Development of Software Cost Estimation Application using Constructive Cost Model (COCOMO)

Pitra Aprliawan
Program Studi S1 Informatika
Universitas Sebelas Maret Surakarta
Jln. Ir. Sutami 36A Kentingan, Jebres, Surakarta
pitra.apriliawan@student.uns.ac.id

Ristu Saptono
Program Studi S1 Informatika
Universitas Sebelas Maret Surakarta
Jln. Ir. Sutami 36A Kentingan, Jebres, Surakarta
ristu.saptono@staff.uns.ac.id

Haryono Setiadi
Program Studi S1 Informatika
Universitas Sebelas Maret Surakarta
Jln. Ir. Sutami 36A Kentingan, Jebres, Surakarta
hsd@staff.uns.ac.id

ABSTRACT

The factors of experience and background of a person about software management is one of the causes why difficult to calculate the software estimation. It is required software estimation method that generates the calculation so it is easy to use. Constructive Cost Model (COCOMO) is a software estimation method of producing effort, duration, and resource. COCOMO use Kilo Line of Code (KLOC) as the basis of calculation. FPA is a suitable method to supplement the COCOMO because it can be used to calculate the Line of Code (LOC). This research proposed to build an application that can be used to calculate the software estimation. The method of software development is used SDLC (System Development Life Cycle) with the model of waterfall process. Black Box testing methods using 13 application data, obtained as a result of the value of the Mean Relative Error (MRE) of 3.52%.

Keywords : *Estimaton, COCOMO, FPA, Waterfall, MRE*

ABSTRAK

Faktor pengalaman dan latar belakang seseorang tentang manajemen perangkat lunak merupakan salah satu penyebab mengapa estimasi perangkat lunak sulit dihitung. Diperlukan metode estimasi perangkat lunak yang menghasilkan perhitungan jelas sehingga mudah digunakan. Constructive Cost Model (COCOMO) merupakan metode estimasi perangkat lunak yang menghasilkan effort, duration, dan resource. COCOMO menggunakan Kilo line of Code (KLOC) sebagai dasar perhitungan. FPA merupakan metode yang cocok untuk melengkapi COCOMO karena dapat digunakan untuk menghitung Line of Code (LOC). Dalam penelitian ini diusulkan untuk dibangun sebuah aplikasi yang dapat digunakan untuk menghitung estimasi perangkat lunak. Metode pengembangan perangkat lunak yang digunakan adalah SDLC (System Development Life Cycle) dengan model proses waterfall. Pengujian dengan metode Black Box menggunakan 13 data aplikasi, didapatkan hasil nilai Mean Relative Error (MRE) sebesar 3,52%. Dan mendapatkan Field Testing dari expert yang berpengalaman dibidang manajemen perangkat lunak..

Kata Kunci : *Estimasi, COCOMO, FPA, Waterfall,* MRE

1. PENDAHULUAN

Menghitung estimasi dalam proyek perangkat lunak mempunyai kesulitan tersendiri karena karakteristik proyek perangkat lunak berbeda dengan proyek fisik pada umumnya. Kesulitan-kesulitan yang sering dihadapi dalam estimasi proyek perangkat lunak sangat berkaitan dengan sifat alami software khususnya kompleksitas dan invisibilitas (keabstrakan). Selain itu pengembangan software merupakan kegiatan yang lebih banyak dilakukan secara intensif oleh manusia sehingga tidak dapat diperlakukan secara mekanistik murni [1]. Oleh karena itu faktor pengalaman dan latar belakang seseorang tentang manajemen perangkat lunak merupakan salah satu penyebab mengapa estimasi perangkat lunak sulit dihitung.

Kebingungan selalu melanda perusahaan atau individu setiap kali akan membuat atau mengembangkan suatu perangkat lunak. Mereka tidak mengetahui secara pasti sebesar apa perangkat lunak yang akan dibuat atau dikembangkan, berapa *cost* yang harus dikeluarkan, berapa lama waktu pengerjaan proyek, berapa jumlah *resource* yang harus dikerahkan agar pengerjaan proyek berjalan efektif dan efisien. Sedangkan *software* atau *tool* untuk mengukur estimasi pembuatan atau pengembangan perangkat lunak dari berbagai aspek tersebut sangat sulit untuk dibuat [2].

Metode estimasi perangkat lunak yang menghasilkan perhitungan jelas sehingga banyak dipakai dan umum digunakan adalah *Constructive Cost Model* (COCOMO) [3]. COCOMO pertama kali dikenalkan oleh Barry Boehm pada tahun 1981. Perhitungan paling fundamental dalam COCOMO adalah penggunaan *Effort Equation* untuk mengestimasi jumlah dari *Person-Months* yang dibutuhkan untuk pengembangan proyek. Sehingga COCOMO dapat menghasilkan nilai estimasi dari berbagai aspek, yaitu : *effort, time,* dan *resource.*

Namun dalam metode COCOMO tidak ada perhitungan untuk menghitung *Kilo Line of Code (*KLOC), sehingga diperlukan metode lain yang dapat menghitung KLOC. *Function Point Analysis* (FPA) merupakan metode perkiraan yang cukup populer digunakan dikalangan pengembang karena tingkat akurasi yang tinggi dalam memperkirakan ukuran perangkat lunak [4]. Pendekatan secara tidak langsung yang dilakukan dalam metode FPA memberikan hasil perkiraan dalam satuan FP yang dapat digunakan untuk menghitung *Line of Code* (LOC) dari perangkat lunak [5].

Nilai estimasi effort yang dihasilkan dari COCOMO akan digunakan untuk menghitung nilai estimasi *cost* pada perangkat lunak. Dalam penelitian ini metode *job order costing* digunakan untuk menghitung estimasi *cost* perangkat lunak. Dengan menggunakan metode *job order costing* maka perusahaan dapat menghitung dan mengetahui jumlah biaya yang dikeluarkan untuk suatu pesanan setiap saat, karena di dalam metode *job order costing* biaya produksi masing-masing produk dipisahkan secara jelas sehingga dapat dihitung harga pokok produksi tiap pesanan dengan mudah [6].

Kombinasi metode COCOMO, FPA, dan *Job Order Costing* memiliki langkah perhitungan yang cukup kompleks dan membutuhkan pemahaman lebih tentang estimasi perangkat lunak, sehingga dibutuhkan *tools* bantuan untuk mempermudah proses

estimasi bagi orang-orang yang tidak mendalami estimasi perangkat lunak. Namun, hingga penelitian ini diusulkan belum ada *software* atau *tools* yang tersedia secara gratis yang dapat digunakan untuk melakukan perhitungan estimasi perangkat lunak dengan metode COCOMO dan FPA yang mudah digunakan dan dipahami oleh pengguna yang tidak mendalami bidang manajemen perangkat lunak. Sehingga pada penelitian ini akan dikembangkan sebuah perangkat lunak untuk menghitung estimasi biaya perangkat lunak dengan metode COCOMO. Perangkat lunak yang akan dibangun tidak hanya dapat menghitug estimasi perangkat luank, tetapi juga mudah digunakan oleh pengguna yang tidak mempunyai keahlian di bidang manajemen perangkat lunak. Aspek yang akan dihitung perangkat lunak meliputi size (ukuran) Effort Equation, time (waktu), resource (sumber daya manusia), dan cost (biaya).

2. TINJAUAN PUSTAKA

2.1. *Constructve Cost Model* (COCOMO)

Pada tahun 1981, Barry Boehm mendesain COCOMO untuk memberikan estimasi/perkiraan jumlah person-months untuk mengembangkan suatu produk perangkat lunak. Referensi pada model ini dikenal dengan nama COCOMO 81. Model estimasi COCOMO telah digunakan oleh ribuan manajer proyek suatu proyek perangkat lunak, dan berdasar pada pengalaman dari ratusan proyek sebelumnya. Tidak seperti model estimasi biaya yang lain, COCOMO adalah model terbuka, sehingga semua detail dipublikasikan, termasuk [1]:

1. Dasar persamaan perkiraan biaya
2. Setiap asumsi yang dibuat didalam model
3. Setiap definisi yang dibutuhkan didalam model
4. Biaya yang disertakan dalam perkiraan dinyatakan secara eksplisit

Perhitungan paling fundamental dalam COCOMO model adalah penggunaan Persamaan Usaha untuk mengestimasi jumlah dari *personmonths* yang dibutuhkan untuk pengembangan proyek. Sebagian besar dari hasil-hasil lain COCOMO, termasuk estimasi untuk requirement dan maintenance berasal dari persamaan tersebut [1].

Model COCOMO dapat diaplikasikan dalam tiga tingkatan kelas 7 (mall):

1. Proyek organik, adalah proyek dengan ukuran relatif kecil, dengan anggota tim yang sudah berpengalaman, dan mampu bekerja pada permintaan yang relatif fleksibel.
2. Proyek sedang (semi-terpisah), adalah proyek yang memiliki ukuran dan tingkat kerumitan yang sedang, dan tiap anggota tim memiliki tingkat keahlian yang berbeda.
3. Proyek terintegrasi, adalah proyek yang dibangun dengan spesifikasi dan operasi yang ketat.
4. Model COCOMO dasar ditunjukkan dalam persamaan 1, 2, dan 3 berikut ini (Mall, 2014):

Tabel 1. Koefsien COCOMO

$$\textbf{Effort (E)} = a_b(\text{KLOC})^{b_b}\ \textbf{PM} \quad (2.1)$$

$$\textbf{Tdev} = c_b(E)^{d_b}\ \textbf{Months} \quad (2.2)$$

$$\textbf{Person} = \textbf{Effort} / \textbf{Tdev} \quad (2.3)$$

Dimana :

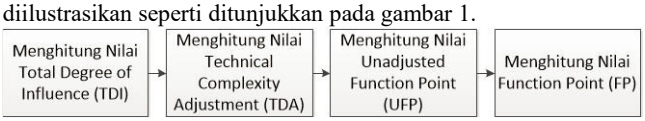
Effort : besarnya usaha (orang-bulan)
Tdev/Duration : lama waktu pengerjaan (bulan)
Person : banyaknya orang yang bekerja
KLOC : estimasi jumlah baris kode

Sedangkan koefisien ab, bb, cb, dan db berikut:

2.2. *Function Point Analysis* (FPA)

Pada pengembangan perangkat lunak, diperlukan adanya estimasi atau perkiraan ukuran perangkat lunak yang akan dibuat untuk menentukan seberapa besar *effort* atau usaha yang akan dikeluarkan untuk membuat perangkat lunak tersebut. *Software Size Estimation* merupakan pembelajaran untuk memperkirakan ukuran suatu perangkat lunak yang akan dikembangkan atau dibuat [2]. Salah satu metode dalam perkiraan ukuran perangkat lunak adalah *Function Point*.

Function Point (FP) merupakan sebuah teknik terstruktur yang memecah sistem menjadi komponen yang lebih kecil dan menetapkan beberapa karakteristik dari sebuah *software* sehingga dapat lebih mudah dianalisis. *Function Point Analysis* (FPA) merupakan metode untuk mengukur pengembangan atau pembuatan perangkat lunak berdasarkan jumlah fungsionalitas dan komplesitas dari pandangan user [8]. FPA diperkenalkan pada tahun 1986 oleh *International Function Point User Group* (IFPUG) [5]. Langkah perhitungan dengan metode FPA diilustrasikan seperti ditunjukkan pada gambar 1.



Gambar 1. Alur Estimasi dengan FPA [2]

2.3. *Function Point* (FP)

Dengan menggunakan data aplikasi, FP dapat digunakan untuk (1) memperkirakan biaya atau usaha yang dibutuhkan untuk perancangan, koding dan *testing* perangkat lunak; (2) memprediksikan jumlah *error* yang mungkin akan dihadapi ketika *testing*, dan (3) memperkirakan jumlah komponen dan atau jumlah baris *source code* pada sistem yang diimplementasikan [10].

Function Point dihitung dengan parameter berikut:

1. Jumlah *external inputs* (EIs). Setiap *external input* berasal dari pengguna atau didapatkan dari aplikasi lain dan menyediakan data yang digunakan untuk mengkontrol informasi. Input biasanya digunakan untuk memperbarui *Internal Logical Files* (ILF). Input harus dibedakan dari *inquiries*, yang nantinya akan dihitung secara terpisah.
2. Jumlah *external outputs* (EOs). Setiap *external output* ditampilkan dalam aplikasi dan menyediakan informasi untuk pengguna. Dalam konteks ini, *external output* dapat berupa laporan, tampilan, pesan *error* dan sebagainya. Item data secara individual didalam laporan tidak dihitung secara terpisah.
3. Jumlah *external inquiries* (EQs). Sebuah *external inquiry* didefinisikan sebagai masukan secara *on-line* yang berdampak pada perubahan secara langsung dan respon *software* dalam *form output on-line* (biasanya diambil dari ILF).
4. Jumlah *internal logical files* (ILFs). Setiap *internal logical file* merupakan kumpulan data logikal yang berada dalam ikatan aplikasi dan diperbarui melalui *external input*.
5. Jumlah *external interface files* (EIFs). Setiap *external interface file* merupakan kumpulan data logikal yang berapa diluar aplikasi namun menyediakan data yang berguna untuk aplikasi.

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Kelima parameter tersebut digunakan untuk menentukan *weight* yang nantinya digunakan untuk menghitung nilai *Unadjusted Function Point* (UFP) yang merupakan nilai *Function*

Point (FP) sebelum disesuaikan dengan nilai *Total Degree of Influences* (TDI) sebagai *Value Adjustment Factor* (VAF) atau *Technical Complexity Adjustmnet* (TCA) yang didapat dari 14 *General System Characteristics* (GSC) [10]. Nilai UFP dicari dengan mengalikan banyak fungsi pada masing-masing parameter dengan nilai *weight*. Penghitungan nilai *weight* ILF, EIF, EI, EQ, dan EO dipengaruhi oleh tingkat kompleksitas komponen-komponen tersebut. Pengkategorian komponen didasarkan pada penghitungan DET, RET, dan FTR dengan rincian [11]:

1. *Data Element Type* (DET) adalah elemen data yang dikenal oleh *user* dan merupakan *non -repeatable data field*.
2. *Record Element Type* (RET) adalah *subgroup* data yang dikenal oleh *user*.
3. *File Type Reference* (FTR) adalah ILF atau EIF yang dibaca atau diakses oleh proses elementer, yaitu EI, EQ, dan EO.

Nilai *weight* didapatkan dengan mengelompokkan ILF, EIF, EO, EI dan EQ ke dalam kelompok *low*, *average*, dan *high* berdasarkan jumlah DET/FTR dan RET. Pengelompokan dilakukan berdasarkan tabel *Complexity Matrix* [11] untuk masing-masing parameter seperti ditunjukan pada tabel 1.

Tabel 2. Complexity Matrix ILF/EIF [11]

ILF/EIF	DET		
RET	1-19	20-50	51+
1	*Low*	*Low*	*Average*
2-5	*Low*	*Average*	*High*
6+	*Average*	*High*	*High*

Setelah dikelompokkan, masing-masing dari kelima parameter dapat dicari nilai *weight* melalui tabel *Function Point Complexity Weight* [7] seperti ditunjukan pada tabel 2.

Tabel 3. Complexity Weight [11]

Component	Low	Average	High
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiris	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

Nilai *Unadjusted Function Point* dihitung dengan mengalikan *weight* yang didapatkan dengan banyak fitur untuk masing-masing parameter.

Setelah didapatkan nilai UFP, kemudian dapat dicari nilai *Function Point* (FP). Untuk menghitung FP, digunakan persamaan 1 [10]:

$$FP = UFP \times [0.65 + 0.01 \times TDI] \quad (1)$$

Total Degree of Influence (TDI) pada persamaan 1 merupkan nilai jumlah derajat kepentingan dari 14 faktor dalam *General System Characteristic* (GSC) yang dibahas pada bagian *Value Adjustment Factor* (VAF). Nilai TDI yang telah dikalikan dengan koefisien 0.01 dan dijumlahkan dengan koefisien 0.65 merupakan nilai *Value Adjustment Factor* (VAF) atau nilai *Technical Complexity Adjustment* (TCA) yang digunakan untuk mengalikan UFP menjadi FP.

2.4. *Value Adjustment Factor* (VAF)

Value Adjustment Factor (VAF) atau *Technical Complexity Adjustment* (TCA) adalah nilai yang dihitung berdasarkan besar pengaruh dari 14 faktor dalam *General System Characteristics* (GSC) dan digunakan untuk mengalikan nilai *Unadjusted Function Point* menjadi *Function Point* sesuai dengan persamaan 1. *General System Characteristics* (GSC) merupakan 14 faktor yang menunjukan karakter sebuah sistem secara umum. Setiap faktor dinilai besar pengaruhnya pada perangkat lunak dengan skala 0 - tidak penting, hingga 5 - sangat penting. Nilai TDI merupakan hasil penjumlahan dari nilai kepentingan masing-masing faktor. Adapun 14 faktor tersebut adalah [12]:

1. *Data Communications* - Didefinisikan sebagai tingkat kebutuhan komunikasi langsung antara aplikasi dengan *processor.*
2. *Distributed Functions* - Didefinisikan sebagai tingkat kebutuhan transfer data antara komponen-komponen aplikasi.
3. *Performances Objectives* - Didefinisikan sebagai tingkat *response time* dan *throughput* yang perlu dipertimbangkan dalam pengembangan aplikasi.
4. *Heavily Used Configuration* - Didefinisikan sebagai tingkat kebutuhan, dimana *setting* konfigurasi komputer berpengaruh terhadap pengembangan aplikasi.
5. *Transaction Rate* - Didefinisikan sebagai tingkat transaksi bisnis yang berpengaruh terhadap pengembangan aplikasi.
6. *On Line Data Entry* - Didenisikan sebagai tingkat kebutuhan *input* data secara interaktif.
7. *End User Efficiency* - Didefinisikan sebagai tingkat kemudahan penggunaan aplikasi.
8. *On Line Update* - Didefinisikan sebagai tingkat kebutuhan ILF di *update* secara interaktif.
9. *Complex Processing* - Didefinisikan sebagai tingkat kesulitan logika proses yang mempengaruhi proses *development.*
10. *Reusability* - Didefinisikan sebagai tingkat kebutuhan aplikasi dan kode program dirancang dan dikembangkan untuk bisa digunakan pada aplikasi lain.
11. *Installation Ease* - Didefinisikan sebagai tingkat kemudahan konversi ke sistem baru yang berpengaruh pada proses *development.*
12. *Operational Ease* - Didefinisikan sebagai tingkat kemudahan aplikasi dalam aspek-aspek operasional, seperti *startup, backup,* dan proses *recovery.*
13. *Multiple Sites* - Didefinisikan sebagai tingkat kebutuhan aplikasi dapat dioperasionalkan pada lingkungan *hardware* dan *software* yang berbeda-beda.
14. *Facilitate Change* - Didefinisikan sebagai tingkat kemudahan aplikasi untuk modifikasi logika proses maupun struktur data.

Pemberian nilai bobot kepentingan dengan besar antara 0 sampai 5 untuk masing-masing faktor mengacu pada dokumen *Standar Documentation* untuk *General System Characteristics* (GSC) yang dipublikasikan oleh *Software Matric* [10] seperti ditunjukan pada tabel 3. Untuk masing-masing faktor, bobot nilai dijelaskan dalam deskripsi untuk menyamakan standar pemberian bobot nilai dan memudahkan untuk digunakan dalam aplikasi berbasis GUI.

Tabel 4. Standar Documentation GSC's [10]

Faktor	Data Communication
Deskripsi	
Informasi data dan kontrol yang digunakan dalam aplikasi dikirim atau diterima melalui fasilitas komunikasi. Koneksi lokal antara terminal dengan unit kontrol dianggap menggunakan fasilitas komunikasi. Protokol merupakan kumpulan aturan yang mengijinkan transfer atau pertukaran informasi antara dua sistem atau perangkat. Semua tautan komunikasi data membutuhkan sejenis protokol.	
Nilai	Deskripsi untuk menentukan *Degree of Influences*
0	Aplikasi merupakan kumpulan proses yang diproses dalam satu perangkat yang berdiri sendiri.
1	Aplikasi merupakan kumpulan proses yang memiliki entri data ATAU proses *printing* secara *remote*.

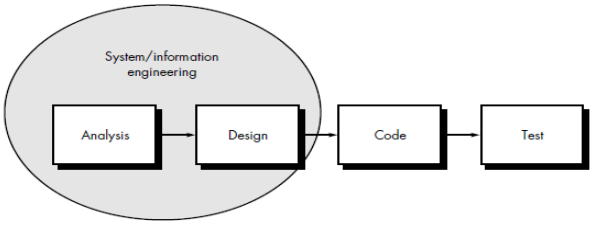
2	Aplikasi merupakan kumpulan proses yang memiliki entri data DAN proses *printing* secara *remote*.
3	Aplikasi termasuk pada pengumpulan data secara *online* atau TP (*teleprocessing*) tampilan antarmuka ke kumpulan proses atau *query sistem*.
4	Aplikasi lebih dari sekedar tampilan antarmuka tapi mendukung hanya satu tipe protokol komunikasi TP.
5	Aplikasi lebih dari sekedar tampilan antarmuka dan mendukung lebih dari satu tipe protokol komunikasi TP.

2.5. Waterfall Model

Metode pengembangan sistem yang penulis gunakan adalah SDLC (System Development Life Cycle) dengan model proses waterfall.

Untuk mengembangkan perangkat lunak dibutuhkan metode atau model pengembangannya, salah satunya adalah model *waterfall.* Metode *waterfall* adalah sebuah metode pengembangan aplikasi dengan pendekatan *sekuensial*.

Pendekatan model ini terlihat mengalir menurun seperti air terjun (*Waterfall*) yang dikembangkan oleh Pressman melalui beberapa tahap. Penggunaan istilah *waterfall* pertama kali dikenal oleh Winston Royce pada tahun 1970.



Gambar 2. Metodologi Penelitian dengan Waterfall Pressman

Metode ini bisa juga disebut dengan *linier sequensial* model, menggunakan pendekatan sistematis dan sekuensial dalam pengembangan aplikasi, dimulai melalui proses komunikasi, perencanaan, pemodelan, konstruksi, dan penyebaran [9].

3. METODOLOGI PENELITIAN

Dalam pengembangan perangkat lunak ini, penulis menggunakan model pendekatan SDLC (*System Development Life Cyrcle*) dengan model *waterfall* [13] yang dikemukakan oleh pressman (2001).

Berikut ini diuraikan proses secara garis besar mengenai tahapan-tahapan siklus SDLC model *waterfall* pada perangkat lunak yang akan dibuat :

1. Analysis (Analisis)

Pada tahap ini, penulis melakukan beberapa aktivitas, yaitu identifikasi masalah, usulan pemecahanan masalah dan analisa kebutuhan sistem. Pemodelan ini diawali dengan mencari kebutuhan dari keseluruhan sistem yang akan diaplikasikan ke dalam bentuk software [14]. Di dalam analisis sistem terdapat tiga langkah dasar yang harus dilakuan, yaitu :

1. Identifikasi masalah, yaitu mengidentifikasi masalah yang didapat dalam aktivitas pengumpulan data sebelumnya.
2. Usulan pemecahan masalah, yaitu mengusulkan pemecahan masalah yang telah diidentifikasi sebelumnya.
3. Software Requirement, yaitu menganalisis kebutuhan software. Bardasarkan identifikasi masalah di atas yang akan dikembangkan secara.

2. Design (Perancangan)

Tahap berikutnya adalah perancangan, pada tahap ini penulis mulai melakukan pemodelan berdasarkan hasil analisis. Perancangan menentukan bagaimana suatu aplikasi menyelesaikan apa yang harus diselesaikan. Pada tahap ini dilakukan pembuatan model dari aplikasi. Maksud pembuatan model ini adalah untuk memperoleh pengertian yang lebih baik terhadap aliran data dan control, proses-proses fungsional, tingkah laku operasi dan informasi-informasi yang terkandung di dalamnya. Proses ini meliputi proses *blueprint* perangkat lunak dengan *Unified Modeling Language* (UML).

3. Code Generation (Pengkodean)

Tahap berikutnya yang dilakukan adalah pemprograman atau *coding*. Pada tahap ini merupakan hasil transfer dari perancangan ke dalam bahasa pemprograman yang telah ditentukan lalu diuji coba dan jika lulus uji coba maka sistem akan diinstal dan di operasikan.

4. Test (Pengujian)

Pada tahap ini adalah pengujian perangkat lunak yang akan dikembangkan. Untuk mendapatkan hasil yang optimal dalam penelitian ini dilakukan tiga macam pengujian, yaitu :

a. Pengujian *Black Box*
   Pengujian dengan cara menguji masing-masing fitur dan fungsi untuk mengetahui apakah dapat bekerja dengan semestinya. Pengujian dengan menggunakan metode pendekatan uji coba *black box* digunakan untuk mendemonstrasikan fungsi *software* yang dioperasikan apakah sudah bekerja dengan semestinya.
b. Pengujian *Performance*
   Pengujian hasil perhitungan yang dilakukan aplikasi menggunakan metode Mean Relative *Error* (MRE). Menunjukkan perbandingan ukuran asli aplikasi dalam SLOC dengan hasil perhitungan menggunakan aplikasi *Function Point Analysis* yang dikembangkan.
c. Pengujian Lapangan (*Field Testing*)
   Pengujian dengan cara melempar perangkat lunak langsung ke lapangan. Pengujian ini bertujuan untuk mengetahui bagaimana tanggapan pengguna akan perangkat lunak yang telah di kembangkan.

5. Support (Pemeliharaan)

Pada tahap ini, merupakan tahap yang perlu dijalankan dalam melakukan pemeliharaan dengan meng-*update* aplikasi dan melakukan maintenance secara berkala agar aplikasi dapat terpelihara dengan baik.

4. HASIL DAN PEMBAHASAN

4.1. Development Life Circle

Pada tahap *Analysis*, ditentukan bahwa data yang dikumpulkan meliputi data untuk membangun kebutuhan sistem dan data untuk melakukan pengujian sistem. Untuk membangun sistem, data didapatkan dari literatur yang membahas tentang metode COCOMO dan metode yang terkait lainnya. Hasil pengumpulan data untuk membangun sistem kemudian disusun menjadi *functional requirement* (Kebutuhan Fungsional) seperti ditunjukan pada tabel 5.

Setelah proses analisis dilakukan tahap berikutnya adalah *design.* Dengan adanya *design* akan memberikan gambaran yang lebih detail tentang perangkat lunak yang akan dibangun.

Tabel 5. Kebutuhan Fungsional

No	Tampilan	Fungsional
1	Home	register login lupa password

		melihat tentang aplikasi
2	Dashboard	tambah project mengunduh contoh data melihat hasil estmasi project melihat status project melihat riwayat project melihat detail project mengunduh project menghapus project keluar
3	TDI	tambah tdi
4	Fungsional	mengisi tabel mengisi fungsional
5	Kompleksitas	megisi kompleksitas
6	Estimasi	menghitung estimasi perangkat lunak menghitung estimasi biaya
7	Overview	melihat project mengedit project melihat fungsional mengedit fungsional melihat tabel mengedit table melihat mengedit tdi melihat kompleksitas mengedit kompleksitas melihat perhitungan estimasi

Dari hasil Tabel 5 kebutuhan fungsional dapat dianalisa estimasi dari perangkat lunak yang akan dibuat. Estimasi yang dilakukan adalah menggunakan data fungsional yang didapat lalu dilakukan estimasi menggunakan metode COCOMO.

Tabel 6. Data Aplikasi Estimasi Perangkat Lunak

Unadjusted Function Point					
No	**Tabel**	**Tipe**	**Bahasa**	**Kolom**	**Relasi**
1	cocomo_user	ILF	SQL	4	2
2	cocomo_project	ILF	SQL	12	2
3	cocomo_gsc	ILF	SQL	3	1
4	cocomo_gscpoint	ILF	SQL	4	1
5	cocomo_tdi	ILF	SQL	4	1
6	cocomo_model	ILF	SQL	3	2
7	cocomo_fp	ILF	SQL	9	2
8	cocomo_ history	ILF	SQL	8	1
No	**Fungsi**	**Tipe**	**Bahasa**	**Data Field**	**Tabel**
9	login	EI	PHP	2	1
10	register	EI	PHP	3	2
11	tambah_project	EI	PHP	2	2
12	tambah_tdi	EI	PHP	14	1
13	tambah_fungsion al	EI	PHP	2	2
14	tambah_kompleksitas	EI	PHP	1	1
15	isi_tabel	EI	PHP	4	1
16	isi_fungsional	EI	PHP	5	1
17	hitung_biaya	EI	PHP	4	5
18	edit detail	EI	PHP	5	22
19	unduh detail	EI	PHP	1	1
20	hapus detail	EI	PHP	1	1
21	keluar	EI	PHP	1	1
22	home	EI	PHP	1	3
23	lihat history	EQ	PHP	7	1
24	lihat detail	EQ	PHP	5	1
25	overview_project	EQ	PHP	12	8
26	dashboard	EQ	PHP	11	8
27	melihat_tentangAplikasi	EQ	PHP	1	1
28	hitung_estimasi	EO	PHP	6	3
Value Adjusment Factor					
No	**Faktor**			**Output**	
1	Data communications			3	
2	Distributed functions			1	
3	Performances objectives			0	
4	Heavily used configuration			0	
5	Transaction rate			2	
6	On-line data entry			5	
7	End-user efficiency			3	
8	On-line update			3	
9	Complex processing			3	
10	Reusability			0	
11	Installation ease			1	
12	Operational ease			3	
13	Multiple sites			3	
14	Facilitate change			5	

Tabel 7. Hasil Perhitungan Estimasi

Function Point	**Line Of Code**	**Effort**	**Duration**	**Person**
125.13 FP	5763 Baris	21.332 PM	7.296 Bulan	2.924 Orang

Dari hasil Tabel 6 dapat dianalisa estimasi *function point* dan *line of code* cukup tinggi sehingga mengakibatkan waktu yang

dibutuhkan untuk membangun aplikasi akan lama. Untuk mengatasi hal tersebut maka solusinya, perangkat lunak akan dibangun menggunakan *framework.* Sehingga diharapkan bisa mempercepat waktu pengembangan perangkat lunak. *Framework* yang digunakan adalah *codeigniter.* Alasannya adalah aturan yang ada dalam *codeigniter* tidak begitu ketat sehingga akan mudah memelajarinya jika sudah menguasai bahasa pemrograman PHP.

Tahap Pengujian merupakan elemen kritis dari jaminan kualitas perangkat lunak dan mempresentasikan kajian pokok dari spesifikasi desain dan pengkodean. Berikut merupakan hasil dari pengujian yang telah dilakukan :

1. Pengujian Black Box

Pengujian dari fitur dan fungsi dari apikasi, menggunakan metode *black box.* Tabel 7 menunjukkan hasil dari pengujian *Black Box.*

Tabel 8. Hasil Pengujian Black Box

No	Tampilan	Hasil Yang Diharapkan	Berhasil	Gagal
1	Home	- Berhasil melakukan register - Berhasil melakukan login - Berhasil mendapat password saat lupa password - Berhasil melihat tentang aplikasi	V V V V	
2	Dashboard	- Berhasil menambah project - Berhasil mengunduh contoh data - Berhasil melihat hasil estmasi project - Berhasil melihat status project - Berhasil melihat riwayat project - Berhasil melihat detail project - Berhasil mengunduh project - Berhasil menghapus project - Berhasil keluar dari aplikasi	V V V V V V V V	
3	TDI	- Berhasil meambah tdi	V	
4	Fungsional	- Berhasil mengisi table - Berhasil mengisi fungsional	V V	
5	Kompleksitas	- Berhasil megisi kompleksitas	V	
6	Estimasi	- Berhasil menghitung estimasi perangkat lunak - Berhasil menghitung estimasi biaya	V V	
7	Overview	- Berhasil melihat project - Berhasil mengedit project - Berhasil melihat fungsional - Berhasil mengedit fungsional - Berhasil melihat table - Berhasil mengedit table - Berhasil melihat - Berhasil mengedit tdi - Berhasil melihat kompleksitas - Berhasil mengedit kompleksitas - Berhasil melihat perhitungan estimasi	V V V V V V V	

2. Pengujian Performance

Pengujian hasil perhitungan yang dilakukan aplikasi, menggunakan metode Mean Relative *Error* (MRE). Tabel 4.6 menunjukkan perbandingan ukuran asli aplikasi dalam LOC dengan hasil perhitungan menggunakan aplikasi *Function Point Analysis* yang dikembangkan.

Tabel 9. Perbandingan Ukuran Asli Software dengan Hasil estimasi

No	Nama *Project*	*Real* LOC	LOC FPA
1	Akomodasi	4089	3891.60
2	Avenger	2468	2340.90
3	Catering	4172	3707.36
4	IMB	3341	3290.40
5	iSpeedy	5310	5296.50
6	Wisuda	4486	4389.60
7	Meeting Room	3520	3482.20
8	Beasiswa	6587	6323.52
9	Bidikmisi	7078	6930.24
10	Penundaan	5213	5045.76
11	Keluhan	4662	4535.04
12	Profile	3264	3265.23
13	RegBidikMisi	6223	5752.32

Tabel 10. Error Relatif Hasil Estimasi

No	*Real* LOC	LOC FPA	|*Real* LOC–LOC FPA|	*Error* Relatif
1	4089	3891.6	197.4	0.04828
2	2468	2340.9	127.1	0.05150
3	4172	3707.36	464.64	0.11137
4	3341	3290.4	50.6	0.01514
5	5310	5296.5	13.5	0.00254
6	4486	4389.6	96.4	0.02149
7	3520	3482.2	37.8	0.01074
8	6587	6323.52	263.48	0.04000
9	7078	6930.24	147.76	0.02088
10	5213	5045.76	167.24	0.03208
11	4662	4535.04	126.96	0.02723
12	3264	3265.23	1.23	0.00038
13	6223	5752.32	470.68	0.07564
Mean Relative Error				0.03517

Dari hasil perhitungan, kemudian dicari selisih antara hasil perhitungan dengan ukuran asli perangkat lunak. Kemudian dicari nilai Relative *Error* untuk masing-masing aplikasi. *Error* yang

didapatkan untuk masing-masing aplikasi ditunjukkan pada Tabel 8.

Dari *error* relatif masing-masing aplikasi, dicari rerata *error* agar didapatkan *Mean Relative Error* (MRE) hasil prediksi. Nilai MRE yang didapatkan untuk hasil prediksi aplikasi *Function Point Analysis* dalam persen adalah 3.52%.

3. Pengujian Lapangan (*Field Testing*)

Pada penelitian ini penulis meminta bantuan expert yang sudah berpengalaman dibidang manajemen perangkat lunak. Yakni Direktur Utama dari PT Global Media Inti Semesta untuk menilai dan memberikan pendapat tentang aplikasi yang berhasil di kembangkan. Aspek yang diberikan penilaian adalah : tampilan, kemanfaatan, dan kemudahan.

4.2. Hasil Implementasi Perangkat Lunak

COCOMO *Estimation* merupakan perangkat lunak (*software*) berbentuk website yang dapat digunakan untuk memperkirakan atau mengestimasi perangkat lunak yang akan dikembangkan. Secara umum, *software* ini membantu anda para pengembang untuk mengestimasi perangkat lunak yang dikembangkan.

Berikut ini merupakan hasil implementasi perangkat lunak yang telah berhasil dibangun :

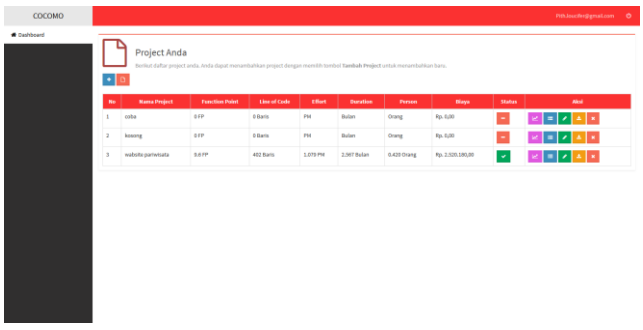
1. Home

Home merupakan tampilan pertama yang muncul apabila membuka aplikasi Estimation COCOMO. Terdapat beberapa fitur yang ada pada tampilan Home, antara lain : login, mendaftar, lupa password, dan tentang aplikasi.Tampilan *Home* ditunjukan pada Gambar 3.

Gambar 3. Tampilan *Home*

2. Dashboard

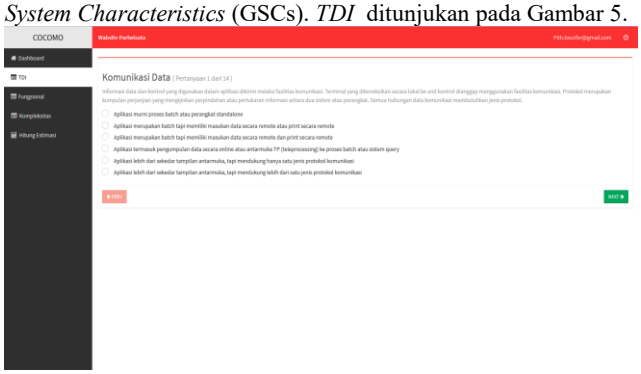
Dashboard merupakan tampilan utama setelah pengunjung login ke dalam aplikasi Estimation COCOMO. Terdapat beberapa fitur yang ada pada tampilan Home, antara lain : tambah project, mengunduh contoh data, melihat hasil estmasi project, melihat status project, melihat riwayat project, melihat detail project, mengunduh project, menghapus project, dan keluar.Tampilan *Dashboard* ditunjukan pada Gambar 4.



Gambar 4. Tampilan *Dashboard*

3. TDI

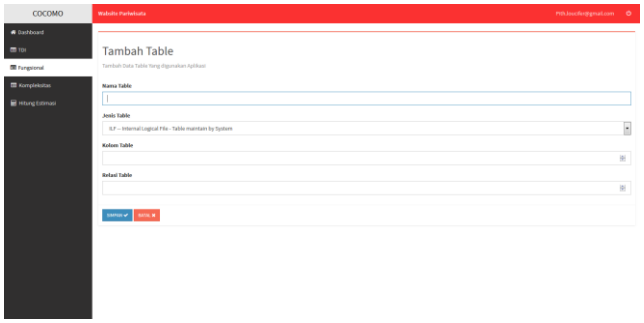
TDI merupakan tampilan untuk menmasukkan TDI dalam aplikasi Estimation COCOMO. Dalam tampilan TDI pengunjung akan diminta melakukan pengisian 14 faktor dalam *General System Characteristics* (GSCs). *TDI* ditunjukan pada Gambar 5.



Gambar 5. Tampilan *Input* Nilai TDI

4. Fungsional

Fungsional merupakan tampilan untuk menmasukkan kebutuhan fungsional dalam aplikasi Estimation COCOMO. Dalam tampilan Fungsional pengguna akan diminta memasukkan fungsional dan tambah tabel. Tambah table ditunjukan pada Gambar 6.



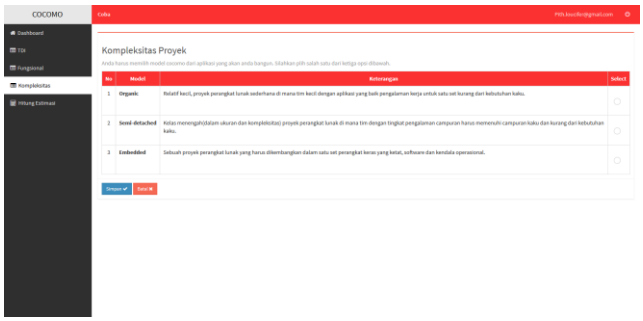
Gambar 6. Tampilan Tambah *Table*

Sedangkan untuk menambah fungsional dapat dilakukan melalui menu Tambah Fungsional. Tampilan untuk menambah fungsional ditunjukan pada Gambar 7.

Gambar 7. Tampilan Tambah Fungsonal

5. Kompleksitas

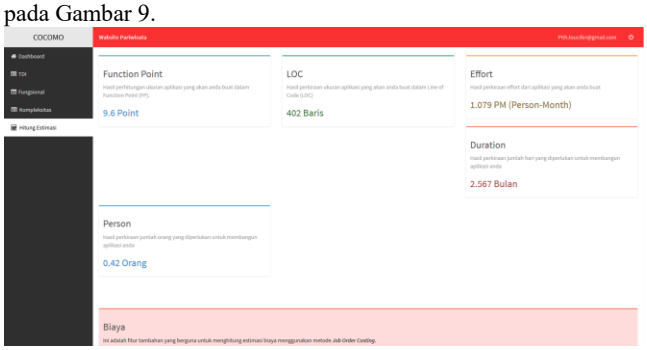
Kompleksitas merupakan tampilan untuk menmasukkan Kompleksitas dalam aplikasi Estimation COCOMO. Dalam tampilan kompleksitas pengguna akan diminta melakukan pilihan kompleksitas dari perangkat lunak yang akan diestimasi, Kompleksitas ditunjukan pada Gambar 8.



Gambar 8. Kompleksitas

6. Hitung Estimasi

Hitung Estimasi merupakan tampilan untuk menghitung estimasi dalam aplikasi Estimation COCOMO. Dalam tampilan kompleksitas pengguna tinggal memilih menu Hitung Estimasi maka estimasi otomatis akan dihitung.Kompleksitas ditunjukan pada Gambar 9.



Gambar 9. Hitung Estimasi

7. Overview

Overview merupakan tampilan yang muncul pengunjung melakukan Tambah Project ke dalam aplikasi Estimation COCOMO. Terdapat beberapa fitur yang ada pada tampilan Overview, antara lain : melihat project, mengedit project, melihat fungsional, mengedit fungsional, melihat table, mengedit table, melihat mengedit tdi, melihat kompleksitas, mengedit kompleksitas, melihat perhitungan estimasi.Tampilan *Overview* ditunjukan pada Gambar 10.

Gambar 10. Tampilan *Overview*

5. PENUTUP

5.1. Kesimpulan

Pengembangan aplikasi estimasi ukuran perangkat lunak dengan metode COCOMO dapat dilakukan dengan menggunakan model *Waterfall*. Aplikasi selesai dibangun setelah melalui tahapan dari model *waterfall*, dimana aplikasi yang dikembangkan dapat menerima input data yang diperlukan dalam proses estimasi berupa nilai TDI, Fungsional, dan Kompleksitas. Nilai TDI dan Function meliputi Tabel dan Fungsional dari software yang akan dibangun. Dari data yang diinputkan oleh pengguna, aplikasi berhasil menghitung estimasi ukuran, waktu, jumlah sumber daya manusia, dan biaya. Testing yang dilakukan pada aplikasi memberikan hasil perkiraan ukuran software memiliki nilai rata-rata relatif error dalam persen adalah sebesar 3,52%.

5.2. Saran

Hasil penelitian ini disebar luaskan sehingga banyak dicoba oleh banyak orang, dan akan mendapatkan *feedback* guna untuk peningkatan kemampuan perangkat lunak ini. Ditambahkan dengan Algoritma *Decision Support System* sehingga akan lebih menyempurnakan perangkat lunak ini pada versi selanjutnya.

6. DAFTAR PUSTAKA

[1] Akbar, A. I. (2011). Estimasi Pembuatan Perangkat Lunak E-Goverment Menggunakan Metode Cocomo.

[2] Saptono, R., & Hutama, G. D. (2015). Peningkatan Akurasi Estimasi Ukuran Perangkat Lunak dengan Menerapkan Logika Samar Metode Mamdani. Scientific Journal of Informatics, 41-52.

[3] Khatibi, V., & Jawawi, D. N. . (2010). Software Cost Estimation Methods : A Review. *Journal of Emerging Trends in Computing and Information Sciences*, *2*(1), 21–29

[4] Jeng, B., Yeh, D., Wang, D., Chu, S.-L., & Chen, C.-M. (2011). A Specific Effort Estimation Method Using Function Point. *Journal of Information Science and Engineering*, *27*, 1363–1376.

[5] Tunali, V. (2014). Software Size Estimation Using Function Point Analysis – A Case Study for a Mobile Application. Muhendisik ve Teknoloji Sempozyumu.

[6] Maulana, R. (2011). Perhitungan Harga Pokok Produksi Dengan Job Order Costing Method Guna Meningkatkan Akurasi Laba Pada Perusahaan Mebel UD. Cipta Jaya Demak.

[7] Mall, R. (2014). Engineering Lesson Basic Issues in Software Engineering.

[8] Georgi, R., & Vogt, T. (2008). Illustrative Example of a Function Point Analysis for The NASA Crew Exploration Vehicle Guidance, Navigation & Control Flight Software. Houston: National Aeronautics & Space Administration.

[9] Pressman, R. S. (2005). Software Engineering A Practicioner's Approach. New York: McGraw-Hill.

[10] Software Matrix. (2016, May 30). General System Characteristics. Diambil kembali dari Software Metrics: http://softwaremetrics.com/pdf/GSC.pdf

[11] Jones, C. (2008). Applied Software Measurement, Global Analysis of Productivity and Quality (3rd ed.). New York: McGraw-Hill.

[12] Pradani, W. (2013). Kajian Metode Perhitungan Metrik Function Point dan Penerapannya pada Dua Perangkat Lunak yang Dipilih . Jurnal Al-Azhar Indonesia Seri Sains dan Teknologi.

[13] Royce, Winston W. (1970). MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS Dr. Winston W. Rovce INTRODUCTION, (August), 1–9.

[14] Hamzah, Saptono, R., & Rini, A. (2016). Development of Software Size Estimation Application using Function Point Analysis (FPA) Approach with Rapid Application ... Development of Software Size Estimation Application using Function Point Analysis (FPA) Approach with Rapid Application Develop, (December). https://doi.org/10.20961/its.v5i2.1988