu memprediksi *Modifiability* [6], dan lain-lain.

Paper ini mengusulkan sebuah pendekatan kolaboratif untuk mengevaluasi desain berdasarkan faktor kualitas dari *desain class diagram*, mendeteksi potensi terjadinya *defect*, dan memperbaiki desain dengan mengeliminasi *defect* yang terdeteksi. Gambar 2 menunjukkan gambaran umum evaluasi dan perbaikan kualitas desain diagram kelas.



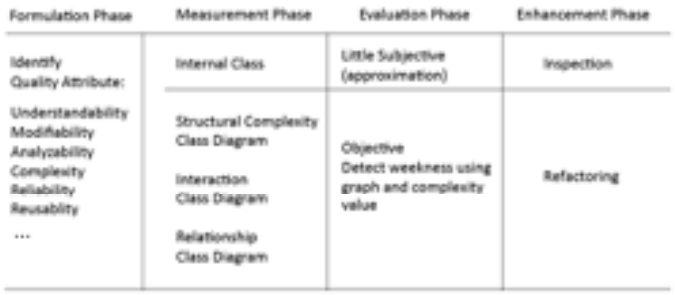
Gambar 2. Usulan Pendekatan untuk Evaluasi dan Perbaikan Kualitas Desain Diagram Kelas

Pendekatan yang diusulkan terdiri dari empat tahapan, antara lain:

1. *Formulation Phase*. Pada tahapan ini, perancang harus menentukan tingkat kualitas yang ingin dicapai dari desain diagram kelas (*planning*). Hal ini dikarenakan adanya *trade-off* diantara atribut kualitas seperti kompleksitas dan reliabilitas. Atribut kualitas dapat berupa *understandability, reusability, reliability*, dan lain-lain.
2. *Measurement Phase*. Pada tahapan ini, pengukuran (evaluation) dibagi dalam empat kategori antara lain:
    a. Internal class. Pada bagian ini, dilakukan pengukuran terhadap ukuran kelas (jumlah kelas, jumlah atribut dan jumlah method), tingkat kohesi pada internal kelas, data akses pada atribut kelas, dan rata-rata parameter per metode.
    b. Structural Complexity Class Diagram. Pada bagian ini, dilakukan pengukuran terhadap jenis-jenis hubungan antar class, yaitu association, aggregation, dependency, dan generalisasi.
    c. Interaction Class Diagram. Pada bagian ini, dilakukan pengukuran kompleksitas berdasarkan interaksi antar class (*class interaction*)
    d. Relationship Class Diagram. Pada bagian ini, dilakukan pengukuran kompleksitas class berdasarkan pengaruh class lain, yang menunjukkan pentingnya suatu class pada desain class diagram
3. *Evaluation phase*. Tahapan ini merupakan tahapan analisis dari pengukuran (evaluation) pada *measurement phase*. Tahapan ini terdiri dari evaluasi internal class dan evaluasi metrik. Evaluasi internal class masih subjektif, karena belum memiliki nilai threshold yang jelas. Proses evaluasi metrik yang lain dilakukan secara objektif berdasarkan nilai threshold yang diperoleh dari struktur desain secara langsung atau model abstrak berbentuk *graf* yang diperoleh dari desain class diagram.

4. *Enhancement Phase*. Pada tahapan ini, hasil evaluasi akan diperiksa dan dilakukan inspeksi atau refactoring berdasarkan sumber potensi terjadinya defect yang ditemukan pada tahapan evaluasi (engineering). Bois et al [14] memberikan panduan praktis untuk melakukan refactoring secara optimal.

Penerapan dari pendekatan yang diusulkan ditunjukkan pada Gambar 3.



Gambar 3. Contoh Penerapan Pendekatan Usulan untuk Evaluasi dan Perbaikan Kualitas Desain Diagram Kelas

5. Diskusi

Pada bagian ini, Penulis akan menunjukkan berbagai metrik yang telah diusulkan oleh peneliti sebelumnya untuk diterapkan pada langkah-langkah yang dibahas pada Bagian 4. Selain itu, Penulis juga menentukan sasaran pencapaian untuk setiap tahapan dari model sesuai dengan pendekatan yang diusulkan.

5.1 Formulation Phase

Setelah melewati tahapan ini, perancang diharapkan telah menentukan fokus utama atau kriteria faktor kualitas dari desain yang dapat diterima. Kriteria ini yang akan menentukan tingkat kualitas desain yang ingin dicapai. Sebagai contoh, pada iterasi pertama, kita dapat menentukan faktor kualitas yang paling sesuai dengan kriteria kita berdasarkan kombinasi teknik AHP dan PROMETHEE yang telah diusulkan dalam penelitian Penulis pada referensi [15].

5.2 Measurement Phase

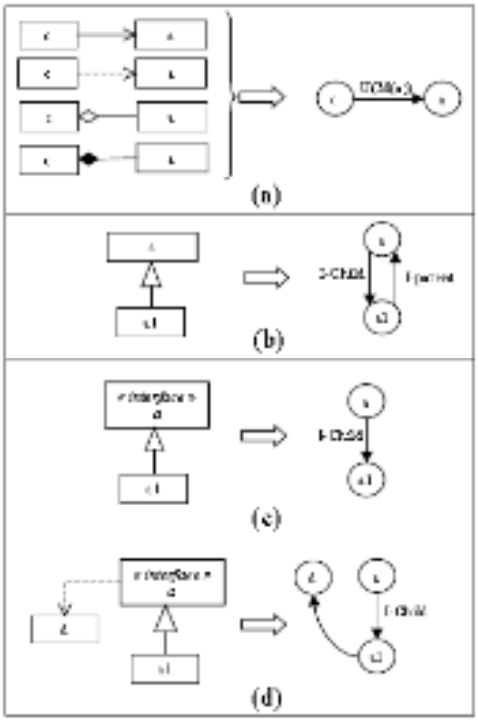
Setelah melewati tahapan ini, perancang diharapkan dapat menentukan apakah desain telah memenuhi spesifikasi kriteria kualitas atau tidak. Keputusan tersebut dapat diambil berdasarkan hasil perhitungan metrik terhadap desain diagram kelas. Sebagai contoh, untuk melakukan pengukuran internal class untuk ukuran kelas, kita dapat menggunakan metrik ukuran kelas yang ditunjukkan pada tabel 2 [10].

Tabel 2. Metrik Ukuran Kelas

No	*Nama Metric*	*Definisi Metrik*
1	Number of Classes (NC)	Jumlah kelas pada desain
2	Number of Attributes (NA)	Jumlah atribut pada kelas
3	Number of Method (NM)	Jumlah metode pada kelas

Untuk menghitung tingkat kohesi pada class, kita dapat menggunakan metriks WMC [16] atau CAMC [11], dan rata-rata paramater per metode dapat dihitung dengan metrik APPM [11]. Pengukuran *structural complexity class* akan disesuaikan dengan faktor kualitas yang akan dicapai. Jika fokus desain adalah *maintainability* maka aspek kriteria yang diperhatikan adalah *understandability* dan *modifiability*. *Understandability time* pada desain dapat diprediksi dengan menghitung metrik NM dan MaxHAgg, dan *modifiability time* dapat diprediksi dengan menghitung metrik NA dan NGen [6].

Pengukuran *interaction class* dilakukan dengan menghitung kompleksits berdasarkan class interaction yang mungkin terjadi [4]. Langkah awal dari proses ini adalah dengan membentuk *Class Dependency Graph* (CDG) sesuai dengan aturan yang ditunjukkan pada Gambar 4.



Gambar 4. Aturan Konversi Diagram Kelas Menjadi CDG

Proses dilanjutkan dengan mencari kelas yang berpotensi melakukan interaksi. Kelas-kelas tersebut adalah dua class yang memiliki *origin* dan *end* yang sama, tetapi memiliki jalur penelurusan (itVertices) yang berbeda [4]. Langkah terakhir adalah menghitung kompleksitas dari interaksi sesuai dengan persamaan berikut.

$$complexity\ (dp) = h\ .\ (h - l) \tag{1}$$

$$complexity\ (IH, P) = \sum_{i=0}^{nbDP} \text{complexity(dp1)} \tag{2}$$

$$complexity\ (P) = \prod_{i=1}^{nbCrossed} complexity\ (IH1, P) \tag{3}$$

$$\text{complexity (Cl)} = \sum_{j=1}^{nbPaths} (\text{Complexity (Pi)}.\sum_{j>i} \text{complexity (Pj)}) \tag{4}$$

Pengukuran *relationship class* dilakukan dengan menghitung kompleksitas antar class dengan menggunakan algoritma *pagerank* [12] yang berfokus pada *relationship class*. Metrik ini dapat menunjukkan tingkat *relationship* antar kelas yang berbeda walaupun memiliki jenis *relationship* yang sama atau berbeda dengan kelas lain. Metrik ini juga dapat menentukan kelas-kelas yang memiliki pengaruh yang tinggi pada desain diagram kelas yang dirancang. Nilai kompleksitas dan derajat *relationship* antar kelas dapat digunakan untuk mendeteksi kelas-kelas yang berpotensi mengandung kesalahan pada desain diagram kelas.

5.3 Evaluation Phase

Proses evaluasi dilakukan dengan memperhatikan nilai hasil pengukuran dan *threshold* yang diinginkan, sehingga desain dapat disimpulkan dapat diterima, harus diperbaiki, atau harus diganti. Jika desain harus diperbaiki, maka dilakukan proses penentuan class-class yang berpotensi terdapat kesalahan pada desain melalui perhitungan metrik *Coupling Between Object* (CBO) [16], atau DCoupling yang diperkenalkan oleh Reddy & Rao [13]. Jika hasil perhitungan diatas nilai *threshold*, maka kelas-kelas tersebut memiliki kompleksitas yang tinggi dan berpotensi merupakan sumber kesalahan pada implementasi.

5.4 Enhancement Phase

Tahapan ini akan dilaksanakan jika desain diagram kelas belum memenuhi kriteria kualitas yang diharapkan. Proses perbaikan yang dilakukan adalah melakukan inspeksi terhadap kelas yang telah ditentukan pada tahapan evaluasi dan jika diperlukan, melakukan proses *refactoring*. Setelah melewati tahapan ini, perancang diharapkan telah menghasilkan desain baru yang akan dievaluasi kembali pada *measurement phase* dan *evaluation phase*. Proses ini akan terus berulang sampai kriteria kualitas yang diharapkan tercapai.

6. Kesimpulan

Salah satu tahapan yang dilalui saat melakukan proses pengembangan perangkat lunak adalah memodelkan data. Pemodelan data pada perangkat lunak berorientasi objek dilakukan dengan membuat desain UML diagram kelas. Kualitas dari desain akan menentukan kualitas dari keseluruhan perangkat lunak.Kesalahan yang tidak diperbaiki pada desain akan diteruskan ke tahapan berikutnya. Paper ini memperkenalkan pendekatan baru untuk mengevaluasi dan memperbaiki kualitas dari desain diagram kelas. Indikator yang digunakan dalam proses pengukuran adalah faktor kualitas yang ingin dicapai dan faktor pengukuran berdasarkan *internal class*, *structural complexity class*, *interaction class*, dan *relationship class* pada *desain class diagram*. Setelah melakukan pengukuran, langkah selanjutnya adalah melakukan evaluasi dengan menentukan sumber-sumber potensi kesalahan yang dapat diperbaiki dan memperbaikinya untuk memperoleh desain yang lebih baik. Pada penelitian selanjutnya, pendekatan yang diusulkan akan dievaluasi kembali dengan menerapkannya pada studi kasus untuk mengetahui persentase peningkatan kualitas yang dapat dicapai dan merancang *tool* perangkat lunak yang dapat digunakan dalam membantu pengukuran dan perbaikan kualitas dari tahap desain.

Referensi

[1]. A. Alsaadi, 2006,*Checking Data Integrity via the UML Class Diagram*, International Conference on Software Engineering Advances (ICSEA'06), 37-37.

[2]. C.F.J. Lange, 2006, *Improving the Quality of UML Models in Practice*, ICSE'06, 993-996.

[3]. M. Piattini, M. Genero, and C. Calero, 2005, *Data model metrics*, Handbook of Software Engineering and Knowledge Engineering Vol 2: Emerging Technologies, 325-342.
[4]. [B. Baudry, Y. Le Traon, and G. Sunye, 2002, *Testability analysis of a UML class diagram*, Proceedings Eighth IEEE Symposium on Software Metrics, 54-63.
[5]. J. Cardoso, 2005, *Control-flow Complexity Measurement of Processes and Weyuker's Properties*, Engineering and Technology, 213-218.
[6]. M. Genero, M. Piatini, and E. Manso, 2004, *Finding early indicators of UML class diagrams understandability and modifiability*, Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04., 207-216.
[7]. B. Baudry, Y.L. Traon, G. Sunye, and J.-M. Jezequel, 2003, *Measuring and improving design patterns testability*, Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717), 50-59.
[8]. B. Baudry, Y. Le Traon, and G. Sunye, 2004, *Improving the testability of UML class diagrams*, First International Workshop onTestability Assessment, IWoTA 2004, 70-80.
[9]. M.R. Girgis, T.M. Mahmoud, and R.R. Nour, 2009,*UML class diagram metrics tool*, 2009 International Conference on Computer Engineering & Systems, 423-428.
[10]. M. Genero, J. Olivas, M. Piattini, and F. Romero, 2003, *Assessing Object-Oriented Conceptual Models Maintainability*, Lecture Notes in Computer Science including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 2784, 288-299.
[11]. M. Genero, M. Piattini, and C. Calero, 2005, *A Survey of Metrics for UML Class Diagrams*, Journal of Object Technology, vol. 4, 59-92.
[12]. T. Yi, 2009, *Measuring Complexity of Relationships among Classes*, 2009 International Conference on Management and Service Science, 1-4.
[13]. K.N. Reddy and A.A. Rao, 2009, *A Quantitative Evaluation of Software Quality Enhancement by Refactoring Using Dependency Oriented Complexity Metrics*, Second International Conference on Emerging Trends in Engineering and Technology, ICETET-09, 1011-1018.
[14]. B.D. Bois, S. Demeyer, and J. Verelst, 2004, *Refactoring - Improving Coupling and Cohesion of Existing Code*, Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04), 144-151.
[15]. A. Halim, A. Sudrajat, A. Sunandar, I. K. R. Arthana, S. Megawan, and P. Mursanto, 2011, *Analytical Hierarchy Process and PROMETHEE Application in Measuring Object Oriented Software Quality*, IEEE International Conference on Advanced Computer Science and Information Systems, 165-170.
[16]. S.R. Chidamber and C.F. Kemerer, 1994, *A Metrics Suite for Object Oriented Design*, IEEE Transactions on Software Engineering, vol. 20, 476-493.