BAB 2 LANDASAN KEPUSTAKAAN

2.1 Tinjauan Pustaka

Penelitian tentang pengembangan sistem aplikasi telah banyak dilakukan dengan berbagai objek penelitian. Selain itu juga banyak fitur yang ditawarkan untuk setiap sistem aplikasi tersebut. Fitur-fitur ini diharapkan dapat membawa efek positif kedalam organisasi. Pada penelitian ini digunakan beberapa referensi yang nantinya akan menjadi rujukan pada proses penelitian.

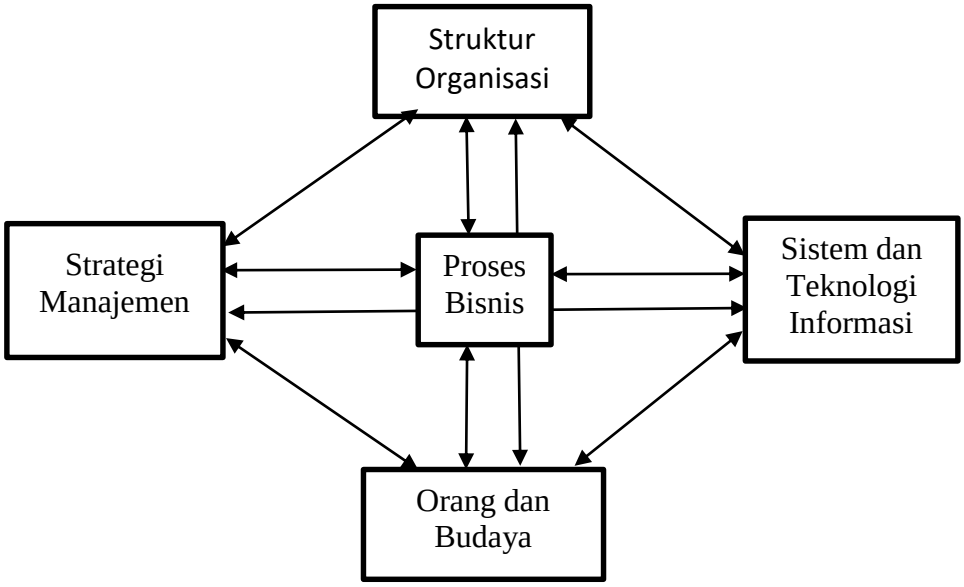
Penelitian pertama adalah penelitian dengan judul *Design and Implementation of Help Desk System on the Effective Focus of Information System*. Penelitian ini dilakukan pada perusahaan Schneider-Electric tepatnya pada divisi *Information Process Organization* (IPO). Divisi ini bertugas sebagai *user support*, yaitu menjawab pertanyaan dari *user* melalui *web* berkenaan dengan perusahaan. Pertanyaan dijawab secara personal melalui *web*. Pada penelitian ini, dikembangkan suatu sistem *help desk* yang akan mengumpulkan pertanyaan-pertanyaan tersebut lalu dirangkum dan dibuat dokumennya. Selain itu pertanyaan personal yang tidak dapat dijawab akan diteruskan ke *user specialist*. Penelitian ini dinilai cukup efisien dengan peningkatan kepuasan konsumen tentang *accessing to help desk team and solution quality* dari 47% menjadi 97% (Serbest, et al., 2015).

Penelitian kedua yaitu *Comparison Analysis of CPU Scheduling : FCFS, SJF, and Round Robin*. Penelitian ini membahas mengenai perbandingan antara ketiga metode tersebut untuk mengetahui *average waiting time*. FCFS dinilai cocok untuk proses dengan *burst time* yang kecil, SJF dinilai cocok untuk kasus penjadwalan dengan prioritas dan *round robin* dinilai cocok untuk menyesuaikan *average waiting time* sesuai yang diinginkan (Siahaan, 2016). Penelitian ketiga yaitu Pengukuran Beban Kerja pada Departemen PPIC di PT.X (Wijaya, et al., 2017). Penelitian ini membahas mengenai perhitungan beban kerja untuk setiap divisi pada PT.X menggunakan metode *Full Time Equivalent* (FTE). Berdasarkan metode FTE tersebut, dapat diketahui apakah beban kerja pada setiap divisi terlalu ringan, normal, atau terlalu berat. Jika beban kerja terlalu berat, akan diberikan saran untuk penambahan jumlah karyawan sesuai nilai FTE.

2.2 Dasar Teori

2.2.1 Sistem Aplikasi

Sistem aplikasi merupakan suatu perangkat lunak yang digunakan untuk mengelola berbagai macam data menjadi informasi yang berguna sesuai dengan tujuan yang diharapkan (Widianti dalam Ibrahim, 2013). Kemampuan komputer yang terus meningkat dan harga yang semakin terjangkau mendorong penggunaan sistem aplikasi dalam kegiatan bisnis. Sehingga tidak heran jika banyak perusahaan yang menggunakan komputer dalam kegiatan bisnisnya (O'Brien dalam Kadir, 2014). Hal-hal yang dulu tidak dapat dilakukan karna proses pengolahan data yang relatif lambat dapat diatasi menggunakan komputer.



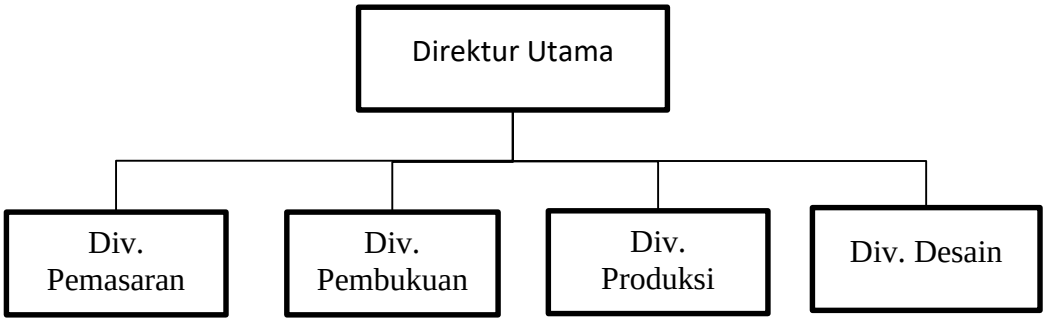
Gambar 2.1 Interaksi antara komponen penting di perusahaan

Sumber : Kadir (2014)

Dapat dilihat dalam Gambar 2.1, bahwa kelima komponen saling berinteraksi dan mempengaruhi. Sebagai contoh, penerapan sistem dan teknologi informasi akan mempengaruhi struktur organisasi, proses bisnis, serta orang dan budaya di organisasi. Perubahan strategi manajemen dapat mempengaruhi struktur organisasi, proses bisnis, serta orang dan budaya di organisasi. Berdasarkan gambar dapat dilihat bahwa sistem dan teknologi informasi mengambil peran penting dalam suatu perusahaan.

2.2.2 PT. Zona Sangangiti Grafika

PT. Zona Sangangiti Grafika adalah sebuah perusahaan percetakan yang beroperasi di kota Pekanbaru, Riau. Perusahaan ini menyediakan jasa untuk mengolah tulisan dan gambar kedalam media cetak. Hasil cetakan dapat berupa buku, kalender, nota, spanduk, dan sebagainya. Mayoritas pelanggan dari perusahaan ini yaitu perusahaan ataupun organisasi lain dengan jumlah pesanan yang cukup besar.



Gambar 2.2 Struktur Organisasi

Gambar 2.2 menjabarkan struktur organisasi pada PT. Zona Sangangiti Grafika yang terdiri dari :

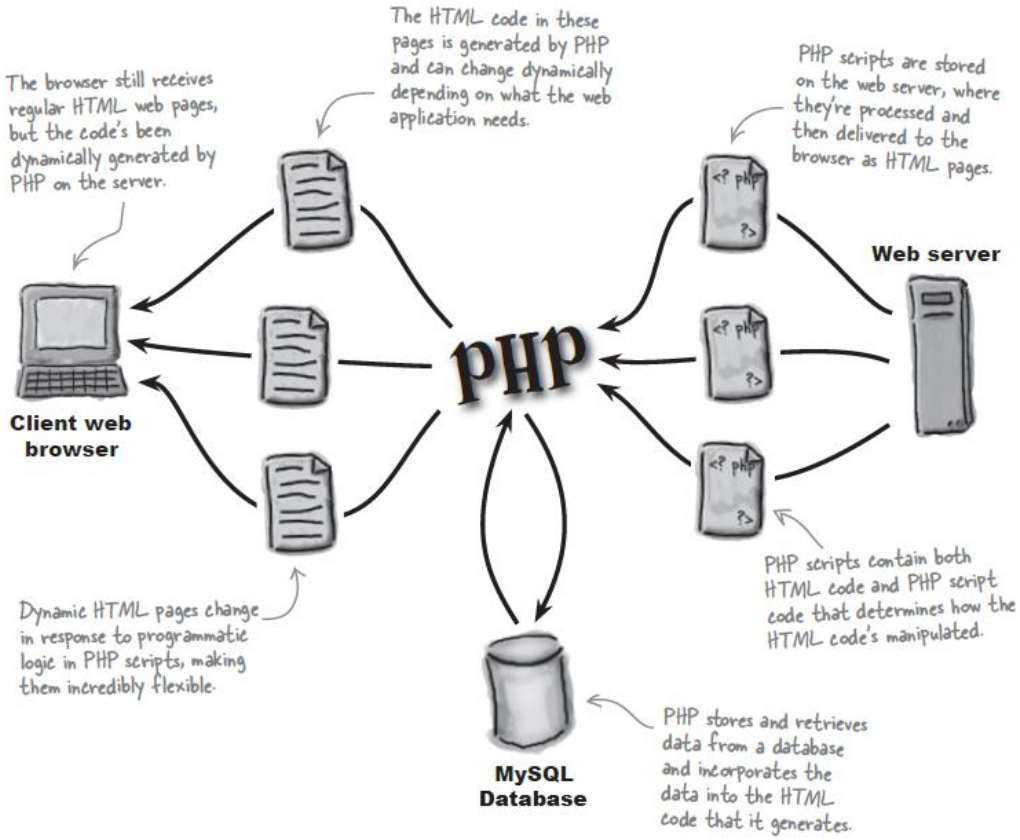
1. Direktur utama, yaitu orang yang berperan sebagai pemilik perusahaan. Pada perusahaan ini direktur utama merangkap sebagai manajer seluruh divisi.
2. Divisi pemasaran, bertugas untuk menjembatani antara perusahaan dengan pelanggan dalam bentuk proses pemesanan, pelayanan, dan sebagainya.
3. Divisi pembukuan, bertugas mengelola keuangan perusahaan baik uang masuk ataupun uang keluar.
4. Divisi produksi, bertugas memproduksi barang sesuai dengan pesanan.
5. Divisi desain, bertugas untuk merancang desain sesuai dengan pesanan.

PT. Zona Sangangiti Grafika menyediakan beberapa jenis jasa percetakan. Jasa tersebut antara lain :

1. Cetak *offset*, yaitu jenis cetakan yang berfokus pada penggunaan kertas. Salah satu contoh hasil dari cetak *offset* yaitu nota dan buku tulis,
2. Cetak *plotter*, yaitu jenis cetakan yang secara umum hasilnya sama seperti printer pada umumnya namun dengan ukuran yang besar. Biasa digunakan untuk pembuatan spanduk dan sebagainya.
3. Cetak *digital print*, yaitu jenis cetakan yang hampir sama dengan cetak plotter namun dengan ukuran yang lebih kecil yaitu maksimal A3+. Cetak digital print hanya dapat mencetak pada kertas, stiker vinyl, ataupun cromo. Biasa digunakan untuk cetak kalender dan sebagainya.

2.2.3 PHP

PHP adalah salah satu bahasa pemrograman yang populer pada beberapa tahun belakangan. Sekitar 60 persen *web server* yang berjalan pada Apache menggunakan PHP. Miliaran *website* dan aplikasi *web* dikembangkan menggunakan bahasa ini. PHP awalnya hanya merupakan pengganti untuk Perl, namun dalam beberapa tahun melonjak penggunaannya (Hayder, 2007). Salah satu alasan mengapa PHP menjadi sangat popular adalah kemudahannya untuk dipelajari. Terutama bagi anda yang familiar dengan sintaks bahasa Java atau C. Siapapun dapat menulis *script* PHP tanpa harus mengikuti banyak aturan. PHP membuat halaman *web* menjadi dinamis. PHP memungkinkan untuk memanipulasi isi halaman *web* pada *server* sebelum dikirim ke *browser* klien. Sehingga perubahan konten halaman *web* dapat dilakukan tanpa harus mengubah *file* secara langsung.



Gambar 2.3 Proses kerja PHP

Sumber: Beighley dan Morrison (2009)

Terlihat dalam Gambar 2.3 bahwa pada PHP, *browser* tetap menerima halaman *web* seperti biasa. Namun, kode nya secara dinamis dihasilkan oleh PHP pada *server*. *Script* PHP disimpan pada *web server*, dimana PHP akan diproses dan dikirimkan ke *browser* dalam bentuk HTML. Disini juga dibutuhkan sebuah *database* sebagai tempat penyimpanan data.

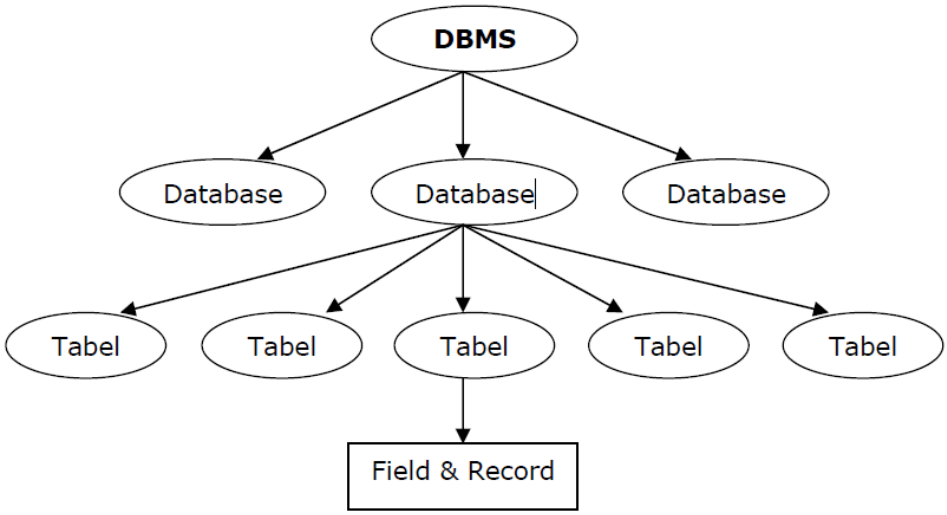
Sejak awal, PHP bukanlah bahasa yang mendukung *object-oriented programming* (OOP). PHP mulai mendukung OOP sejak diluncurkannya PHP 4. Namun, konsep OOP pada PHP 4 dinilai masih kurang lengkap. Sebab, masih ada fitur penting OOP yang tidak berjalan dengan baik. Seperti penanganan variabel internal yang menghasilkan hasil yang tidak diharapkan. Penggunaan PHP dalam konsep OOP mulai popular ketika PHP 5 diluncurkan pada Juli 2004. Disini terjadi perubahan besar-besaran pada *script* sebelumnya. Hal ini menyebabkan objek pada PHP 4 tidak dapat digunakan pada PHP 5. Namun, PHP 5 dinilai dapat menangani penulisan kode dengan konsep OOP.

2.2.4 MySQL

Database adalah sekumpulan data yang disimpan secara sistematik di dalam komputer dan data tersebut dapat ditampilkan menggunakan program komputer tertentu. Program komputer yang digunakan untuk mengelola *database* tersebut biasa disebut *Database Management System* (DBMS). DBMS ini memfasilitasi pengguna untuk melakukan pengelolaan terhadap *database* secara mudah (Solichin, 2010).

Beberapa istilah di dalam *database* antara lain:

1. *Table*: Kumpulan data yang disimpan ke dalam baris dan kolom. Setiap kolom memiliki nama yang spesifik dan unik.
2. *Field*: Kolom dari sebuah tabel. Setiap *field* biasanya memiliki tipe data tertentu yang menjadi acuan untuk data yang akan disimpan.
3. *Record*: Kumpulan nilai yang saling terikat.
4. *Key*: Suatu *field* yang dijadikan kunci dalam operasi tabel. Ada berbagai macam jenis *key* antara lain *Primary Key*, *Foreign Key*, dan *Composite Key*.
5. SQL : Bahasa yang digunakan untuk mengakses *database*.



Gambar 2.4 Hirarki *Database*

Sumber: Solichin (2010)

Gambar 2.4 menjabarkan hirarki *database* secara umum. Hirarki tertinggi yaitu DBMS sebagai pengelola *database*. Tiap DBMS ini dapat menangani satu atau lebih *database* yang berbeda-beda. Setiap *database* dapat memiliki satu atau lebih tabel yang dapat saling berhubungan. Pada setiap tabel terdapat *field* dan *record* yang menjadi penyusun dari sebuah tabel.

MySQL adalah salah satu *software* DBMS yang sangat sering digunakan. MySQL bersifat *open source* dan paling popular di dunia saat ini dengan total pengguna kurang lebih 100 juta orang. MySQL dipilih oleh para pengembang karna dianggap handal, cepat, dan mudah untuk digunakan baik dalam aplikasi *web* maupun *desktop*. Perusahaan besar seperti Yahoo!, Google, Nokia, Youtube, Wordpress, dan Facebook juga pernah menggunakan MySQL sebagai *database* nya (Solichin, 2010).

MySQL memiliki beberapa fitur antara lain *relational database system*, yaitu mendukung hubungan atau relasi antar tabel. MySQL memiliki arsitektur *client-server*, dimana *server database* MySQL terinstal di *server* dan *client* MySQL terinstal di komputer yang sama dengan *server* atau komputer lain yang berkomunikasi dengan *server*. MySQL juga mengenal perintah SQL standar. Selain itu, MySQL mendukung *sub select*, yaitu perintah *select* di dalam *select*, mendukung *foreign key*, bebas untuk digunakan, stabil dan tangguh, dapat menangani berbagai bahasa pemrograman, terdapat banyak dukungan dari berbagai macam komunitas, serta perkembangan *software* yang cukup cepat.

2.2.5 *Object Oriented Programming*

Object Oriented Programming (OOP) adalah jenis pemrograman yang cocok digunakan agar pengelolaan projek menjadi lebih mudah. OOP mengatasi penulisan kode pada projek yang besar tanpa harus memikirkan terlalu banyak tentang pengelolaan kode tersebut. OOP menyediakan fasilitas untuk membuat objek yang dapat di *reuse* sehingga anda ataupun pengembang lain dapat menggunakannya pada projek lain tanpa harus mengembangkannya kembali. OOP menghilangkan kesulitan dalam penulisan dan pengelolaan aplikasi yang besar (Hayder, 2007).

Pada OOP, untuk menggunakan variabel atau fungsi pada kelas, anda harus membuat suatu *instance* dari kelas tersebut, biasa disebut objek. Variabel yang terkait dengan objek disebut properti dan fungsi yang terkait dengan objek disebut *method*. Perbedaan besarnya dengan *procedural programming* yaitu properti dan *method* dari sebuah objek dapat dideklarasikan *protected* atau *private*, sehingga detail dapat disembunyikan dari *end user*.

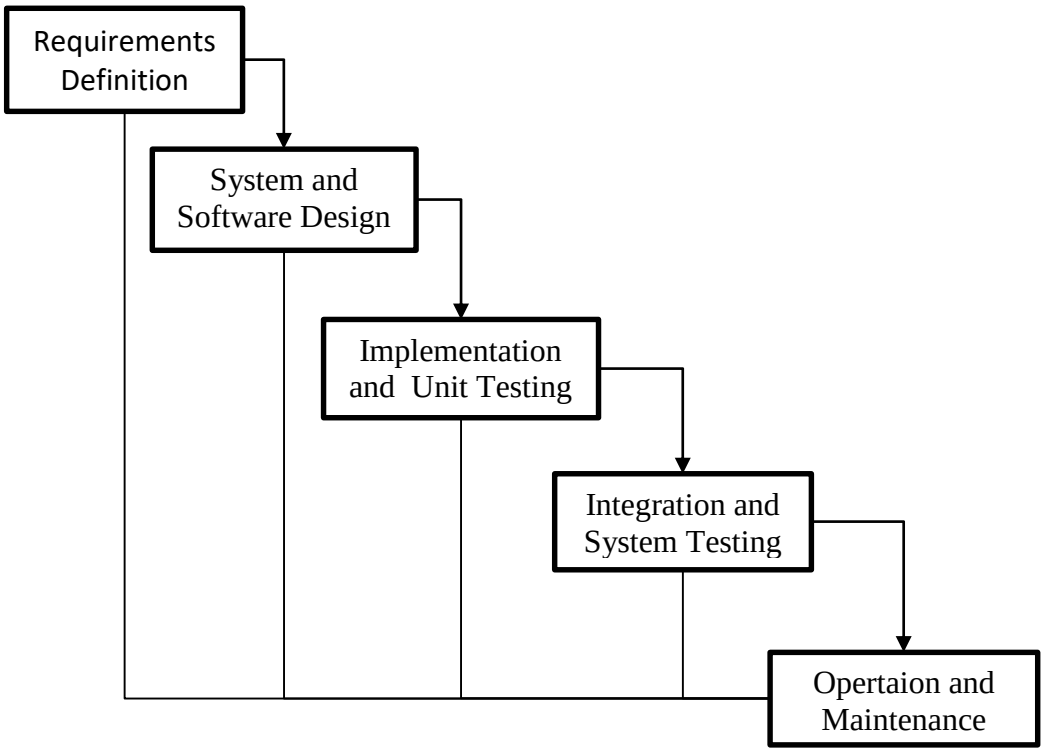
Kemampuan tersebut merupakan salah satu dari tiga ciri khas OOP (Powers, 2008), yaitu:

1. *Encapsulation*: Kemampuan untuk menyembunyikan detail dari *end user* dan mencegah akses secara langsung ke kode.
2. *Polymorphism*: Memungkinkan memberikan nama yang sama untuk *method* atau properti pada kelas yang berbeda. Menerapkan konsep *encapsulation* dimana terdapat penyembunyian detail tentang bagaimana salah satu *method* bekerja dengan cara penyamaan nama *method*.
3. *Inheritance*: Memungkinkan untuk membuat kelas baru yang diturunkan dari kelas yang sudah ada dan secara otomatis memiliki semua properti dan *method* yang terdapat pada kelas tersebut. Kelas yang baru ini tidak hanya dapat menambahkan properti atau *method* baru pada kelasnya, tetapi juga dapat melakukan *override* properti atau *method* yang terdapat pada kelas *parent* nya.

Selain itu, pada OOP juga terdapat konsep yang bernama *loose coupling*, dimana perubahan pada satu bagian kode tidak mengganggu kode yang lain. Semua konsep ini saling berhubungan dan sangat sulit untuk dipahami. Namun semakin familiar anda dengan konsep OOP, semakin banyak keuntungan yang diperoleh terutama pada *reusable code* yang mudah untuk di kelola.

2.2.6 *Waterfall* SDLC

Waterfall merupakan SDLC yang paling banyak digunakan dalam pengembangan sistem saat ini. Fase pada *waterfall* berjalan secara linear, dimana untuk melanjutkan ke fase berikutnya harus terlebih dahulu menyelesaikan fase sebelumnya. Terdapat 6 fase pada *waterfall*, yaitu *requirement analysis*, *system design, implementation, testing, deployment,* dan *maintenance*. Untuk masuk ke fase *implementation*, harus menyelesaikan fase *system design* terlebih dahulu dan seterusnya. *Output* dari fase pertama akan menjadi *input* untuk fase kedua dan seterusnya.



Gambar 2.5 Waterfall Lifecycle

Sumber : Sommervile (2011)

Pada Gambar 2.5, terlihat fase yang terdapat pada *waterfall* yaitu (Sommerville, 2011) :

1. *Requirement definition*

   Fase ini merupakan fase yang sangat penting yang mempengaruhi apakah sistem yang dikembangkan akan sesuai dengan yang diharapkan atau tidak. Konsep dari sistem di evaluasi untuk memastikan bahwa sistem akan dapat terus berjalan dan sejalan dengan tujuan organisasi. Seluruh kemungkinan kebutuhan dari sistem yang akan dibangun dianalisa dan didokumentasikan. Selain itu harus dipastikan bahwa kebutuhan bisnis, keamanan, dan kebutuhan lainnya telah direncanakan pada fase ini.

2. *System and software design*

   Tujuan utama dari fase ini adalah mengevaluasi kebutuhan yang telah dirangkum pada fase pertama apakah kebutuhan tersebut sesuai dan dapat dikembangkan menjadi suatu sistem. Rancangan arsitektur sistem mulai dari *hardware* dan *software* akan diperiksa untuk memastikan apakah sistem yang dikembangkan dapat diterapkan kedalam organisasi atau tidak. Setiap fungsi diperiksa untuk memastikan agar fungsi tersebut dapat berfungsi sebagaimana mestinya dan dapat diuji nantinya.

3. *Implementation and unit testing*

   Berdasarkan rancangan sistem yang telah dibuat pada fase sebelumnya, maka sistem akan mulai dibangun dalam bentuk sistem kecil yang disebut unit. Unit-unit ini nantinya akan diintegrasikan pada fase berikutnya. Setiap unit diuji untuk mengetahui fungsionalitasnya.

4. *Integration and system* t*esting*

   Seluruh unit yang sudah dibangun akan diintegrasikan menjadi sebuah sistem setelah sebelumnya dilakukan pengujian untuk setiap unit. Kemudian seluruh unit yang sudah terintegrasi ini akan diuji kembali untuk mengetahui apakah ada kesalahan atau kegagalan sistem.

5. *Operation and maintenance*

   Terkadang muncul permasalahan setelah sistem diterapkan. Untuk mengatasi permasalahan tersebut, maka harus dilakukan perawatan dengan cara mengeluarkan edisi terbaru dari sistem. Perawatan ini juga dapat dilakukan ketika sistem membutuhkan penambahan fungsi atau fungsi baru.

2.2.7 Penjadwalan

Penjadwalan merupakan proses perancangan untuk menjalankan suatu proses pada waktu tertentu. Terdapat beberapa metode yang dapat digunakan dalam proses penjadwalan. Metode yang sering digunakan yaitu *first come first served, shortest job first* dan *round robin.* Khusus untuk penelitian ini hanya digunakan satu buah metode yaitu *shortest job first*.

1. *Shortest job first* (SJF)

SJF merupakan algoritma penjadwalan dengan prioritas. Algoritma ini mengacu pada waktu berjalannya proses hingga proses tersebut selesai. Pada algoritma ini waktu jalan proses diasumsikan telah diketahui sebelumnya. Proses-proses akan diurutkan berdasarkan waktu jalannya. Proses dengan waktu jalan terpendek akan dijalankan terlebih dahulu hingga selesai. Pada SJF, penetapan jadwal akan berjalan secara dinamis. Sebab, tidak dapat dipungkiri bahwa urutan akan selalu berubah seiring masuknya proses-proses baru (Siahaan, 2016).

Dalam penelitian ini, SJF digunakan untuk mengurutkan proses pengerjaan pesanan sesuai durasi pengerjaan masing-masing pesanan. Pengerjaan pesanan untuk setiap mesin akan diurutkan mulai dari pesanan dengan durasi pengerjaan terpendek hingga pesanan dengan durasi pengerjaan terpanjang. Pesanan dengan durasi pengerjaan terpendek akan dikerjakan terlebih dahulu pada hingga pesanan tersebut selesai pada mesin tersebut. Hal ini akan meminimumkan rata-rata waktu tunggu untuk pesanan.

2.2.8 *Full Time Equivalent* (FTE)

Pengukuran beban kerja digunakan untuk mengetahui efektifitas dan efisiensi dari suatu divisi dalam mengerjakan tugasnya. Salah satu metode perhitungan beban kerja yaitu *full time equivalent* (FTE). FTE adalah jumlah pekerja yang dibutuhkan untuk melakukan tugasnya dalam periode waktu tertentu (Dewi dan Satriya, 2012). Berdasarkan analisis beban kerja yang dikeluarkan oleh Badan Kepegawaian Negara, total nilai FTE dapat dibagi menjadi 3 kategori yaitu :

1. Nilai FTE diatas 1,28 dianggap *overload.*
2. Nilai FTE interval 1 – 1,28 dianggap normal.
3. Nilai FTE interval 0 – 0,99 dianggap *underload.*

Perhitungan rumus untuk mendapatkan nilai FTE dapat dilihat pada Persamaan 2.1.

$$\mathrm{FTE} = \frac{Total\ Working + Allowance}{Effective\ Working} \qquad (2.1)$$

Keterangan :

Total working = total waktu produktif pekerja

Allowance = persentase kelonggaran yang dibutuhkan pekerja

Effective working = total waktu jam operasional perusahaan

Nilai indeks FTE dapat diuraikan lagi berdasarkan jumlah tenaga kerja yang dibutuhkan (Dewi dan Satriya, 2012) yaitu sebagai berikut.

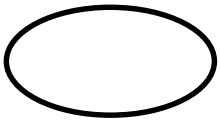
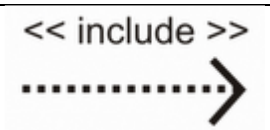
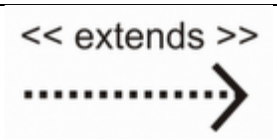
1. Jika nilai FTE > 1,28 maka harus ditambahkan 1 orang tenaga kerja.
2. Jika nilai FTE > 2,56 maka harus ditambahkan 2 orang tenaga kerja
3. Jika nilai FTE > 3,84 maka harus ditambahkan 3 orang tenaga kerja
4. Jika nilai FTE > 5,12 maka harus ditambahkan 4 orang tenaga kerja

2.2.9 UML

2.2.9.1 *Use Case* Diagram

Diagram *use case* menggambarkan fungsionalitas sistem secara umum. Apa saja yang dapat dilakukan terhadap sistem digambarkan pada diagram ini. Diagram *use case* terdiri dari beberapa elemen seperti yang terlihat pada Tabel 2.2. Diagram *use case* tidak menjelaskan secara detail tentang proses kerja fungsi.

Tabel 2.1 Simbol-simbol *use case diagram*

No	Nama	Gambar	Fungsi
1	*Use case*		Sebagai gambaran fungsi pada sistem, biasanya dinyatakan dengan menggunakan kata kerja.
2	*Actor*		Menggambarkan seseorang atau sesuatu yang berhubungan dengan sistem, biasanya dinyatakan dengan menggunakan kata benda.
3	*Association*		Menggambarkan hubungan antara aktor dan *use case* atau sebaliknya.
4	*Generalization*		Menggambarkan hubungan generalisasi antara dua buah *use case* atau *actor*, dimana salah satu *use case* atau *actor* merupakan induk dari *use case* atau *actor* lain.
5	*Include*		Menggambarkan kondisi ketika sebuah *use case* memerlukan *use case* lain sebagai syarat untuk menjalankan fungsinya.
6	*Extend*		Menggambarkan kondisi ketika menjalankan sebuah *use case*, maka *use case* lain dapat ikut dijalankan ataupun tidak.

Sumber: Sukamto & Shalahuddin (2011)

2.2.9.2 *Use Case Scenario*

Use case scenario dibuat berdasarkan diagram *use case*. *Use case scenario* menjelaskan setiap *use case* pada diagram *use case* secara detail. Aktor yang berhubungan dengan *use case*, kondisi awal ketika menjalankan *use* case, alur kerja *use case*, skenario alternatif dari *use case*, hingga kondisi akhir ketika *use case* selesai dijalankan dijabarkan pada *use case scenario. Form*at penulisan *use case scenario* dapat dilihat pada Tabel 2.2.

Tabel 2.2 *Form*at u*se case scenario*

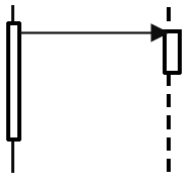
Flow of event for x use case	
Objective	Tujuan yang ingin dicapai dari *use case*.
Actors	Aktor yang berhubungan dengan *use case*.
Pre-Condition	Kondisi awal sebelum menjalankan *use case*.
Main Flow	Alur kerja *use case* dijelaskan disini.
Alternative Flow	Alur alternatif ketika *use case* dijalankan tidak sesuai alur yang ditetapkan.
Post Condition	Kondisi akhir ketika *use case* selesai dijalankan.

2.2.9.3 *Sequence Diagram*

Sequence diagram dibuat berdasarkan *use case scenario*, dimana satu *sequence diagram* menggambarkan satu *use case scenario*. *Sequence diagram* menggambarkan interaksi antar objek didalam sistem mengacu pada urutan waktu (Sukamto & Shalahuddin, 2011). Pertukaran pesan antara satu objek dengan objek lain dalam menjalankan fungsinya digambarkan pada diagram ini. Elemen penyusun *sequence diagram* dapat dilihat pada Tabel 2.3.

Tabel 2.3 Simbol-simbol *sequence diagram*

No	Nama	Gambar	Fungsi
1	*Actor*		Menggambarkan seseorang atau sesuatu atau yang berinteraksi dengan sistem.
2	*Boundary*		Menggambarkan antarmuka yang terdapat pada sistem.
3	*Controller*		Mengatur alur kerja sistem, sebagai penghubung antara *boundary* dan *entity*.
4	*Entity*		Menggambarkan hubungan dengan *database*.

No	Nama	Gambar	Fungsi
5	Lifeline		Menggambarkan lama aktifnya suatu objek dalam fungsi yang dijalankan.
6	*Message*		Menggambarkan pesan yang dikirim oleh satu objek ke objek lain.

Sumber : Sukamto & Shalahuddin (2011)

2.2.9.4 *Class Diagram*

Class diagram dibuat berdasarkan *sequence diagram* yang telah dibuat sebelumnya. *Class diagram* berisi seluruh objek yang digambarkan pada *sequence diagram* beserta *method* serta relasinya. Pada *class* diagram, setiap objek pada *sequence diagram* menjadi kelas dan pesan yang dikirimkan antar objek tersebut menjadi *method*. *Class diagram* ini nantinya akan dijadikan dasar dalam implementasi sistem.

Secara umum sebuah class diagram dibagi menjadi tiga area yaitu nama, atribut, dan *method*. Atribut merupakan variabel-variabel yang dimiliki oleh suatu *class*, dan *method* adalah fungsi yang dimiliki oleh *class* tersebut. Atribut dan *method* memiliki salah satu sifat dari beberapa sifat berikut:

1. *Private*, dimana suatu atribut atau *method* tidak dapat digunakan diluar *class* yang bersangkutan.
2. *Protected*, dimana suatu atribut atau *method* hanya dapat digunakan oleh *class* yang bersangkutan dan *class* yang mewarisinya.
3. *Public*, dimana suatu atribut atau *method* dapat digunakan oleh siapa saja.

Terdapat beberapa jenis hubungan antar *class* yaitu:

1. Asosiasi, yaitu hubungan *class* secara umum yang menggambarkan suatu *class* sebagai atribut *class* lain.
2. Agregasi, yaitu hubungan dimana satu *class* merupakan bagian dari *class* lain dan dapat berdiri sendiri.
3. Komposisi, yaitu hubungan dimana satu *class* merupakan bagian dari *class* lain dan tidak dapat berdiri sendiri.
4. Dependensi, yaitu hubungan yang menyatakan operasi pada suatu *class* menggunakan *class* lain.
5. Generalisasi, yaitu hubungan pewarisan dimana satu *class* merupakan turunan dari *class* lain dan mewarisi seluruh atribut serta *method* pada kelas induk.

2.2.10 Pengujian

2.2.10.1 *Whitebox Testing*

Whitebox testing merupakan teknik pengujian yang berfokus pada kode atau algoritma program. Tujuan dari *whitebox testing* ini untuk memastikan bahwa tidak ada kesalahan pada kode atau algoritma program sehingga nantinya sistem dapat berjalan dengan baik. Pengujian dilakukan menggunakan metode *basis path* dengan tujuan untuk mengetahui nilai *cyclomatic complexity. Cyclomatic complexity* menggambarkan seberapa kompleks program. *Nilai cylomatic complexity* didapat berdasarkan *flow graph* suatu algoritma, dimana *flow graph* ini nantinya akan dijadikan dasar dalam membuat jalur eksekusi program. Jalur ini akan dijadikan dasar dalam membangun kasus uji yang digunakan untuk menjalankan setiap pernyataan di dalam program dalam satu kali uji. Elemen yang terdapat pada *basis path* yaitu:

1. *Flow graph*

   Flow graph merupakan representasi dari algoritma program. *Flow graph* menggambarkan algoritma berdasarkan *node* dan *edge* nya. *Node* digambarkan dengan lingkaran. Setiap *node* dapat mewakili satu atau lebih prosedur. *Node* yang mewakili *node* lain disebut *predicate* node. *Edge* menggambarkan arah suatu algoritma pada program. *Edge* digambarkan dengan tanda panah. Wilayah yang dibatasi antara *edge* dan *node* disebut region. Wilayah diluar *flow graph* dihitung satu region.

2. *Cyclomatic complexity*

   Cyclomatic complexity adalah suatu nilai yang digunakan sebagai acuan dalam menentukan kompleksitas suatu algoritma. *Cyclomatic complexity* dihitung menggunakan rumus :

   a. V(G) = jumlah region
   b. V(G) = E (*edge*) + N (*node*) + 2
   c. V(G) = P (*predicate node*) + 1

   Penjelasan untuk setiap nilai *cyclomatic complexity* dapat dilihat pada Tabel 2.4.

Tabel 2.4 Deskripsi nilai *cylomatic complexity*

Cyclomatic complexity	**Arti**
1-10	Program mudah dipahami, mudah dilakukan implementasi, mudah dilakukan perbaikan, mudah dilakukan pengujian dan risiko kesalahan program rendah.
11-20	Program lebih kompleks, pengujian lebih sulit, dan resiko kesalahan program sedang.

Cyclomatic complexity	Arti
21-50	Program sangat kompleks, pengujian memerlukan usaha karena terdapat banyak jalur eksekusi program, dan resiko kesalahan program tinggi.
Lebih dari 50	Program sulit dilakukan implementasi, perbaikan, pengujian, dan risiko kesalahan program sangat tinggi.

Sumber : *Software Engineering Institute* (1997)

3. *Independent path*

   Independent path merupakan semua jalur yang dilewati untuk menghasilkan suatu hasil yang baru. Sebuah *independent path* sedikitnya melewati satu *edge* dan *edge* tersebut belum dilewati oleh *independent path* lain.

4. *Test case*

   Test case merupakan data yang digunakan untuk menguji alur logika pada masing-masing *independent path*.

2.2.10.2 Blackbox

Blackbox testing merupakan teknik pengujian sistem yang hanya melibatkan *input* dan *output* sistem. Pada *blackbox testing*, diujikan suatu masukan terhadap suatu fungsi untuk mengetahui *output* dari fungsi tersebut. Pada *blackbox* tidak ada pengujian terhadap kode program, hanya berfokus pada fungsi-fungsi yang terdapat pada sistem. Pengujian ini dilakukan untuk mengetahui apakah sistem yang dibangun sesuai dengan daftar kebutuhan atau tidak (Agarwal, et al., 2010).

Pada awal pengujian, penguji menyiapkan atau membuat set kondisi *input* yang nantinya akan diujikan ke seluruh fungsi yang terdapat pada sistem. Penguji juga menyiapkan prosedur pengujian serta hasil yang diharapkan. Kemudian, *output* hasil pengujian akan dicek apakah sesuai dengan hasil yang diharapkan atau tidak. Berdasarkan hasil pengujian, selanjutnya ditentukan apakah fungsi valid atau tidak.

2.2.11 *Javascript*

Javascript merupakan bahasa pemrograman yang digunakan untuk membuat halaman *web* yang interaktif dan dinamis (Mahanani, 2012). *Javascript* bersifat *client side* dimana proses pengolahannya dilakukan di sisi *client.* Berbeda dengan bahasa pemrograman seperti PHP atau ASP yang bersifat *server side* dimana hanya bekerja di sisi *server*. Ketika kita merequest sebuah halaman *web* yang memuat kode *javascript*, maka *javascript* tersebut akan cepat ditampilkan di halaman *browser* tanpa perlu diolah di *server* terlebih dahulu. *Javascript* dapat merespon aktifitas *user* secara instan berdasarkan event yang dilakukan *user*. *Javascript* akan digunakan untuk menampilkan tabel serta notifikasi pesan masuk secara dinamis.