ONTOLOGI PADA METODE *REQUIREMENTS RECOVERY* DALAM PROSES *REVERSE ENGINEERING*

Elviawaty Muisa Zamzami[1], Eko Kuswardono Budiardjo[2]
[1]*Program.Studi Ilmu Komputer, Universitas Sumatera Utara, Medan*
E-mail : elvi_zamzami@usu.ac.id

[2]*Fakultas Ilmu Komputer, Universitas Indonesia, Depok*
E-mail : eko@ui.ac.id

ABSTRAK

Piranti lunak kerap mengalami perubahan, antara lain disebabkan perubahan bisnis organisasi. Akibatnya, peranti lunak tidak mampu lagi mendukung organisasi sehingga harus dilakukan reengineering terhadapnya. Software reengineering didukung antara lain oleh riset reverse engineering. Riset reverse engineering saat ini lebih terfokus pada pengembangan software tools dan perolehan kembali model fisik dan struktur lojik dalam bentuk model-model dari legacy system [1] . Namun, riset untuk requirements recovery melalui proses reverse engineering masih relatif minim. Disisi lain, requirements sangat penting bagi keberhasilan software reengineering. Dengan demikian, terdapat kepentingan melakukan riset reverse engineering untuk requirements recovery dari peranti lunak yang ada (existing software). Requirements recovery dari peranti lunak dapat memastikan pemahaman lebih baik dari apa yang redundan, apa yang harus dipertahankan, dan apa yang dapat digunakan kembali [1]. Requirements yang diperoleh dapat digunakan pada forward engineering sebagai bagian dari reengineering ataupun menyusun ulang dokumen existing requirements. Pada paper ini, requirements recovery bersumber dari end-to-end interaction antara user dan sistem komputer. Dari existing software, diidentifikasi user dan fitur peranti lunak. Selanjutnya diobservasi end-to-end interaction yang terjadi antara user dan sistem komputer. Hasil identifikasi dan observasi tersebut digunakan untuk membangun ontologi. Ontologi dapat merepresentasikan pengetahuan tentang existing software yang menyiratkan requirements.

Kata kunci : *Software Reengineering, Reverse Engineering, Requirements Recovery, End-to-end Interaction, Ontologi*

1. PENDAHULUAN

Suatu peranti lunak kerap mengalami perubahan. Perubahan dilakukan untuk memenuhi perubahan kebutuhan *stakeholders*. Perubahan mungkin dilakukan langsung terhadap *source code* dan tanpa pendokumentasian perubahan. Keadaan ini memungkinkan dokumen tidak sesuai dengan peranti lunak. Perubahan lain yang berpengaruh terhadap peranti lunak adalah perubahan lingkungan bisnis organisasi yang didukungnya. Akibatnya, peranti lunak tidak mampu mendukung sebagaimana harusnya. Namun, peranti lunak tersebut tidak dapat begitu saja diganti dengan yang baru karena mendukung untuk hal-hal yang krusial dalam organisasi. Keadaan ini mengharuskan *reengineering* terhadap peranti lunak.

Reengineering (rekayasa ulang) dapat dikatakan sebagai solusi terbaik karena tetap memberdayakan peranti lunak yang ada sehingga mencegah kesenjangan yang terjadi akibat perubahan peranti lunak yang lama menjadi peranti lunak yang sama sekali baru. *Reengineering* sebagai 'daur ulang' juga mendukung *green computing* dalam hal efisiensi terhadap volume penyimpanan dan biaya pemeliharaan peranti lunak [2]. Proses dalam *reengineering* mempunyai dua tahapan utama, yaitu tahapan *reverse engineering* dan tahapan *forward engineering*. Tahapan *reverse engineering* dilakukan untuk memperoleh artefak peranti lunak. Artefak tersebut dapat berupa *requirements*, model arsitektur, dan spesifikasi desain peranti lunak. Hasil yang diperoleh dari *reverse engineering* digunakan pada tahapan selanjutnya, *forward engineering*.

Pada *forward engineering* akan dielisitasi *requirements* yang baru. *Requirements* memegang peranan sangat penting dari kesuksesan peranti lunak yang dikembangkan. Jika pada tahapan *forward engineering*, tidak hanya digunakan *requirements* yang baru tetapi juga menyertakan *requirements* yang telah tertanam dalam peranti lunak akan menjadikan *requirements* yang lengkap.

Kelengkapan *requirements* dalam artian terdiri dari *requirements* dari *forward engineering* dan *requirements* dari *reverse engineering* diharapkan mampu menghasilkan peranti lunak yang tetap memenuhi

maksud dan tujuan pengembangan peranti lunak tersebut di waktu lalu dan juga mampu mengadaptasikan dengan maksud dan tujuan saat sekarang. Karenanya, sangat diperlukan *requirements recovery* dari peranti lunak yang akan di-*reengineering* dan juga penyesuaian dokumen peranti lunak.

Telah banyak riset yang dilakukan pada kajian *software reverse engineering*. Namun, kebanyakan riset dan literatur terutama berfokus pada pengembangan *tools* yang berbeda untuk program-program *reverse engineering* dan memperoleh kembali model fisik dan struktur lojik dalam bentuk model-model berbeda untuk *legacy system* [1]. Tidak demikian halnya riset untuk memperoleh *requirements* dari proses *reverse engineering*, masih relatif minim periset yang melakukannya.

Berdasarkan paparan diatas, penting dilakukan riset pada kajian metode *reverse engineering* untuk memperoleh metode *requirements recovery*. Pada riset ini, peranti lunak yang dikenakan *requirements recovery* disebut sebagai *existing software*. Dengan ketidaksesuaian dokumen atau bahkan ketiadaan dokumen mengharuskan metode *requirements recovery* bebas dari penggunaan dokumen. Karenanya, metode ini akan melakukan *requirements recovery* berdasarkan pada identifikasi dan observasi terhadap *end-to-end interaction* antara *user* dan sistem komputer. Selanjutnya dibangun ontologi menggunakan Protégé - OWL. Ontologi tersebut pada akhirnya merepresentasikan pengetahuan tentang *existing software* yang menyiratkan *requirements* yang telah tertanam dalam *existing software*.

2. *SOFTWARE REENGINEERING*

Pada beberapa literatur, *reengineering* didefinisikan sebagai [3]:

a. Memeriksa dan merubah sebuah sistem yang ada untuk rekonstitusinya menjadi sebuah bentuk baru dan mengimplementasikan sub sekuen dari bentuk baru tersebut.
b. Proses penyesuaian sebuah sistem yang ada ke perubahan dalam lingkungannya atau perubahan teknologi tanpa harus mengubah keseluruhan fungsionalitasnya.
c. Modifikasi dan pengembangan kedepan dari sebuah sistem yang ada.
d. Memperbaiki sebuah sistem melalui *reverse engineering* (dan *restructuring*) diikuti oleh *forward engineering*.

Beberapa manfaat *software reengineering*, antara lain:

a. Mengurangi resiko, terdapat resiko tinggi dalam pengembangan peranti lunak baru; memungkinkan timbul permasalahan pengembangan, masalah staf, dan masalah spesifikasi.
b. Mengurangi biaya, yangmana biaya rekayasa ulang sering signifikan lebih rendah dari biaya pengembangan peranti lunak baru.

Software reengineering mempunyai dua (2) tahap utama, yaitu *forward engineering* dan *reverse engineering*. Chikofsky mendefinisikan *forward engineering* sebagai proses tradisional perpindahan dari abstraksi tingkat tinggi dan lojik, desain yang bebas terhadap imlementasi menjadi implementasi fisik dari sebuah sistem. *Forward engineering* terkadang disebut sebagai reklamasi atau renovasi.

Sommerville menyatakan bahwa pada *forward engineering* akan dilakukan *recover* informasi rancangan dari *source code* yang ada, juga disusun kembali keberadaan sistem untuk memperbaiki kualitas atau performansi keseluruhan sistem dengan menggunakan informasi desain.

Proses *forward engineering* menerapkan prinsip-prinsip *software engineering*, konsep, dan metode-metode untuk menciptakan ulang sebuah *existing application*. Dengan demikian, *forward engineering* mengikuti sebuah urutan dari *requirements* melalui perancangan implementasinya.

3. *REVERSE ENGINEERING*

Tapahan yang mendahului *forward engineering* pada proses *software reengineering* adalah *reverse engineering*. *Reverse engineering* merupakan proses dari menganalisa sebuah sistem subyek untuk mengidentifikasi komponen sistem dan hubungan antar komponen, dan menciptakan representasi dari sistem dalam bentuk lain atau pada sebuah abstraksi dengan tingkat yang lebih tinggi. Dalam cakupan tahap siklus hidup, *reverse engineering* mencakup jangkauan luas, mulai dari *existing implementation*, *recapturing* atau *recreation* dari desain, dan mengartikan *requirements* yang secara nyata diimplementasikan oleh subyek sistem [4].

Dari "*Software Maintenance: Concepts and Practice*" [5], disebutkan beberapa faktor yang dapat memotivasi untuk melakukan *reverse engineering* antara lain seperti berikut ini:

a. Hilangnya atau tidak lengkapnya desain/spesifikasi.
b. Kadaluarsa, tidak sesuai atau hilangnya dokumentasi.
c. Meningkatnya kompleksitas program.
d. Kurang terstrukturnya *source code*.
e. Kebutuhan untuk menerjemahkan program kedalam bahasa pemrograman berbeda.

Reverse engineering seringkali digunakan untuk mendapatkan solusi cepat dalam desain dan pemeliharaan. *Reverse engineering* dapat mengekstraksi informasi desain dari *source code*, tetapi level abstraksi, kelengkapan dokumentasi, tingkatan yangmana *tools* dan analis bekerja bekerja bersama-sama, pengarahan proses adalah sangat bervariasi. Singkatnya, *reverse engineering* dapat digunakan untuk mengekstraksi artefak-artefak yang ada dalam peranti lunak. Artefak-artefak tersebut dapat digunakan untuk membangun kembali peranti lunak ataupun membuat dokumentasi yang *up-to-date* dan akurat terhadap peranti lunak.

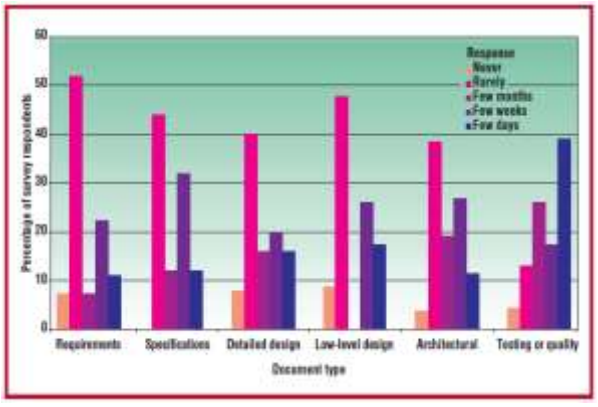
Artefak merupakan istilah yang diadopsi dari istilah Arkeologi, berupa informasi yang digunakan atau dihasilkan oleh proses pengembangan peranti lunak dan untuk menentukan unit dasar abstraksi peranti lunak. Artefak seperti *requirements*, model arsitektur, spesifikasi desain, *source code*, dan *test script*. Artefak dapat menjadi sebuah model, sebuah deskripsi, atau peranti lunak.

Salah satu artefak dari peranti lunak adalah *requirements*. *Requirements* membentuk sebuah fondasi peranti lunak. Dalam *requirements* terkandung sebuah kepentingan, dapat diukur, dan kemampuan diverifikasi, fungsi, properti, karakteristik, atau prilaku bahwa sebuah produk harus memperlihatkan untuk menyelesaikan permasalahan dunia nyata, atau batasan yang harus dipenuhi selama pengembangan sebuah poduk [6]. *Requirements* direpresentasikan dalam dokumen [7]. *Requirements* yang baik dibutuhkan untuk kesuksesan *engineering* dan *reengineering*. Berikut ini akan dipaparkan tentang perolehan *requirements* sebagai artefak hasil proses *reverse engineering*, dikenal sebagai *requirements recovery*.

4. *REQUIREMENTS RECOVERY*

Reverse engineering sering melibatkan sebuah *existing functional system* sebagai subyeknya, hal ini bukanlah sebuah *requirements* [4]. Namun, *requirements* dapat diperoleh dari proses *reverse engineering*. *Requirements recovery* dari sebuah peranti lunak adalah sebuah aktifitas penting dalam merekayasa ulang peranti lunak tersebut. Kesuksesan *engineering* dan *reengineering* membutuhkan *requirements* yang baik.

Requirements sering mengalami perubahan seiring dengan kebutuhan perubahan peranti lunak. Sering perubahan tersebut tidak didokumentasikan menjadikan dokumen tidak akurat, akurasi dokumen terhadap sistem dapat dilihat pada Gambar 1 sebagai hasil riset Lethbridge dkk. Keadaan tersebut mengabaikan hal penting dimana perubahan pada *requirements* terjadi dan berpengaruh pada semua tahapan dari pengembangan, dari pemodelan, perancangan, implementasi, juga untuk pemeliharaan.



Gambar 1: Pembaruan dokumen [8].

Dari peranti lunak, dapat ditemukan *existing requirements* dan motivasi *requirements* tersebut. Perolehan kembali *requirements* (*requirements recovery*) dari peranti lunak dapat memastikan pemahaman lebih baik dari apa yang redundan, apa harus dipertahankan dan apa dapat digunakan kembali. *Requirements* dapat dikumpulkan dari berbagai sumber-sumber berbeda yang berhubungan dengan peranti lunak, misalnya *user*

dan *stakeholder*, bermacam-macam dokumen, *source code*, serta pemicu basis data.

Dari definisi tentang *reverse engineering* dapat diketahui bahwa hasil yang diperoleh dari *reverse engineering* kebanyakan sampai tahap desain. Demikian juga hasil riset, kebanyakan masih pada pengembangan *tool* dan perolehan arsitektur (desain) peranti lunak. Masih sedikit riset untuk memperoleh *requirements* (*requirements recovery*).

Pada 2005, WCRE, terdapat sebuah *workshop* yang diberi nama RETR *Reverse Engineering to Requirements*. Dalam *workshop* itu, E.J.Chikofsky menyebutkan tentang kemungkinan menggunakan *reverse engineering* untuk memperoleh *requirements*. *Requirements* yang diperoleh dapat digabungkan dengan *requirements* yang baru pada proses *forward engineering*. Juga, dari studi empiris menunjukkan bahwa penggabungan dari *user requirement* baru adalah inti permasalahan untuk evolusi peranti lunak dan pemeliharaan [9].

Paper ini memuat riset *requirements recovery* yang termasuk *continuous program understanding* dalam *roadmap* untuk *reverse engineering* yang diberikan oleh Müller dkk [10]. Riset ini telah melakukan komparasi terhadap riset *requirements recovery* milik Yu dkk [11], Rayson dkk [12], Liu dkk [13], El-Ramly dkk [14]. Komparasi tersebut dimuat pada
Tabel *1*, yang juga memuat metode *requirements recovery* dalam paper ini sebagai suatu usulan.

Tabel 1: Komparasi riset requirements recovery.

Researcher / Resource	Yijun Yu, *et al.* [11]	Paul Rayson, *et al.* [12]	Kecheng Liu, *et al.* [13]	Mohammad El-Ramly, *et al.* [14]	*Proposed*
Stakeholders			*checklist*		
User interaction			*model-oriented scenario (behaviour)*	*Interaction-pattern mining*	*analysis with ontology*
User interface					*user & feature identified*
Source code	*refactoring*				
Requirements documents		*semantic parsing*			
Other documents			*context analysis*		

5. ONTOLOGI

Metode *requirements recovery* ini menyertakan ontologi. Ontologi (*ontology*) berasal dari *Greek ontos* + *logos*. Dikenalkan dalam filsafat pada abad ke-19 oleh filsuf Jerman. Ontologi digunakan untuk membedakan studi dari beragamnya ilmu pengetahuan alam. Ontologi secara umum didefinisikan sebagai sebuah spesifikasi eksplisit dari sebuah konseptualisasi. Sebuah konseptualisasi adalah sekumpulan definisi yang mengizinkan satu untuk mengkonstruksi ekspresi tentang beberapa domain aplikasi.

Pandangan sederhana tentang ontologi adalah sekumpulan *terms* memuat *classes*, *relations*, *functions*, dan *instances* [15]. *Class* adalah sebuah pengelompokan konseptual dari obyek serupa. *Relation* digunakan untuk menggambarkan sebuah *relationship* diantara hanya dua *terms*, ini disebut sebuah *slot* atau sebuah relasi biner. *Function* adalah sebuah tipe khusus dari relasi yang menghubungkan sejumlah *terms* ke satu lainnya secara tepat. *Classes* adalah *instances* dari *Class*, *functions* adalah *instance* dari *Function*, dll. Sebuah *instance* harus tidak menjadi kacau dengan sebuah individual karena *instance* mungkin berupa sebuah *class* dimana sebuah individual tidak dapat menjadi sebuah *class*.

6. MEMBANGUN ONTOLOGI DARI *EXISTING SOFTWARE*

Pada metode *requirements recovery* dalam paper ini melakukan identifikasi dan observasi terhadap antarmuka interaktif berupa peranti interaksi yangmana bila diberikan setiap *user entry* menyebabkan sebuah respon (aksi) oleh sistem komputer. Pada antarmuka di layar dapat memuat menu, form isian, dan lainnya. Melalui antarmuka, disampaikan pesan dari perancang ke *user*. Pada pesan memuat konsepsi perancang tentang siapa *user*, apa yang dibutuhkan dan diharapkan oleh *user*, dan bagaimana perancang

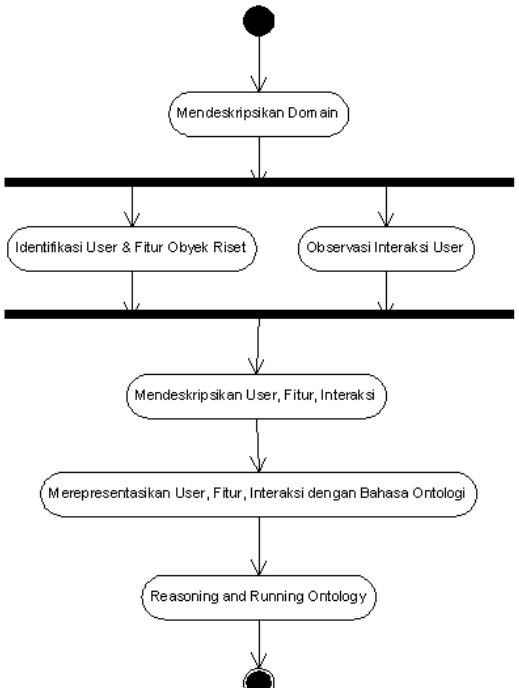
telah menemukan *requirements*. Hasil observasi dan identifikasi tersebut selanjutnya diinterpretasikan dalam ontologi *requirements* dari *existing software*.

Ontologi *requirements* merepresentasikan [16]:
a. Sebuah hirarki fungsional dari sistem peranti lunak tertentu.
b. *Relationships* diantara *functional requirements*.
c. *Attributes* dari *functional requirements*.

Banyak cara mengembangkan ontologi. Terdapat pengembangan ontologi yang dapat dilakukan secara iteratif selama siklus hidupnya, dengan tahapan [16]:
a. Menentukan *domain* dan *scope* ontologi.
b. Mempertimbangkan penggunaan ulang ontologi yang ada.
c. Melakukan *enumerasi term* penting dalam ontologi.
d. Mendefinisikan *classses* dan hirarki *class*.
e. Mendefinisikan properti *classes – slots*.
f. Mendefinisikan facets dari slots.
g. Menciptakan instances.

Paper ini memuat cara untuk membangun ontologi *existing software* dilakukan seperti diagram aktivitas pada

Gambar 2 [17]. Ontologi yang dibangun diharapkan dapat merepresentasikan *functional requirements* yang dimiliki oleh *existing software*.

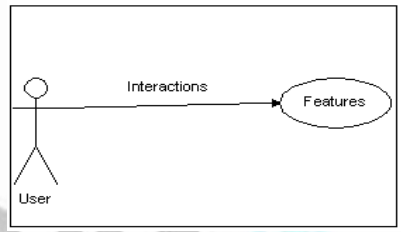


Gambar 2: Aktivitas membangun ontologi.

Aktivitas-aktivitasnya adalah:

a. Mendeskripsikan *domain*.
   Pada awal membangun ontologi ini, diperlukan untuk mendeskripsikan domain. Hal ini untuk pemahaman terhadap konteks obyek riset yang akan dibangun ontologinya.
b. Mengidentifikasi *user*, fitur, dan mengobservasi interaksi *user* dengan fitur.
   Aktivitas berikutnya adalah mengidentifikasi *user* dan juga fitur-fitur yang ada pada *existing software*. *User* dalam hal ini adalah orang-orang yang melakukan interaksi langsung dengan sistem komputer. Dilakukan juga observasi yang terjadi antara *user* dan sistem peranti lunak. Observasi untuk mempelajari interaksi yang terjadi dan makna interaksi tersebut.
c. Mendeskripsikan *user*, fitur, dan interaksi.
   Mendeskripsikan user, fitur, dan interaksi merupakan aktivitas berikutnya. Dilakukan pendeskrisian ini agar tidak terjadi perbedaan memaknai user, fitur, dan interaksi.

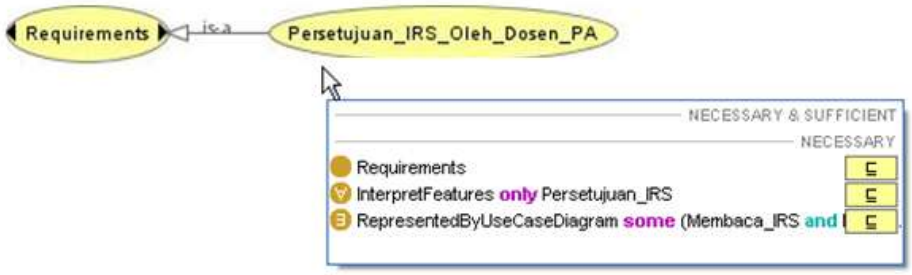
d. Menginterpretasikan fitur, *user*, dan interaksi menggunakan bahasa ontologi.
   Pada metode ini, fitur, *user*, dan interaksi dapat diinterpretasikan seperti pada . Bahasa ontologi yang digunakan adalah OWL DL. Fitur dan *user* direpresentasikan sebagai *classes*. Interaksi direpresentasikan sebagai *property*. *User* sebagai *domain* dan fitur sebagai *range*.



Gambar 3 : Interaksi antara *user* dan fitur.

e. Melakukan *reasoning* dan *running ontology test*.
   Untuk mengetahui kebenaran ontologi yang dibangun pada riset ini dilakukan dengan *reasoning* dan *running test*. *Reasoning* dilakukan menggunakan *reasoner* Pellet.

Dari ontologi yang dibangun, dapat dipandang sebagai *requirements* dari sebuah *existing software*. Contoh *requirements* yang diperoleh dari ontologi divisualisasi pada Gambar 4.



Gambar 4: Requirements ‘Persetujuan IRS Oleh Dosen PA’.

7. PENUTUP

Paper ini memuat metode untuk memperoleh kembali *requirements* (*requirements recovery*) dari *existing software* menggunakan proses *reverse engineering*. *Requirements recovery* bersumber dari *end-to-end interaction*.

Untuk memperoleh kembali *requirements*, digunakan ontologi pada metode yang digunakan. Penggunaan ontologi pada riset ini untuk memperoleh representasi yang benar, konsisten, dan tidak ambigu tentang *existing software*. Dengan demikian *requirements* yang diperoleh adalah *requirements* yang benar.

Requirements yang diperoleh dari metode ini tidak hanya penting untuk *software reengineering* tetapi juga pada *software engineering*. Pada *software reengineering*, *requirements* ini digunakan untuk melengkapi *requirements* pada tahapan *forward engineering*. Pada *software engineering* digunakan dalam fase pemeliharaan yaitu untuk kesesuaian dokumen *requirements* dengan *existing software*.

DAFTAR PUSTAKA

[1] Syed Ahsan Fahmi, and Ho-Jin Choi, "*Software Reverse Engineering to Requirements*", http://www.computer.org, IEEE, 2007.

[2] Elviawaty Muisa Zamzami, dan Eko Kuswardono Budiardjo, "*Green Computing* dan Riset *Requirements Recovery*", Seminar Nasional Ilmu Komputer (SNIKOM) 2010, 29 Oktober 2010, Medan, Indonesia, 2010.

[3] Peter H Feiler, "*Reengineering: An Engineering Problem*", Software Engineering Institute, Special Report CMU/SEI-93-SR-5, 1993.

[4] Elliot J. Chikofsky, and James H. Cross, "*Reverse Engineering and Design Recovery: A Taxonomy*", IEEE Software, pp.13-17, 1990.

[5] Penny Grubb, and Armstrong A. Takang, "*Software Maintenance: Concepts and Practice*", 2nd Edition, World Scientific Publishing Co.Pte.Ltd, pp.143, 2003.

[6] Ronald Kirk Kandt, "*Software Requirements Engineering: Practices and Techniques*", http://www.whalen.ws/index_files/JPL_SW_Reqmts_Engr_D-24994%5B1%5D.pdf, 2003.

[7] IEEE, "*IEEE Standard Glossary of Software Engineering Terminology*", lEEE Std 610.121990, http://ieeexplore.ieee.org/, 1990.

[8] Timothy C. Lethbridge, Janice Singer, and Andrew Forward, "*How Software Engineers Use Documentation: The State of the Practice*", IEEE Software, 2003.

[9] Keith Bennett, and Vaclav Rajlich, "*Software Maintenance and Evolution: A Roadmap*, Conference on The Future of Software Engineering at ICSE", Limerick, Ireland, June 2000, ACM Press, pp. 73-87, 2000.

[10] Hausi A. Müller, et al., "*Reverse Engineering: A Roadmap*", available at http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalmuller.pdf, 2000.

[11] Yijun Yu, et al., "*Reverse Engineering Goal Models from Legacy Code*", available at http://www.cs.utoronto.ca/~alexei/pub/re05re.pdf, 2005.

[12] Paul Rayson, Roger Garside, and Pete Sawyer, "*Recovering Requirements from Legacy Documents*", http://www.comp.lancs.ac.uk/ computing/research/soft_eng/projects/ revere/papers/rgs_icsm99.pdf, 1999.

[13] Kecheng Liu, Albert Alderson, and Zubair Qureshi, "*Requirements Recovery from Legacy Systems by Analysing and Modelling Behaviour*", IEEE International Conference on Software Maintenance, 1999, (ICSM '99) Proceedings, 1999.

[14] Mohammad El-Ramly, Eleni Stroulia, and Paul Sorenson, "*Recovering Software Requirements from System-user Interaction Traces*", ACM, 2002.

[15] Hongji Yang, Zhan Chui, and Paul Obrien, "*Extracting Ontologies from Legacy Systems for Understanding and Re-Engineering*", IEEE, 1999.

[16] Dang Viet Dzung, and Atsushi Ohnishi, "*Improvement of Quality of Software Requirements with Requirements Ontology*", 2009 Ninth International Conference on Quality Software, IEEE, 2009.

[17] Elviawaty Muisa Zamzami, dan Eko Kuswardono Budiardjo, "*Riset Awal: Metode Requirements Recovery Dari Existing Information System Software*", Konferensi Nasional Sistem Informasi (KNSI) 2011, 29 Februari 2011, Medan, Indonesia, 2011.