# Development of Software Size Estimation Application using Function Point Analysis (FPA) Approach with Rapid Application Development (RAD)

Hamzah
Program Studi S1 Informatika
Universitas Sebelas Maret Surakarta
Jln. Ir. Sutami 36A Kentingan, Jebres, Surakarta
hamzah12021@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

Rini Anggrainingsih
Program Studi S1 Informatika
Universitas Sebelas Maret Surakarta
Jln. Ir. Sutami 36A Kentingan, Jebres, Surakarta
rini.anggrainingsih@staff.uns.ac.id

## ABSTRACT

*Software size estimation has an important role on software development life cycle (SDLC) for helping developers to focus on their target and give a control on development process. Estimation needs a method that give high accuracy with minimum result errors that developers can use it as an approach to the real software size they built. Function Point Analysis (FPA) is one of many methods that can estimate the software size. FPA estimate the software size on Function Point that can be used to count the software line of code (LOC). The research aim is to build a software to estimate a software size using FPA method. Rapid Application Development (RAD) is used to develop the software and needs four iteration to complete the development. On 4th iteration, the Black Box Testing method has been used to evaluate the program, using 13 datas of a software that have been completely developed. The result is, mean relative error on the estimation result for the proposed program is 3,52%.*

**Keywords** : *Estimation*, *Function Point Analysis*, *Line of Code*, *Rapid Application Development*.

## ABSTRAK

*Estimasi ukuran perangkat lunak berperan penting dalam proses pengembangan untuk menentukan arah dan tujuan serta menjaga proses pengembangan tetap dalam kendali. Untuk mendapatkan hasil estimasi yang akurat, diperlukan metode yang memiliki akurasi tinggi. Function Point Analysis (FPA) merupakan metode estimasi ukuran perangkat lunak yang memiliki akurasi cukup baik. FPA memperkirakan ukuran perangkat lunak dalam Function Point (FP) yang dapat digunakan untuk menghitung Line of Code (LOC). Dalam penelitian ini diusulkan untuk dibangun sebuah aplikasi yang dapat digunakan untuk memperkirakan ukuran software dengan metode FPA yang dapat digunakan oleh pengembang perangkat lunak. Metode pengembangan perangkat lunak yang digunakan dalam penelitian ini adalah Rapid Application Development (RAD). Dibutuhkan empat iterasi untuk membangun aplikasi dalam penelitian ini. Dilakukan pengujian dengan metode Black Box menggunakan 13 data aplikasi pada iterasi keempat, didapatkan hasil nilai rata-rata error relatif hasil perhitungan yang dilakukan oleh aplikasi adalah sebesar 3,52%.*

**Kata Kunci** : Estimasi, *Function Point Analysis*, *Line of Code*, *Rapid Application Development*.

## 1. PENDAHULUAN

Estimasi atau perkiraan ukuran perangkat lunak dalam proses pengembangan perangkat lunak memegang peranan penting untuk menjaga proses pengembangan tetap dalam kendali pengembang. Estimasi merupakan salah satu tugas paling menantang dalam manajemen proyek [1]. Proyek akan sulit dikendalikan jika sebelumnya tidak memiliki estimasi dan perencanaan [2]. Estimasi dan perencanaan dalam proyek perangkat lunak bisa diawali dengan pengukuran besar dari perangkat lunak yang akan dibuat atau dikembangkan [3]. Salah satu cara untuk memperkirakan ukuran perangkat lunak yang akan dikembangkan adalah dengan menggunakan pendekatan *Function Point Analysis* (FPA).

*Function Point Analysis* (FPA) adalah salah satu metode perkiraan ukuran perangkat lunak yang menggunakan pendekatan secara tidak langsung atau *indirect approach* yang memperkirakan ukuran perangkat lunak dalam satuan *Function Point* (FP) [1]. Metode FPA memperkirakan ukuran perangkat lunak dalam satuan *Function Point* (FP) yang didapatkan dari lima parameter yaitu: *External Input* (EI), *External Output* (EO), *External Inquiries* (EQ), *Internal Logical File* (ILF) serta *External Interface File* (EIF) yang dikelompokkan dalam kelompok *low*, *average*, atau *high* berdasarkan dari banyaknya *Record Element Type* (RET), *Data Element Type* (DET) serta *File Type Reference* (FTR) [1]. Nilai *Function Point* didapatkan dari hasil perkalian antara *Unadjusted Function Point* (UFP) yang merupakan nilai untuk masing-masing komponen fungsi dengan nilai *Technical Complexity Adjustment* (TCA) yang merupakan penyeimbang untuk kompleksitas sitem yang dibangun. Pendekatan secara tidak langsung yang dilakukan dalam metode FPA memberikan hasil perkiraan dalam satuan FP yang dapat digunakan untuk menghitung *Line of Code* (LOC) yang merupakan ukuran besar perangkat lunak dengan mengalikan nilai FP yang didapat dengan nilai *productivity factor* bahasa pemrograman yang digunakan untuk mengembangkan aplikasi [3].

Dalam penelitian ini metode yang digunakan adalah metode FPA standar yang dipublikasikan oleh *International Function Point User Group* IFPUG karena metode ini memiliki nilai akurasi yang tinggi dan cukup populer digunakan dikalangan pengembang, namun standar dalam *Function Point Analysis* yang dikembangkan oleh IFPUG menjadikan metode ini memiliki langkah perhitungan yang cukup kompleks dan membutuhkan pemahaman lebih tentang estimasi ukuran perangkat lunak, sehingga dibutuhkan *tools* bantuan untuk mempermudah proses estimasi bagi orang-orang yang tidak mendalami estimasi ukuran perangkat lunak. Namun, hingga penelitian ini diusulkan belum ada *software* atau *tools* yang tersedia secara gratis yang dapat digunakan untuk melakukan perhitungan perkiraan ukuran perangkat lunak dengan metode FPA yang mudah digunakan dan dipahami oleh pengguna yang tidak mendalami *software size estimation* terutama estimasi dengan metode FPA. Aplikasi yang dikembangkan dalam penelitian ini ditujukan agar dapat

membantu pengembang sistem memperkirakan *software* yang akan mereka bangun dengan mudah.

Untuk mengembangkan aplikasi *Function Point Analysis*, dibutuhkan metode pengembangan yang sesuai dimana aplikasi harus dibangun dalam waktu singkat namun telah memiliki kebutuhan yang jelas. Metode *Rapid Application Development* (RAD) dipilih karena sesuai dengan kebutuhan pengembangan aplikasi yang cepat dan memiliki kebutuhan yang jelas. Model RAD merupakan sebuah adaptasi "kecepatan tinggi" dari model sekuensial linier dimana perkembangan cepat dicapai dengan menggunakan pendekatan kontruksi berbasis komponen. Selain itu jika kebutuhan dipahami dengan baik, maka proses RAD memungkinkan untuk mengembangkan sistem dengan fungsional utuh dalam waktu singkat [12].

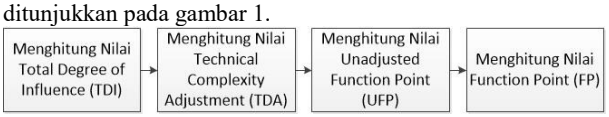
# 2. TINJAUAN PUSTAKA

## 2.1. *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 [5]. FPA diperkenalkan pada tahun 1986 oleh *International Function Point User Group* (IFPUG) [3].

*Function Point Analysis* memperkirakan ukuran perangkat lunak dengan satuan *Function Point* (FP) yang didapatkan dengan langkah yang disadur dari [2]. Langkah perhitungan dengan metode FPA diilustrasikan seperti ditunjukkan pada gambar 1.




**Gambar 1. Alur Estimasi dengan FPA [2]**

## 2.2. *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 [6].

*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.

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) [9]. 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 [7]:

a. *Data Element Type* (DET) adalah elemen data yang dikenal oleh *user* dan merupakan *non -repeatable data field*.
b. *Record Element Type* (RET) adalah *subgroup* data yang dikenal oleh *user*.
c. *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* [7] untuk masing-masing parameter seperti ditunjukan pada tabel 1.

**Tabel 1. Complexity Matrix ILF/EIF [7]**

| 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 2. Complexity Weight [7]**

| 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 [6]:

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

*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.

## 2.3. *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 [8]:

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* [9] 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 3. Standar Documentation GSC's [9]**

| 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. |

## 2.4. *Software Process*

*Software Process* atau disebut juga *Software Development Process* atau *Software Development Life Cycle* merupakan metode yang digunakan untuk mengembangkan sebuah perangkat lunak. *Software Development Process* adalah suatu struktur yang diterapkan pada pengembangan suatu produk perangkat lunak yang bertujuan untuk mengembangkan sistem dan memberikan panduan yang bertujuan untuk menyukseskan proyek pengembangan sistem melalui tahap demi tahap [10].

Rangkaian proses aktivitas pengembangan perangkat lunak disederhanakan menjadi beberapa model proses perangkat lunak diantaranya yaitu *The Waterfall Model, Incremental Development, Reuse-oriented Software Engineering*, dan *Agile Methodology* [11]. *Incremental Development* menggabungkan elemen dalam *Waterfall Model* yang diaplikasikan dalam proses yang berulang. *Rapid Application Development* (RAD) merupakan *Incremental Development* yang menekankan pada siklus pengembangan yang pendek [6].

## 2.5. *Rapid Application Development* (RAD)

*Rapid Application Develompment* (RAD) adalah sebuah model proses perkembangan *software* sekuensial linear yang menekankan siklus perkembangan yang sangat pendek. Model RAD ini merupakan sebuah adaptasi "kecepatan tinggi" dari model sekuensial linier di mana perkembangan cepat dicapai dengan menggunakan pendekatan kontruksi berbasis komponen. Jika kebutuhan dipahami dengan baik, proses RAD memungkinkan tim pengembangan menciptakan "sistem fungsional yang utuh" dalam periode waktu yang sangat pendek (kira-kira 60 sampai 90 hari) [6].

Karena dipakai terutama pada aplikasi sistem konstruksi, pendekatan RAD melingkupi fase – fase sebagai berikut [12]:

*a. Business modeling*

Aliran informasi di antara fungsi – fungsi bisnis dimodelkan dengan suatu cara untuk menjawab pertanyaan – pertanyaan berikut : Informasi apa yang mengendalikan proses bisnis? Informasi apa yang di munculkan? Siapa yang memunculkanya? Ke mana informasi itu pergi? Siapa yang memprosesnya?

Tahapan *Business modeling* dilakukan dengan memodelkan hasil dari pertanyaan pada *business modeling* dalam *Use Case Diagram* dan *Business Process Model Notation* (BPMN).

*b. Data modeling*

Aliran informasi yang didefinisikan sebagai bagian dari fase *bussiness modeling* disaring ke dalam serangkaian objek data yang dibutuhkan untuk menopang bisnis tersebut. Karakteristik (disebut atribut) masing – masing objek diidentifikasi dan hubungan antara objek – objek tersebut didefinisikan.

*Data modeling* digambarkan dalam *data model diagram* kemudian diimpelentasikan pada database.

*c. Proses modeling*

Aliran informasi yang didefinisikan di dalam fase *data modeling* ditransformasikan untuk mencapai aliran informasi yang perlu bagi implementasi sebuah fungsi bisnis. Gambaran pemrosesan diciptakan untuk menambah, memodifikasi, menghapus, atau mendapatkan kembali sebuah objek data.

*Process modeling* dimodelkan menggunakan *Use Case Diagram*, *Sequence Diagram*, *Robustness Diagram*, dan *Class Diagram*.

*d. Aplication generation*

RAD mengasumsikan pemakaian teknik generasi ke empat. Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional, RAD lebih banyak memproses kerja untuk memakai lagi komponen program yang ada (pada saat memungkinkan) atau menciptakan komponen yang bisa dipakai lagi (bila perlu). Pada semua kasus, alat – alat bantu otomatis dipakai untuk memfasilitasi konstruksi perangkat lunak.

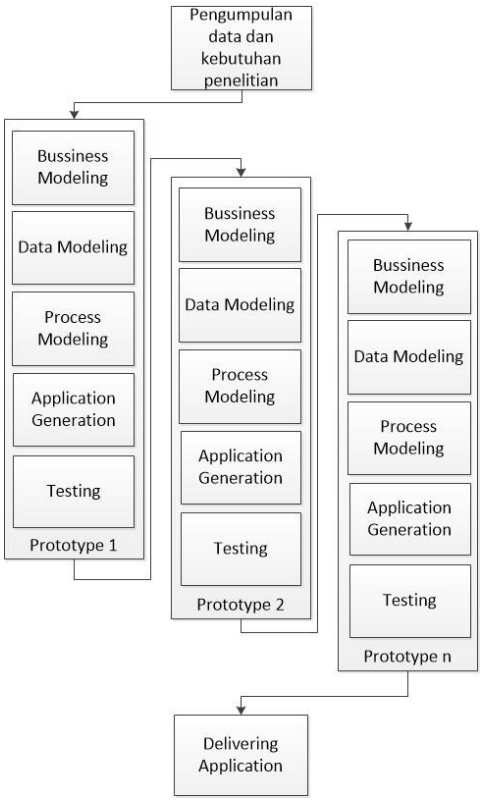
*e. Testing and turnover*

Karena proses RAD menekankan pada pemakaian kembali, banyak komponen program telah diuji. Hal ini mengurangi keseluruhan waktu pengujian. Tetapi komponen baru harus di uji dan semua *interface* harus dilatih secara penuh. Tahapan *testing* akan dilakukan dengan menggunakan metode *Black Box testing* karena metode ini melakukan pengujian untuk *output correctness test*, *reaction time*, dll [13]. Pengujian *Black Box* dilakukan mulai iterasi ke tiga pengembangan aplikasi. Kelima fase dalam RAD diulang dalam siklus iterasi yang dilakukan selama proses pengembangan aplikasi.

RAD dapat dilakukan dalam beberapa cara. Meliputi *Iterative Development*, *System Prototyping* dan *Throwaway Prototyping* [14]. *Itarative Development* memecah keseluruhan proyek menjadi beberapa tahapan versi yang dikembangkan secara berurutan. Kebutuhan paling dasar dan utama dari *software* dikumpulkan pada versi pertama dari sistem. Setelah di implementasikan, *user* dapat memberikan masukan dan timbal balik yang akan digunakan untuk mengembangkan versi aplikasi pada iterasi selanjutnya. *System Prototyping* melakukan pemodelan sistem yang akan dibangun dalam *prototype*. Hal ini dilakukan jika user kesulitan untuk menentukan kebutuhan sistem secara jelas, sehingga diperlukan sebuah *prototype* untuk menggambarkan kebutuhan *user*. *Throwaway Prototyping* melakukan proses *prototype* dalam bentuk desain yang nantinya akan digunakan untuk mencari desain alternatif dari sistem yang akan dibangun. Ketika didapatkan *prototype* yang dikehendaki, maka *prototype* sistem akan dibuang dan sistem baru akan dibangun.

# 3. METODOLOGI PENELITIAN

Langkah yang ditempuh dalam melakukan penelitian ini menggunakan metode *Rapid Application Development* (RAD). Metodologi penelitian yang digunakan dalam penelitian ini diilustrasikan seperti pada gambar 2.




**Gambar 2. Metodologi Penelitian**

## 3.1. Pengumpulan Data dan Kebutuhan Penelitian

Pengumpulan data dilakukan dengan mengumpulkan data dari penelitian yang telah dilakukan sebelumnya. Pada tahapan ini, dilakukan juga studi literatur dengan membaca jurnal penelitian terkait dengan metode *Function Point Analysis*. Metode pengembangan *software* yang dipilih adalah *Rapid Application Development* (RAD) karena menimbang bahwa requirement yang dibutuhkan dalam pengembangan aplikasi sudah cukup jelas dan dibutuhkan waktu yang singkat dalam pengembangan aplikasi.

## 3.2. *Iteratif Development*

Salah satu cara yang pengembanan software dengan RAD adalah dengan menggunakan *Iterative Development*. Cara ini dilakukan dengan mengembangankan software dalam beberapa iterasi yang menghasilkan aplikasi versi tertentu dan akan dievaluasi oleh *user* untuk kemudian diperbarui pada iterasi selanjutnya hingga didapatkan produk yang sesuai dengan kehendak *user*. Kebutuhan paling dasar dari sistem akan dibangun pada versi pertama. Hal yang harus diperhatikan dalam pengembangan *software* dengan metode RAD adalah versi aplikasi yang dikembangkan harus dapat digunakan kembali pada iterasi selanjutnya. Pada penelitian ini, *user* merupakan pengguna yang memahami proyek perangkat lunak.

## 3.3. *Business Modeling*

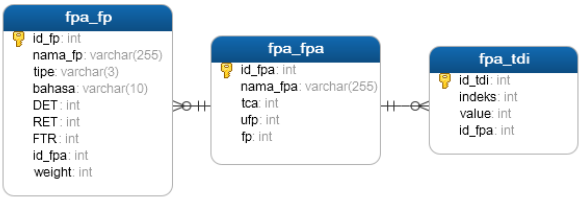
*Business Modeling* dilakukan untuk memodelkan aliran data pada aplikasi yang akan dikembangkan. Pada tahapan ini, analisa dilakukan untuk menjawab pertanyaan berikut terkait dengan proses perhitungan LOC dengan metode FPA:

a. Apakah informasi yang digunakan untuk mengendalikan proses?
b. Apakah informasi yang ditampilkan?
c. Siapa yang menampilkan informasi?
d. Kemana informasi tersebut pergi?
e. Siapa yang memproses informasi?

Hasil analisa dimodelkan dengan diagram *Use Case* dan diagram *Business Process Model Notation* (BPMN)

## 3.4. *Data Modeling*

*Data Modeling* dilakukan untuk menentukan setiap data yang akan digunakan dalam aplikasi dan memodelkan relasi serta atribut dari setiap data yang akan digunakan. Data yang digunakan dan disimpan dalam aplikasi merupakan data yang mendukung proses perkiraan ukuran perangkat lunak dengan metode FPA termasuk hasil perkiraannya dalam bentuk FP dan LOC. *Data modeling* dilakukan dengan membuat *Data Model Diagram* seperti ditunjukan pada gambar 3 dan kemudian diimplementasikan pada *database management system* MySQL.




**Gambar 3. *Data Model Diagram***

## 3.5. *Process Modeling*

*Process Modeling* dilakukan untuk memodelkan setiap proses dan aliran data dalam perangkat lunak untuk memperkirakan ukuran *software* dengan menggunakan data yang telah dimodelkan pada tahapan *data modeling*. *Process modeling* dimodelkan dengan *Sequence Diagram, Robustness Diagram,* dan *Class Diagram.*Proses yang dimodelkan berupa alur perhitungan ukuran *software* dengan metode FPA untuk didapatkan hasil berupa perkiraan besar LOC *software* yang akan dibangun.

## 3.6. *Application Generation*

*Application Generation* merupakan tahapan dimana koding dilakukan sesuai hasil dari *process modeling* dan dengan menggunakan data hasil tahap *data modeling*. Pada tahapan ini, program diimplementasikan dengan bahasa pemrograman PHP dan *Framework* Yii sebagai mesin *generate* kode. Pada tahapan ini, proses perhitungan LOC dengan metode FPA diterapkan dalam aplikasi.

## 3.7. *Testing*

Pada tahap *testing*, versi aplikasi yang telah selesai dibuat akan dievaluasi oleh *user*. Selain itu, *testing* terhadap fungsional juga akan dilakukan menggunakan metode *Black Box Testing* pada iterasi ketiga dan keempat dikarenakan evaluasi oleh *user* tidak dapat menguji tingkat keberhasilan fungsional yang telah dibangun dalam aplikasi. Hasil *testing* dan evaluasi akan digunakan sebagai masukan pada iterasi selanjutnya.

## 3.8. *Delivering Application*

Setelah melalui tahapan-tahapan iterasi dan didapatkan perangkat lunak yang sesuai dengan kebutuhan, maka didapatkan hasil aplikasi yang dikehendaki. Pada tahap ini aplikasi akan diunggah pada website grup riset *Software Architecture and Engineering* Program Studi S1 Informatika UNS.

# 4. HASIL DAN PEMBAHASAN

Pada tahap pengumpulan data dan kebutuhan penelitian, ditentukan bahwa data yang dibutuhkan untuk dikumpulkan meliputi data untuk membangun kebutuhan sistem dan data untuk melakukan pengujian terhadap hasil perhitungan yang akan dilakukan oleh sistem. Untuk membangun sistem, data didapatkan dari literatur yang membahas tentang metode FPA. Hasil pengumpulan data untuk membangun sistem kemudian disusun menjadi kebutuhan dasar sitem seperti ditunjukan pada tabel 4 kolom "*Kebutuhan Dasar*". Untuk melakukan pengujian terhadap hasil perhitungan sistem, diperlukan data dari aplikasi yang telah selesai dikembangkan dan dihitung besar *Software* LOC-nya untuk dibandingkan dengan LOC hasil perhitungan. Didapatkan tujuh data untuk pengujian yang diambil dari penelitian yang dilakukan oleh [2] melalui grup *Software Architecture and Engineering* Jurusan S1 Informatika Universitas Sebelas Maret Surakarta.

Tahapan prototyping merupakan tahap dimana aplikasi dibangun melalui fase-fase dalam metode RAD. Prototyping dilakukan dalam empat iterasi untuk didapatkan aplikasi yang sudah memenuhi kebutuhan dasar. Fase pertama dalam prototyping merupakan *Business Modeling* yang memodelkan proses bisnis didalam aplikasi melalui diagram *Use Case* dan diagram BPMN, hasilnya berupa fungsional yang mengacu pada kebutuhan dasar yang didapatkan pada tahapan pengumpulan data. Kesesuaian antara kebutuhan dasar aplikasi dengan fungsional yang dibangun selama proses pengembangan ditunjukan pada tabel 4.

**Tabel 4. Kesesuaian Fungsional**

| No | Kebutuhan Dasar | Fungsional |
|---|---|---|
| 1. | Menginputkan Nilai TDI | *Input* Nilai TDI |
| | | *Edit* Nilai TDI |
| 2. | Menambah *Function* | Tambah *Function* |
| | | *Edit Function* |
| 3. | Memperkirakan ukuran *software* dalam LOC | Mendaftar |
| | | Menambah *Project* |
| | | Mengedit *Project* |
| | | Memilih *Project* |
| | | Melihat Detail *Project* |
| | | Menghitung LOC |
| | | Melihat Riwayat |

Hasil dari fase *Business Modeling* kemudian digunakan untuk memodelkan data yang digunakan sistem menggunakan diagram *data model* melalui fase *Data Modeling*. Didapatkan tujuh tabel untuk yang digunakan untuk menyimpan data. Ketujuh tabel meliputi tiga tabel utama untuk menyimpan data perhitungan dan hasil perhitungan yaitu *fpa_fpa* untuk menyimpan data aplikasi yang diestimasi dan hasil perhitungan, *fpa_fp* untuk menyimpan data *function* dan *fpa_tdi* untuk menyimpan nilai TDI. Tiga tabel utama dalam sistem seperti ditunjukan pada gambar 3. Selain itu terdapat empat tabel untuk data pendukung yaitu *fpa_user*, *fpa_riwayat*, *fpa_gsc* dan *fpa_gscpoint*.

Fase selanjutnya merupakan fase *Process Modeling* untuk memodelkan proses yang terjadi di dalam sistem melalui *Sequence Diagram* untuk masing-masing fungsional,

*Robustness Diagram* dan *Class Diagram*. Hasil pemodelan proses kemudian diimplementasikan pada fase *Application Generation*.

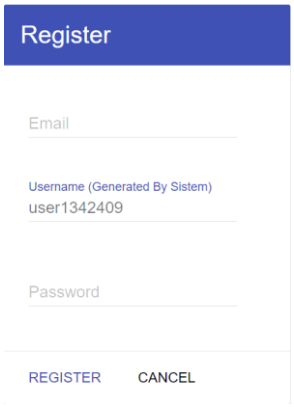
Fase *Application Generation* mengimplementasikan hasil pada fase *Business Modeling*, *Data Modeling* dan *Process Modeling* dalam bentuk program untuk masing-masing fungsional.

1. Mendaftar

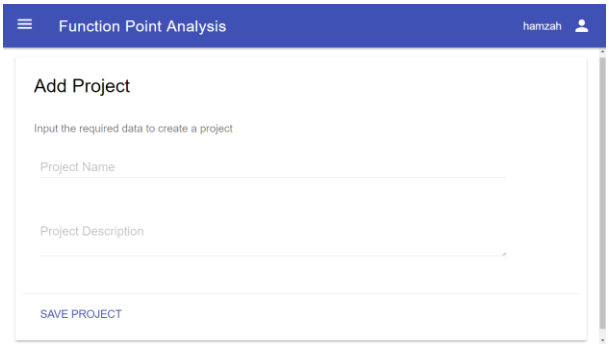
Fungsional Mendaftar digunakan untuk mendaftarkan pengguna agar dapat menggunakan aplikasi ini. Tampilan mendaftar ditunjukan pada gambar 4.

2. Menambah *Project*

Fungsional Menambah *Project* digunakan untuk menambahkan *software* yang akan diestimasi dan menyimpannya dalam bentuk *project*. Istilah *project* digunakan untuk memudahkan pengelolaan *software* yang diestimasi Tampilan menambah *project* ditunjukan pada gambar 5.



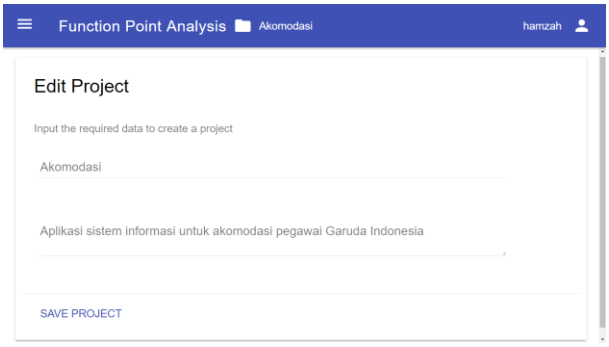

**Gambar 4. Tampilan Mendaftar**




**Gambar 5. Tampilan Fungsional Menambah *Project***

3. Mengedit *Project*

Fungsional Mengedit *Project* digunakan untuk membetulkan kesalahan yang mungkin dilakukan oleh pengguna saat menambah *project*. Tampilan mendaftar seperti ditunjukan pada gambar 4. Tampilan mengedit *project* ditunjukan pada gambar 6.




**Gambar 6. Tampilan Mengedit *Project***

4. Memilih *Project*

Fungsional Memilih *Project* digunakan untuk mengatur *project* mana yang akan diestimasi dengan menampilkan seluruh menu setelah *project* dipilih.

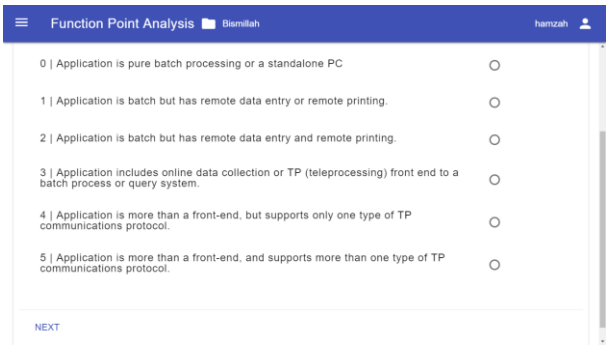
5. Melihat Detail *Project*

Fungsional Melihat Detail *Project* (*Overview*) digunakan untuk melihat deskripsi *project* dan properti yang dimiliki seperti Nilai TDI yang telah diinputkan serta *Table* dan *Function* yang telah ditambahkan. Tampilan melihat detail *project* ditunjukan pada gambar 7.


**Gambar 7. Tampilan *Project Overview***

6. *Input* Nilai TDI

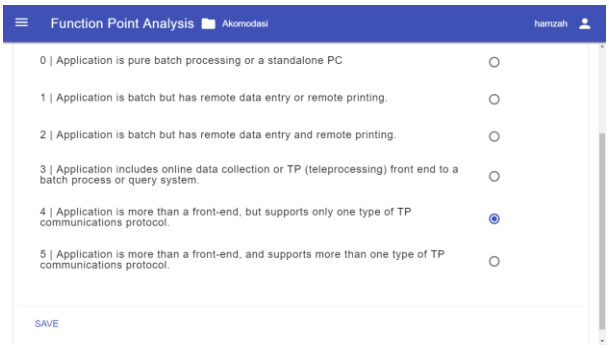
Fungsional Input Nilai TDI digunakan untuk menambahkan nilai TDI pada *project* yang telah dipilih. Tampilan *input* nilai TDI ditunjukan pada gambar 8.




**Gambar 8. Tampilan Input TDI**

7. *Edit* Nilai TDI

Fungsional *Edit* Nilai TDI digunakan untuk membetulkan kesalahan yang mungkin dilakukan oleh pengguna saat menambah nilai TDI atau memang diperlukan untuk mengubah nilai TDI yang kurang sesuai. Tampilan *edit* nilai TDI ditunjukan pada gambar 9.



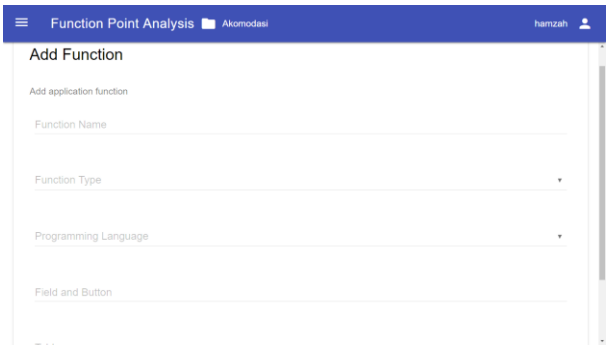

**Gambar 9. Tampilan *Edit* TDI**

8. Tambah *Function*

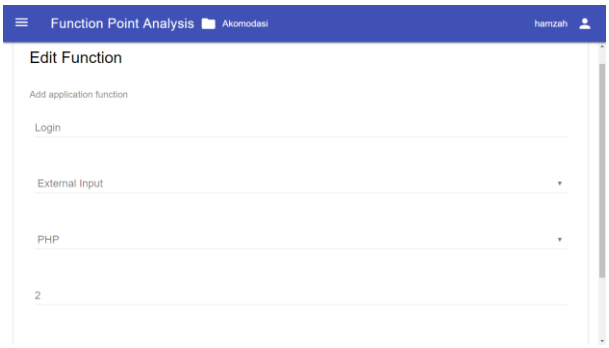
Fungsional Tambah *Function* digunakan untuk menambah *Table* dan *Function* pada *project* yang dikerjakan. Tampilan tambah *function* ditunjukan pada gambar 10.

9. *Edit Function*

Fungsional *Edit Function* digunakan untuk membetulkan kesalahan yang mungkin dilakukan oleh pengguna saat menambah *Table* atau *Function*. Fungsional ini juga dapat digunakan untuk mengubah nilai yang mungkin kurang sesuai atau memang ada perubahan nilai. Tampilan *edit function* ditunjukan pada gambar 11.



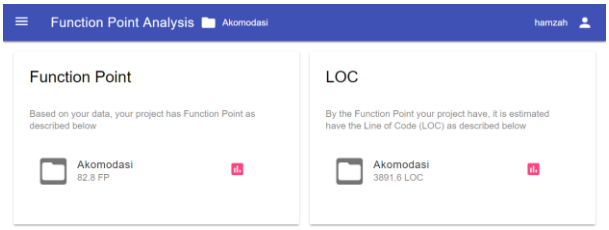

**Gambar 10. Tampilan Tambah *Function***




**Gambar 11. Tampilan *Edit Function***

10. Menghitung LOC

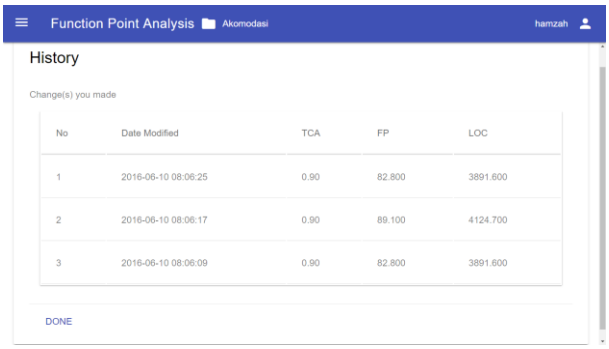
Fungsional menghitung LOC digunakan untuk menghitung besar FP dan banyak LOC dari *project* yang dipilih. Fungsional ini akan menampilkan hasil perhitungan FP dan LOC jika 14 GSC telah dimasukkan nilai TDI nya dan *Function* telah dimasukkan minimal satu untuk masing-masing *Table* dan *Function*. Tampilan menghitung LOC ditunjukan pada gambar 12.




**Gambar 12. Tampilan Menghitung LOC**

11. Melihat Riwayat

Fungsional Melihat Riwayat digunakan untuk melihat perubahan hasil perhitungan FP dan LOC jika dilakukan perubahan pada nilai TDI dan atau penambahan *Table* dan *Function*. Tampilan melihat riwayat ditunjukan pada gambar 13.




**Gambar 13. Tampilan Melihat Riwayat**

Fase terakhir pada tiap iterasi adalah *Testing and Turnover* dimana hasil fase *application generation* diuji dan dievaluasi untuk kemudian diperbaiki pada iterasi selanjutnya. Pengujian dilakukan untuk masing-masing fungsional menggunakan metode *Black Box*, sedangkan pengujian untuk mengetahui besar *error* hasil perhitungan prediksi aplikasi menggunakan *relative error*, hasil pengujian *error* perhitungan dijelaskan pada bagian *Error* Perhitungan.

## 5. *ERROR* PERHITUNGAN

Pengujian hasil perhitungan yang dilakukan oleh aplikasi menggunakan *Mean Relative Error* (MRE). Hasil perhitungan oleh aplikasi didapatkan saat tahapan *testing* pada iterasi ketiga dengan menggunakan metode *Black Box*. Data pengujian yang digunakan pada saat *testing* sebanyak tujuh data dari aplikasi yang telah selesai dikembangkan. Perbandingan antara ukuran asli aplikasi dalam *Software* LOC dengan hasil perhitungan menggunakan aplikasi *Function Point Analysis* yang dikembangkan seperti ditunjukan pada tabel 5.

Dari hasil *testing*, 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 seperti ditunjukan pada tabel 6.

**Tabel 5. Perbandingan Hasil Pehitungan dengan Data Asli**

| Nama *Project* | *Software* LOC | LOC Perhitungan |
|---|---|---|
| Akomodasi | 4089 | 3891.6 |
| Avenger | 2468 | 2340.9 |
| Catering | 4172 | 3707.36 |
| IMB | 3341 | 3290.4 |
| iSpeedy | 5310 | 5296.5 |
| Wisuda | 4486 | 4389.6 |
| MeetingRoom | 3520 | 3482.2 |
| Beasiswa | 6587 | 6323.52 |
| Bidikmisi | 7078 | 6930.24 |
| Penundaan | 5213 | 5045.76 |
| Keluhan | 4662 | 4535.04 |
| Profile | 3264 | 3265.23 |
| RegBidikMisi | 6223 | 5752.32 |

**Tabel 6. *Error* Relatif Perhitungan**

| LOC Asli | LOC FPA | Selisih | *Error Relatif* |
|---|---|---|---|
| 4089 | 3891.6 | 197.4 | 0.048275862 |
| 2468 | 2340.9 | 127.1 | 0.05149919 |
| 4172 | 3707.36 | 464.64 | 0.111371045 |
| 3341 | 3290.4 | 50.6 | 0.015145166 |
| 5310 | 5296.5 | 13.5 | 0.002542373 |
| 4486 | 4389.6 | 96.4 | 0.021489077 |
| 3520 | 3482.2 | 37.8 | 0.010738636 |
| 6587 | 6323.52 | 263.48 | 0.04 |
| 7078 | 6930.24 | 147.76 | 0.020875954 |
| 5213 | 5045.76 | 167.24 | 0.032081335 |
| 4662 | 4535.04 | 126.96 | 0.027232947 |
| 3264 | 3265.23 | 1.23 | 0.000376838 |
| 6223 | 5752.32 | 470.68 | 0.075635546 |
| *Rata-rata error relatif* | | | 0.035174151 |

Dari *error* relatif masing-masing aplikasi, dicari rerata *error* agar didapatkan nilai *Mean Relative Error* hasil prediksi. Nilai MRE yang didapatkan untuk hasil prediksi aplikasi *Function* Point Analysis dalam persen adalah 3,52%.

# 6. PENUTUP

## 6.1. Kesimpulan

Dibutuhkan sebanyak empat iterasi untuk menyelesaikan pembangunan aplikasi estimasi ukuran perangkat lunak dengan metode FPA. Hasilnya didapatkan aplikasi yang dapat memperkirakan ukuran perangkat lunak dalam *Line of Code* (LOC). Untuk memperkirakan ukuran *software*, harus dimasukan nilai TDI dan juga *Function* yang meliputi *Table* dan *Function* dari *software* yang dibangun agar didapatkan perkiraan ukurannya. *Testing* yang dilakukan pada aplikasi memberikan hasil perkiraan ukuran *software* memiliki nilai rata-rata relatif *error* sebesar 3,73%. Sehingga dapat disimpulkan tingkat kepercayaan atau akurasi hasil perhitungan yang dilakukan oleh aplikasi sebesar 96, 27%.

## 6.2. Saran

Hasil penelitian ini dapat dibangun lebih lanjut menjadi aplikasi sistem manajemen pengembangan perangkat lunak apabila dikembangkan untuk dapat memperkiraan besar biaya pengembangan, lama waktu pengerjaan dan banyak sumber daya yang diperlukan. Hal ini dapat dilakukan karena selain memperkirakan dalam satuan LOC, metode FPA dapat digunakan untuk menentukan besar biaya yang diperlukan, lama pengembangan dan sumber daya yang dibutuhkan.

# 7. DAFTAR PUSTAKA


[1] Balaji, N., Shivakumar, N., & Ananth, V. V. (2013). Software Cost Estimation using Function Point with Non. Global Journal of Computer Science and Technology , 1-5.

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

[4] Xia, W., Capretz, L. F., Ho, D., & Ahmed, F. (2008, June). A New Calibration for Function Point Complexity. Electrical and Computer Engineering, 50(7-8), 670-683.

[5] 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.

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

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

[8] 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.

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

[10] Britton, C. (2001). Object-Oriented Systems Development. McGraw-Hill.

[11] Sommerville, I. (2011). Software Engineering. Boston: Pearson Education.

[12] Pressman, R. S. (2002). Rekayasa Perangkat Lunak, Pendekatan Praktisi (Buku Satu). Yogyakarta: Penerbit ANDI.

[14] Dennis, A., Wixom, B. H., & Roth, R. M. (2009). System Analysis and Design Fourt Edition. USA: John Wiley & Sons, Inc.