Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 2, No. 11, November 2018, hlm. 5587-5593 http://j-ptiik.ub.ac.id

Pengembangan Aplikasi Perhitungan Nilai Understandability Berdasarkan Rancangan Perangkat Lunak

Mochammad Adhy[1], Bayu Priyambadha[2], Fajar Pradana[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]adhymch@gmail.com, [2]bayu_priyambadha@ub.ac.id, [3]fajar.p@ub.ac.id

Abstrak

Understandability adalah parameter kualitas perangkat lunak mengenai tingkat kemudahan dalam memahami sebuah modul perangkat lunak yang dikembangkan. Sebagai contoh pentingnya *understandability*, dalam sebuah percobaan mengenai inspeksi kode, 60% dari isu yang dilaporakan oleh *reviewer* profesional pada *maintenance* terkait dengan *understandability*. Oleh karena itu penelitian ini dimaksudkan untuk mengembangkan sebuah aplikasi yang dapat mengukur tingkat *understandability* pada fase perancangan perangkat lunak secara otomatis. Pengukuran *understandability* dilakukan sesuai dengan *multivariate understandability metric*. Pengembangan dilakukan menggunakan metode *waterfall*. Dari hasil penelitian, 100% kebutuhan dapat tervalidasi. Untuk pengukuran efisiensi, diperoleh sistem mampu menyelesaikan 1 pekerjaan pengukuran dalam kurun waktu 10 detik, dan 100% pengguna dapat menyelesaikan tugasnya. Angka tersebut membuktikan sistem berhasil secara efisien membantu mengukur *understandability*. Sedangkan untuk hubungan hasil pengukuran *understandability* dari sistem dengan nilai *maintainability* yang telah diketahui menunjukan korelasi *spearman's rank* sebesar 0.987. Hal tersebut menandakan hasil dari sistem dapat dijadikan salah satu acuan untuk memperkirakan usaha melakukan *maintenance* yang dapat dilakukan sedini mungkin.

Kata kunci: rekayasa perangkat lunak, *maintenance*, *understandability*

Abstract

Understandability is a software quality parameters that shows the level of ease in understanding a developed software module. As an example of the importance of understandability, in an experiment on code inspection, 60% of issues reported by professional reviewers on maintenance are related to understandability. Therefore this research is intended to develop an application that can measure the level of understandability automatically in the software design phase. Measurement of understandability is done in by using multivariate understandability metric. The development is done by adopting the waterfall model methodology. Based on the results, 100% of requirements are validated. As for the system efficiency, system was able to complete 1 measurement task within 10 seconds, and the user can complete 100% of the task, which proved that system is capable to measure understandability efficiently. As for the relationship between understandability results shows by the system and maintainability that has been known shows the spearman's rank correlation of 0.987, which pointing the results of system can be used as one of the reference to estimating the effort to performing maintenance in early stage of software development.

Keywords: *software engineering, maintenance, understandability*

1. PENDAHULUAN

Dalam industri perangkat lunak untuk menjamin dan mengontrol kualitas perangkat lunak membutuhkan alat ukur atau metrik perangkat lunak baik *control metric* maupun *predictive metric*. *Control metric* digunakan untuk mendukung proses manajemen dan *predictive metric* digunakan untuk membantu memprediksi karakteristik dari perangkat lunak (Chidamber & Kemerer, 1994). Salah satu parameter pengukuran kualitas perangkat lunak adalah *understandability* yaitu seberapa tinggi tingkat kemudahan dalam memahami sebuah modul perangkat lunak yang dikembangkan. *Understandability* dapat diukur dengan menghitung derajat kompleksitas, *cohesion* dan

coupling dari perangkat lunak (Nazir, Khan, & Mustafa, 2010).

Parameter kualitas *understandability* berkaitan dengan *maintainability* pada proses *maintenance*. *Maintenance* yaitu tahap untuk menjaga kelangsungan hidup perangkat lunak dengan melakukan modifikasi pada perangkat lunak tersebut dan komponennya setelah ditemukannya kesalahan yang harus dibenahi, atau karena munculnya alasan untuk beradaptasi sesuai perubahan lingkungan. Dalam melakukan *maintenance*, 80% jatah waktu digunakan untuk memahami sebuah perancangan atau modul perangkat lunak dan dokumen terkait (Izadkhah & Hooshyar, 2017). Hal ini dikarenakan dalam praktiknya tidak selalu tim pengembang yang sama yang melakukan perbaikan kesalahan pada perangkat lunak. Jika pengembang sebelumnya tidak ada maka pengembang yang baru atau staff *maintenance* perlu untuk memahami sistemnya terlebih dahulu. Jika sistem sulit dipahami, perubahan yang akan dilakukan bisa saja mengakibatkan kesalahan yang serius dan rentetan perubahan. Hal tersebut dapat menghabiskan banyak biaya dan waktu. Sebagai contoh pentingnya *understandability*, dalam sebuah percobaan mengenai inspeksi kode, 60% dari isu yang dilaporakan oleh *reviewer* profesional pada *maintenance* terkait dengan *understandability* (Uchida & Shima, 2004).

Berdasarkan realita tersebut maka, dalam penelitian kali ini penulis mencoba untuk mengembangkan aplikasi SUMIT (*Software Understandability Measurement Instant Tool*) yang dapat mengukur tingkat *understandability* pada fase perancangan perangkat lunak secara otomatis dengan masukan berupa rancangan perangkat lunak yang telah disimpan kedalam dokumen XML. Dalam pengembangnya penulis menggunakan metode pengembangan *waterfall model*. Secara prinsip *waterfall model* merupakan proses berdasarkan perencanaan, kita harus merencanakan dan menjadwalkan segala proses aktifitas sebelum memulai pelaksanaanya dan dokumentasi diproduksi pada setiap fase (Sommerville, 2011). Metode ini dinilai cocok karena kebutuhan telah dipahami dengan baik dan tidak mungkin untuk terjadi perubahan yang radikal selama proses pengembangan sistem. Sedangkan untuk pengukuran *understandability* pada penulisan kali ini menggunakan karkateristik OO (*object oriented*) yaitu *inheritance*, *coupling* dan *cohesion* dari sebuah rancangan perangkat lunak sesuai dengan persamaan *multivariate undestandability metric*. Pemelihan metrik ini karena *multivariate understandability metric* telah tervalidasi memiliki hubungan korelasi sangat kuat sebesar yaitu 0.94 dengan karakteristik OO yang ada pada *class diagram* (*coupling*, *cohesion*, *inheritance*) (Nazir, Khan, & Mustafa, 2010).

2. DASAR TEORI

2.1. *Waterfall Model*

Model proses pengembangan perangkat lunak yang pertama kali digunakan berasal dari rekayasa sistem yang lebih umum (Royce, 2011). *Waterfall model* dikenal dari bentuknya yang seperti deretan urut dari satu fase ke fase yang lain menyerupai air terjun. *Waterfall model* secara prinsip merupakan proses berdasarkan perencanaan, kita harus merencanakan dan menjadwalkan segala proses aktifitas sebelum memulai pelaksanaanya dan dokumentasi diproduksi pada setiap fase. Permasalahan utama pada *waterfall model* adalah tidak fleksibelnya untuk membagi proyek kedalam tahap yang berbeda. Komitmen harus dilakukan pada tahap awal dalam proses, yang membuat model ini sulit untuk menanggapi perubahan pada kebutuhan pelanggan. Pada dasarnya *waterfall model* hanya harus digunakan pada saat seluruh kebutuhan telah dipahami dengan baik dan tidak mungkin untuk terjadi perubahan yang radikal selama pengembangan sistem (Sommerville, 2011).

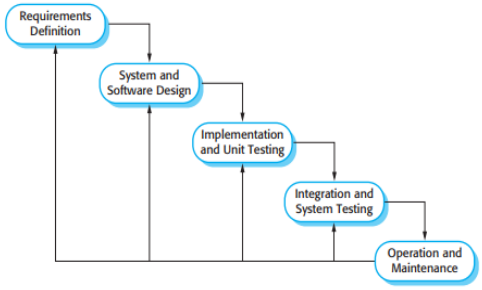
Tahap-tahap utama dalam *waterfall model* mencerminkan secara langsung kegiatan pengembangan mendasar, diantara lain:

1. Analisis kebutuhan dan pendefinisian layanan yang dimiliki sistem, batasan, dan tujuan yang ditetapkan berdasarkan konsultasi dengan pengguna sistem. Setelah ditetapkan secara rinci hal-hal tersebut dipersiapkan sebagai spesifikasi sistem.
2. Perancangan sistem dan sistem perangkat lunak yang mengalokasikan baik kebutuhan perangkat keras maupun perangkat lunak dengan mendirikan arsitektur sistem secara keseluruhan. Perancangan perangkat lunak melibatkan identifikasi dan pengGambaran abstraksi sistem dan hubungan mereka.
3. Implementasi dan pengujian unit, selama tahap ini perancangan perangkat lunak diwujudkan sebagai sebuah rangkaian *program* atau unit-unit *program*. Pengujian

unit melibatkan proses verifikasi bahwa setiap unit telah memenuhi spesifikasinya.

4. Pengujian integrasi dan pengujian sistem dilakukan dengan mengintegrasikan unit-unit *program* dan mengujinya sebagai satu sistem yang lengkap untuk memastikan bahwa kebutuhan perangkat lunak telah dipenuhi. Setelah diuji sistem perangkat lunak akan dikirim kepada pelanggan.
5. *Maintenance* (tidak selalu) biasanya merupakan fase terpanjang dalam siklus hidup perangkat lunak yang dilakukan setelah sistem dipasang dan ditempatkan didalam lingkungan pengguna. Pemeliharaan melibatkan pengkoreksian kesalahan yang tidak ditemukan pada tahap awal pada siklus hidup, meningkatkan implementasi dari unit sistem dan mengembangkan layanan yang dimiliki sistem sebagaimana kebutuhan baru ditemukan.

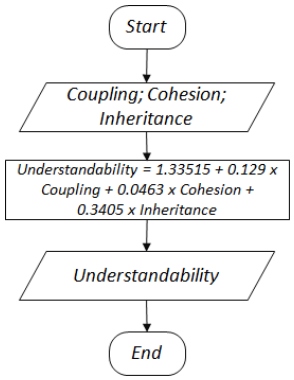
Berikut pada Gambar 1 mengilustrasikan tahapan *waterfall model* secara menyeluruh:



Gambar 1. Tahapan pada Waterfall Model
Sumber: (Sommerville, 2011)

2.2. *Multivariate Understandability Metric*

Multivariate undestandability metric adalah sebuah model regresi linear multi-variabel (*multivariate linear model*) yang digunakan untuk mengukur *understandability* berdasarkan beberapa variabel saling bebas diantaranya *coupling*, *cohesion*, dan *inheritance* yang dinilai mempunyai pengaruh dalam menentukan tingkat *understandability* (variable terikat). Konsep ini digunakan untuk mendapatkan koefisien yang membangun hubungan antara variabel terikat dan variabel saling bebas dimana variabel yang memiliki dampak terhadap *understandability* telah diporsi secara proposional. Dalam model ini nantinya akan dikombinasikan 3 metrik perhitungan yang sesuai dengan atribut atau variabel bebas yang mempengaruhi *understandability*. Ketiga metrik itu adalah *NAssoc* untuk menghitung *coupling*, *NA* untuk menghitung *cohesion*, dan *NOC* untuk menghitung *inheritance* (Nazir, Khan, & Mustafa, 2010). Berikut ini algoritme perhitungan *multivarite undestandability metric* pada Gambar 2:



Gambar 2. Algoritme Multivariate Understandability Metric
Sumber: (Nazir, Khan, & Mustafa, 2010)

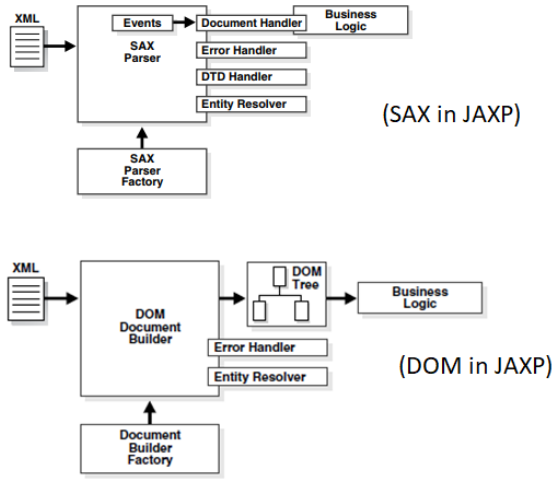
2.3. *Unified Modelling Language* (UML)

UML adalah kumpulan diagram yang digunakan untuk menentukan berbagai aspek seperti kebutuhan dan desain sistem perangkat lunak. UML digunakan sebagai standar notasi untuk memvisualisasikan dan membangun artefak dari sistem perangkat lunak serta membuat bisnis model termasuk sistem lain non-perangkat lunak. Dalam praktiknya UML telah menjadi standar dalam merancang dan mengembangkan sistem berbasis objek (Hamdy, Elsoud, & El-Halawany, 2011). UML diagram terbagi dalam 2 kategori yaitu *structural diagram* dan *behavioral diagram*. *Structural diagram* merepresentasikan aspek statis dari sistem yaitu bentuk atau struktur dari sistem. *Structural diagram* antara lain yaitu *class diagram*, *object diagram*, *component diagram*, *deployment diagram*. Sedangkan *behavioral diagram* merepresentasikan aspek dinamis dari sistem yaitu perubahan karakteristik atau sifat yang terjadi pada sistem. *Behavioral diagram* antara lain yaitu *use case diagram*, *sequence diagram*, *statechart diagram*, *activity diagram*,*collaboration diagram*. (Alhumaidan, 2012).

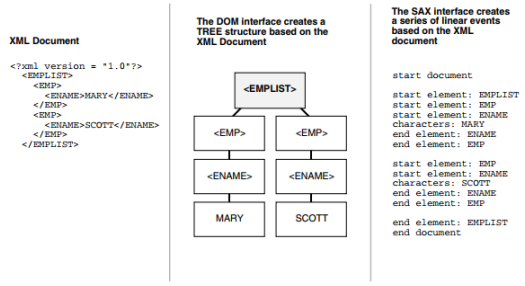
2.4. JAXP XML *Processor*

Pada lingkungan pengembangan menggunakan Java untuk menguraikan dokumen XML dikenal sebuah API yaitu JAXP (Java API for XML Processing). JAXP

menghasilkan antarmuka dan kelas implementasi yang sesuai dengan skema XML. Dengan menggunakan JAXP pengembang yang menggunakan Java dapat melakukan otomatisasi pemetaan antara dokumen XML dan kode Java, yang memungkinkan program menggunakan kode yang dihasilkan untuk membaca, memanipulasi, dan membuat ulang data XML. JAXP API memungkinkan pengguna Java untuk mengimplementasi DOM *parser* atau SAX *parser*. Singkatnya, JAXP memberikan keuntungan untuk pengembangan aplikasi XML pada lingkungan pengembangan menggunakan *Java* (Ashdown, Greenberg, & Melnick, 2014). Artsitektur JAXP adalah sebagai berikut pada Gambar 3 yang menunjukkan bagaimana penggunaan SAX pada JAXP dan DOM pada JAXP. Sedangakan perbandingan hasil penggunaan DOM parser dan SAX parser ketika menggunakan JAXP dapat dilihat pada Gambar 4 :



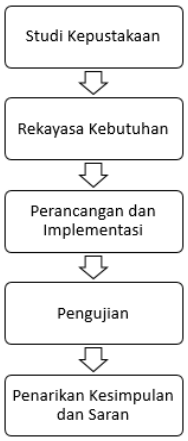
Gambar 3 Arsitektur JAXP
Sumber: (Ashdown, Greenberg, & Melnick, 2014)



Gambar 4 Perbandingan Hasil Penggunaan DOM dan SAX pada JAXP
(Ashdown, Greenberg, & Melnick, 2014)

3. METODOLOGI PENELITIAN

Pada bagian ini dijelaskan tahapan-tahapan dalam pengembangan perangkat lunak SUMIT. Langkah-langkah yang dilakukan antara lain terdiri dari studi kepustakaan, analisis kebutuhan sistem, perancangan sistem dan implementasi, pengujian, kemudian pengambilan kesimpulan dan saran. Dalam penelitian ini penulis juga mengembangkan sebuah perangkat lunak yang metode pengembangannya mengadopsi *waterfall model*. Pada Gambar 6 menunjukkan bagan alur metodologi penelitian ini dilakukan secara *waterfall*.



Gambar 5 Metodologi Penelitian

Pada tahap pertama, studi kepustakaan dilakukan untuk mencari dan mempelajari landasan teori untuk menguatkan penelitian seputar topik yang berkaitan dengan metode pengembangan perangkat lunak, *maintenance* perangkat lunak, urgensi, dan tantangnya, karakteristik pengembangan berbasi objek yang menekan pada *cohesion*, *coupling*, dan *inheritance*, *object oriented metric*, JAXP XML *processor*, *understandability*, dan macam-macam *metric understandability*.

Tahap kedua yaitu melakukan rekayasa kebutuhan. Rekayasa kebutuhan dilakukan menggunakan pendekatan berbasis objek dengan teknik elisitasi kebutuhan menggunakan persona.

Tahap ketiga adalah melakukan perancangan arsitektur dan komponen yang ada pada sistem, perancangan antarmuka sistem, dan perancangan algoritme. Setelah melakukan perancangan, kemudian melakukan implementasi dari hasil melakukan perancangan. Implementasi sistem dibuat menggunakan bahasa *Java*. Dimana sistem yang dibangun akan berjalan sebagai aplikasi *desktop* dan JAXP sebagai XML *document processor*. Sementara itu pengukuran *understandability* dilakukan

sesuai dengan *multivatiare understandabilty metric*.

Pada tahap kelima adalah melakukan pengujian. Pengujian pada penelitian ini meliputi pengujian unit dengan menggunakan teknik *basis path*, pengujian integrasi dengan menggunakan pendekatan *top-down testing*, dan pengujian sistem menggunakan teknik *partitioning equivalence*. Pengujian efisiensi dilakukan dengan mengukur efisiensi berdarkan *time based efficiency* dan *overall relative efficiency* Kemudian melakukan analisis hasil sistem untuk melihat seberapa kuat hubungan hasil *understandability* dengan *maintainability*.

Tahap keenam adalah tahapan terakhir yang dilakukan yaitu penarikan kesimpulan dari hasil penelitian dan saran untuk penelitian kedannya.

4. REKAYASA KEBUTUHAN

Pada tahapan rekayasa kebutuhan dalam penelitian ini dilakukan dengan langkah-langkah melakukan identifikasi masalah seputar pengukuran *understandability* perancangan perangkat lunak, identifikasi aktor, pembuatan persona untuk proses elisitasi kebutuhan dari hasil pengkajian kepustakaan, membuat spesifikasi kebutuhan, pemilihan lingkungan pengembangan, menggambarkan gambaran umum sistem dan memodelkan kebutuhan dengan menggunakan *use case diagram* serta mendeskripsikan perilaku setiap *use case* kebutuhan kedalam *use case scenario*. Dari hasil rekayasa kebutuhan diperoleh kebutuhan fungsional berjumlah 8 buah dan kebutuhan non-fungsional berjumlah 1 buah dan teridentifikasi ada 1 orang aktor yang terlibat dalam sistem perangkat lunak SUMIT. Berikut pada Tabel 1 adalah hasil identifikasi aktor dan Tabel 2 adalah hasil pendefinisan kebutuha fungsional.

Tabel 1 Identifikasi Aktor

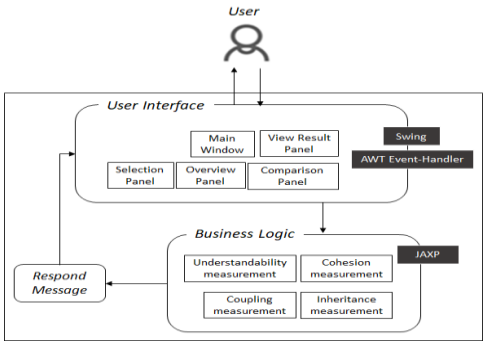
Aktor	Deskripsi
Pengguna	Merupakan individu yang bekerja sebagi *designer* perangkat lunak

Tabel 1 Kebutuhan Fungsional

No	Kebutuhan
1	Seleksi berkas rancangan
2	Ukur *understandability*
3	Ukur porsi *coupling*
4	Ukur porsi *cohesion*
5	Ukur porsi *inheritance*
6	Lihat hasil pengukuran
7	Lihat ikhtisar pengkuran
8	Banding *design*

5. PERANCANGAN & IMPLEMENTASI

Perancangan dilakukan mencakup perancangan arsitektur yang terdiri dari perancangan interaksi antara objek dan perancangan komponen. Hasil dari perancangan arsitektur ini menghasilkan arsitektur sistem sebagai berikut sesuai Gambar 6.



Gambar 6 Arsitektur Sistem

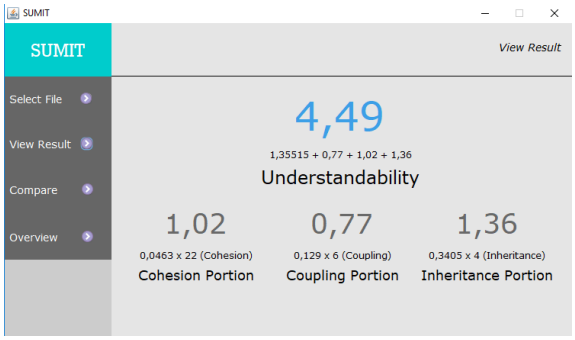
Setelah perancangan arsitektur kemudian dilakukan perancangan algoritme serta pendeskripsian API/*library* yang akan digunakan. Dalam penelitian ini API utama yang digunakan adalah JAXP sebagai XML *processor*. Pada Gambar 7 akan menunjukkan perancangan algoritme operasi *getResult* yang bertujuan untuk mengembalikan nilai *understandability*.

```
FUNCTIONNAME getResult(parameters:
Coup: TYPE double, Coh: TYPE double,
Inherit: TYPE double,) TYPE: double;
TYPE Cov is double;
TYPE Coup is double;
TYPE Coh is double;
TYPE Inheritis double;
TYPE understandability is double;
IF Coup != 0 || Coh !=0 || Inherit !=
0 THEN

understandability = Cov + Coup + Coh
+Inherit;

ELSE understandability = 0 ;
RETURNVALUE understandability;
```

Gambar 7 Algoritme Operasi *getResult*

Setelah proses perancangan selesai maka dilakukan implementasi. Berikut ini adalah hasil implementasi halaman melihat hasil penegukuran *understandability* yang merupakan realisasi dari kebutuhan lihat hasil. Pada Gambar 8 menunjukkan hasil implementasi halaman lihat

hasil pengukuran *understandability*.



Gambar 8 Implementasi Lihat Hasil

6. PENGUJIAN

Pengujian dilakukan kedalam tiga tahapan yaitu pengujian unit, pengujian integrasi, dan pengujian sistem. Pengujian unit dengan menggunakan teknik *basis path* berhasil menguji semua jalur uji, sedangkan pengujian integrasi dengan menggunakan *top-down testing* memberi hasil bahwa semua modul telah terintegrasi sesuai fungsinya, dan pengujian sistem dilakukan dengan teknik *equivalence partitioning* berhasil memvalidasi seluruh kebutuhan fungsional yang berjumlah 8 buah . Sedangkan untuk pengujian kebutuhan non-fungsional yaitu efisiensi, dilakukan terhadap 4 buah *task* yaitu ukur porsi *coupling*, ukur porsi *cohesion*, ukur porsi *inheritance*, dan ukur *understandability*, yang dilakukan lewat 2 kasus uji untuk mengukur berkas rancangan yang sederhana dan berkas rancangan yang kompleks. Hasil efisiensi berdasarkan *time based efficiency* untuk berkas I (rancangan sederhana) sebesar 0,139 *task*/detik ketika menggunakan cara manual dan 0,091 *task*/detik ketika menggunakan SUMIT. Nilai tersebut berarti efisiensi dengan cara manual lebih baik dari pada menggunakan SUMIT. Sedangkan untuk *overall relative efficiency* cara manual dan menggunakan SUMIT sama-sama mengahasilkan nilai 100%. Dan untuk berkas II (rancangan kompleks) diperoleh hasil efisiensi berdasarkan *time based efficiency* sebesar 0,052 *task*/detik ketika menggunakan cara manual dan 0,1 *task*/detik ketika menggunakan SUMIT. Nilai tersebut berarti efisiensi dengan menggunakan SUMIT lebih baik dari pada menggunakan cara manual. Sedangkan untuk *overall relative efficiency* cara manual dan menggunakan SUMIT sama-sama mengahasilkan nilai 100%. Dari hasil yang peroleh dapat disimpulkan bahwa efisiensi otomatisasi perhitungan *understandability* menggunakan SUMIT telah memenuhi kebutuhan non-fungsional dengan mampu menyelesaikan 1 pekerjaan perhitungan dalam waktu 10 detik untuk berkas peracangan yang sederhana maupun yang kompleks. Sedangkan dari analisis hasil korelasi nilai *understandability* yang diperoleh oleh SUMIT terhadap nilai *maintainability* yang telah diketahui dengan menggunakan *spearman's rank corellation* didapat hasil korelasi sebesar 0,987 yang berarti keduanya memiliki korelasi yang sangat kuat. Dari hasil tersebut menandakan bahwa meningkatnya nilai *understandability* turut mempengaruhi meningkatnya nilai *maintainability*. Hasil ini menegaskan agar dalam pengembangan perangkat lunak kita harus lebih menginvestasikan waktu kita pada fase perancangan demi menciptakan rancangan perangkat lunak dengan understandability yang baik agar memudahkan maintenance nantinya. Selain itu hasil ini juga memberikan kesimpulan bahwa nilai hasil perhitungan SUMIT dapat dijadikan salah satu acuan untuk memperkirakan usaha melakukan *maintenance* pada tahap perancangan perangkat lunak.

7. KESIMPULAN

Dari hasil penelitian pengembangan aplikasi perhitungan nilai *understandability* berdasarkan rancangan perangkat lunak diperoleh kesimpulan sebagai berikut:

1. Pada tahap rekayasa kebutuhan menghasilkan 8 buah kebutuhan fungsional dan 1 buah kebutuhan non-fungsional.
2. Pada tahap perancangan dapat menghasilkan perancangan arsitektur, perancangan algoritme dan antarmuka pengguna. Perancangan arsitektur mengasilkan rancangan 9 *sequence diagram* dan *class diagram* yang memuat 5 kelas boundry, 1 buah *interface*, 1 buah *abstract class*, 1 kelas *entity*, dan 5 buah kelas *controller* serta 3 buah kelas yang merepresentasikan penggunaan API/*Library*. Sedangkan pada tahap implementasi berhasil menerapkan rancangan yang dibuat pada tahap perancangan diantaranya arsitektur, algoritme, dan antarmuka pengguna. Penerapan JAXP dinilai berhasil sebagai XML *file processor* untuk melakukan *parsing* dokumen XML.

3. pengujian dilakukan kedalam tiga tahapan yaitu pengujian unit, pengujian integrasi, dan pengujian validasi berhasil memvalidasi seluruh kebutuhan.
4. Berdasarkan *time based efficiency* diperoleh sistem mampu menyelesaikan 1 pekerjaan pengukuran dalam kurun waktu 10 detik, dan berdasarkan *overall relative efficiency* 100% pengguna dapat menyelesaikan tugasnya. Angka tersebut membuktikan sistem berhasil secara efisien membantu mengukur *understandability*.
5. hubungan hasil pengukuran *understandability* menunjukan korelasi *spearman's rank* sebesar 0.987 terhadap *maintainability* yang telah diketahui. Hal tersebut menandakan hasil dari sistem dapat dijadikan salah satu acuan untuk memperkirakan usaha melakukan *maintenance* yang dapat dilakukan sedini mungkin.

DAFTAR PUSTAKA

Alhumaidan, F. (2012). A Critical Analysis and Treatment of Important UML Diagrams Enhancing Modeling Power. *Intelligent Information Management*, 231-237.

Ashdown, L., Greenberg, J., & Melnick, J. (2014). *Oracle XML Developer's Kit Programmer's Guide 11.1 Release.* Oracle.

Chidamber, S., & Kemerer, C. (1994). A Metric Suite for Object Oriented Design. *IEEE Transaction on Software Engineering*, 467-493.

Genero, M., & Mario, P. (2001). A Controlled Experiment for Corraborating The Usefulness of Class Diagram Metrics. *International Journal of Multimedia and Ubiquitous Engineering*, 369-376.

Hamdy, K., Elsoud, M., & El-Halawany, A. (2011). UML-Web Engineering Framework for Modeling Web Application. *Journal of Software Engineering*, 49-63.

Izadkhah, H., & Hooshyar, M. (2017). Class Cohesion Metric for Software Engineering: A Critical Review. *Computer Science Journal of Moldova*, 788-804.

Nazir, M., Khan, R. A., & Mustafa, K. (2010). A Metric Based Model for Understandability Quantification. *Journal of Computing*, 90-94.

Royce, W. (2011). Managing the Development of Large Software Systems: Concepts and Techniques. Dalam I. Sommerville, *Software Engineering 9th Edition* (hal. 30). Boston: Addison-Wesley.

Sommerville, I. (2011). *Software Engineering 9th Edition.* Boston: Addison-Wesley.

Uchida, S., & Shima, K. (2004). An Experiment of Evaluating Software Understandability. *Journal of Systemics, Cybernetics and Informatics*, 7-11.