BAB II

LANDASAN TEORI

2.1. Tinjaun Pustaka

A. Konsep Dasar Sistem

1. Pengertian Sistem

Menurut Fatta (2007:9), Sistem informasi manajemen didefinisikan sebagai suatu alat untuk menyajikan informasi dengan cara sedemikian rupa sehingga bermanfaat bagi penerimanya.

Menurut McLeod dalam Yakub (2012:1) mengemukakan bahwa "sistem adalah sekelompok elemen-elemen yang terintegrasi dengan tujuan yang sama untuk mencapai tujuan".

2. Pengertian Sistem Informasi

Menurut O'Brian dalam Yakub (2012:17) "sistem informasi (information system) merupakan kombinasi teratur dari orang-orang, perangkat keras (hardware), perangkat lunak (software), jaringan komunikasi dan sumber daya data yang mengumpulkan, mengubah dan menyebarkan informasi dalam sebuah organisasi".

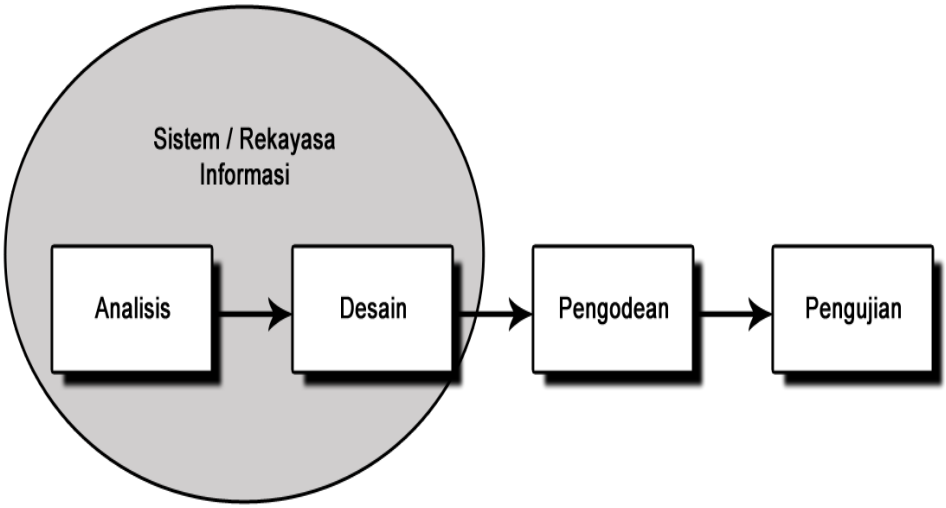
B. Pengertian Penggajian

> Upah atau gaji adalah hak pekerja atau buruh yang diterima dan dinyatakan dalam bentuk uang sebagai imbalan dari pengusaha atas pemberi kerja kepada pekerja atau buruh yang ditetapkan dan dibayarkan menurut perjanjian kerja. Kesepakatan atau peratran peundang-undangan, termasuk tunjangan bagi pekerja/buruh dan keluarganya atas suatu pekerjaan dan/jasa yang telah atau akan dilakukan (Adisu, 2008:2)

C. Siklus Hidup Pengembangan Sistem

Model pengembangan perangkat lunak dikenal dengan nama Siklus Hidup Pengembangan Sistem (SDLC). *Waterfall Model* menggambarkan sekuensial linier untuk rekayasa perangkat lunak, model ini muncul pertama kali tahun 1970 sehingga sering dianggap kuno, tetapi model ini yang paling banyak digunakan dalam rekayasa perangkat lunak.

Model SDLC air terjun *(waterfall)* sering juga disebut model sekuensial linier atau alur hidup klasik. Model air terjun menyediakan pendekatan alur hidup perangkat lunak secara sekuensial atau terurut dimulai dari analisis, desain, pengodean, pengujian, dan tahap pendukung. Menurut Rosa dan Shalahuddin (2013:28-30) model waterfall adalah sebagai berikut:



Gambar II.1
Ilustrasi Model Waterfall
Sumber: Rosa dan Shalahuddin (2013:29)

a. Analisis kebutuhan perangkat lunak

   Proses pengumpulan kebutuhan dilakukan secara intensif untuk menspesifikasikan kebutuhan perangkat lunak agar dapat dipahami perangkat lunak seperti apa yang dibutuhkan oleh *user*. Spesifikasi kebutuhan perangkat lunak pada tahap ini perlu untuk didokumentasikan.

b. Desain

   Desain perangkat lunak adalah proses multi langkah yang focus pada desain pembuatan program perangkat lunak termasuk struktur data, arsitektur perangkat lunak, representasi antarmuka, dan prosedur pengkodean. Tahap ini mentranslasi kebutuhan perangkat lunak dari tahap analisis kebutuhan ke representasi desain agar dapat diimplementasikan menjadi program pada tahap selanjutnya. Desain perangkat lunak yang dihasilkan pada tahap ini juga perlu didokumentasikan.

c. Pembuatan kode program

   Desain harus ditranslasikan ke dalam program perangkat lunak. Hasil dari tahap ini adalah program komputer sesuai dengan desain yang telah dibuat pada tahap desain.

d. Pengujian

   Pengujian fokus pada perangkat lunak dari segi *logik* dan fungsional dan memastikan bahwa semua bagian sudah diuji. Hal ini dilakukan untuk meminimalisir kesalahan (*error*) dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan.

e. Pendukung (*support*) atau pemeliharaan (*maintenance*)

Tidak menutup kemungkinan sebuah perangkat lunak mengalami perubahan ketika sudah dikirimkan ke *user*. Perubahan bisa terjadi karena adanya kesalahan yang muncul dan tidak terdeteksi saat pengujian atau perangkat lunak harus beradaptasi dengan lingkungan baru. Tahap pendukung atau pemeliharaan dapat mengulangi proses pengembangan mulai dari analisis spesifikasi untuk perubahan perangkat lunak yang sudah ada, tapi tidak untuk membuat perangkat lunak baru.

Model air terjun sangat cocok digunakan, kebutuhan pelanggan sudah sangat dipahami dan kemungkinan terjadinya perubahan kebutuhan selama pengembangan perangkat lunak kecil. Hal positif dari model air terjun adalah struktur tahap pengembangan sistem jelas, dokumentasi dihasilkan di setiap tahap pengembangan, dan sebuah tahap dijalankan setelah tahap sebelumnya selesai dijalankan (tidak ada tumpang tindih pelaksanaan tahap).

D. *UML* (*Unified Modeling Language*)

Dalam merancang program dibutuhkan peralatan pendukung yang dapat digunakan untuk membantu meringankan proses perancangan program aplikasi. *UML* yang berarti bahasa pemodelan standar merupakan alat komunikasi yang konsisten dalam mensuport para pengembang sistem yang memungkinkan para anggota *team* untuk bekerja sama dengan model yang sama dalam merancang suatu sistem yang akan dibuat.

UML dipalikasikan untuk maksud tertentu Widodo dan Herlawati (2011:10) antara lain:

a. Merancang perangkat lunak
b. Sarana komunikasi antara perangkat lunak dengan proses bisnis
c. Menjabarkan sistem secara rinci untuk dianalisa dan mencari apa yang diperlukan sistem.
d. Mendokumentasi sistem yang ada, proses-proses dan organisasinya.

Model yang digunakan untuk membangun dalam pengembngan dalam pemrograman terstruktur adalah:

1. *Use Case Diagram*

"*Use Case Diagram* bersifat setatik dan untuk mengorganisasi dan memodelkan perilaku suatu sistem yang dibutuhkan serta diharapkan pengguna" (Widodo dan Herlawati 2011:10).

Menurut Yasin (2012:198) mengemukakan bahwa "*use case diagram* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem". Yang ditekankan adalah "apa" yang diperbuat sistem dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* adalah sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dana merancang *test case* untuk semua *feature* yang ada pada sistem.

2. *Activity Diagram*

Bersifat dinamis, diagram aktifitas adalah tipe khusus dari diagram status yang memperlihatkan aliran dari suatu aktifitas ke aktifitas lainya dalam suatu sistem. Diagram ini penting terutama dalam permodelan fungsi fungsi suatu sistem dan member tekanan pada aliran kendali suatu objek.(Widodo dan Herlawati 2011:11).

Menurut Yasin (2012:201) mengemukakan bahwa "*activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir". *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state diagram* khusus dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di *trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

3. *Component Diagram*

Menurut Yasin (2012:203) mengemukakan bahwa *component diagram* menggambarkan struktur dan hubungan antar komponen piranti perangkat lunak, termasuk ketergantungan (*dependency*) diantaranya. Komponen piranti lunak adalah modul berisi kode, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*.

4. *Deployment Diagram*

Diagram deployment berhubungan erat dengan *component diagram* dimana *diagram* ini memuat satu atau lebih komponen-komponen.

Diagram ini sangat berguna saat aplikasi kita berlaku sebagai aplikasi yang dijalankan pada banyak mesin *(distributed computing)* (Widodo dan Herlawati 2011:12).

Menurut Yasin (2012:204) mengemukakan bahwa *deploynent diagram* menggambarkan detil bagaimana komponen di *deploy* dalam infrastruktur sistem dimana komponen akan terletak (pada mesin, server atau piranti perangkat keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal.

Sebuah *node* adalah server, *workstation*, atau piranti keras lain yang digunakn untuk men-*deploy* komponen dalam lingkungan sebenarmya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

E. Pengertian Pemrograman Terstruktur

Menurut Rosa dan Shalahuddin (2013:67-68) mengemukakan bahwa "pemrograman terstruktur adalah konsep atau paradigma atau sudut pandang pemrograman yang membagi-bagi program berdasarkan fungsi-fungsi atau prosedur-prosedur yang dibutuhkan program komputer". Modul-modul (pembagian program biasanya dibuat dengan mengelompokkan fungsi-fungsi dan prosedur-prosedur yang diperlukan sebuah proses tertentu. Adapun konsep pemrograman terstruktur yang akan dibahas yaitu :

a. Modular

Menurut Al-Bahra (2012:303) mengemukakan bahwa "*modular programming* (pemrograman modular) merupakan suatu teknik yang digunakan untuk menulis program yang berukuran besar". Program dibagi menjadi menjadi beberapa bagian kecil. Tiap bagian disebut modul, yang melakukan suatu tugas tertentu. Umumnya setiap modul program terdiri

dari sekumpulan pernyataan yang memiliki fungsi atau keguanaan tertentu di dalam program. Setiap modul tersebut diberi nama sehingga untuk menyatakannya cukup dengan menyebut namanya. Program yang didefinisikan dengan baik akan mudah dibaca dan dimengerti oleh pemakai serta menjadi lebih efisien karena modul yang sama mungkin dipakai pada beberapa tahapan program.

b. *Top Down*

Pendekatan top-down ini sangat berguna dalam perencanaan pemrograman modular. Dalam pemrograman top-down yang pertama kita definisikan adalah modul utama. Modul utama yang dimaksud adalah modul yang pertama kali dijalankan, yang memanggil modul lain dan juga modul yang mengakhiri program. Keuntungan dari teknik top-down adalah bahwa setiap tahap pemrograman yang ada menjadi sederhana karena setiap tingkat mengabaikan detail dari tingkat yang lebih rendah.

c. *Bottom Up*

Bila ada masalah yang kompleks, maka pemecahan masalah dilakukan dengan menggabungkan prosedur-prosedur yang ada menjadi satu kesatuan program guna menyelesaikan masalah tersebut. Pendekatan ini bertitik tolak pada tujuan program secara umum/menyeluruh dan bukan bagaimana cara mencapainya. Secara bertahap garis besar proses ini diuraikan menjadi tahapan-tahapan yang rinci sehingga pada akhirnya semua bagian program selesai.

F. Perangkat Lunak Pendukung

Dalam membangun *website* digunakan beberapa program aplikasi yaitu:

a. *PHP* menurut Madcom (2013:310) sebagai berikut:

1) *PHP* adalah singkatan dari: *Hypertext Prepocessor*.

2) *PHP* adalah bahasa *scripting server-side*, artinya dijalankan di *server*, kemudian *output*nya dikirim ke *client* (*browser*).

3) *PHP* digunakan untuk membuat aplikasi *web*.

4) *PHP* mendukung banyak *database* (*MySQL, Informix, Oracle, Sybase, Solid, PostgresSQL, Generic ODBC*, dll)

b. *MYSQL*

Menurut Supriyanto (2010:52) "SQL (*Structure Query Language)* merupakan perintah yang digunakan untuk mengelolah dan memanipulasi *database* berdasarkan *query* yang dibuat oleh *user*".

G. *Entity Relationship Diagram (ERD)*

Menurut Yasin (2012:276). "ERD adalah suatu pemodelan dari basis data relasional yang didasarkan atas persepsi didalam dunia nyata, dunia ini senantiasa terdiri dari sekumpulan objek yang saling berhubungan antara satu dengan yang lainnya". Suatu objek disebut *entity* dan hubungan yang dimilikinya disebut *relationship*. Suatu *entity* bersifat unik dan memiliki atribut sebagai pembeda dengan *entity* lainnya. ERD merupakan suatu model untuk menjelaskan hubungan antar data dalam basis data berdasarkan objek-objek dasar data yang mempunyai hubungan antar relasi. ERD untuk memodelkan struktur data dan hubungan antar data, untuk menggambarkannya digunakan beberapa notasi dan simbol. Pada dasarnya ada tiga simbol yang digunakan, yaitu :

1. Entitas (*Entity)*

Merupakan objek yang mewakili sesuatu yang nyata dan dapat dibedakan

dari sesuatu yang lain. Simbol dari *entity* ini biasanya digambarkan dengan persegi panjang. Entitas menunjukkan objek-objek dasar yang terkait didalam sistem. Objek dasar dapat berupa orang, benda atau hal lain yang keterangannya perlu disimpan dalam basis data.

2. Atribut (*Attribute*)

Setiap entitas pasti mempunyai elemen yang disebut atribut yang berfungsi untuk mendeskripsikan karakteristik dari entitas tersebut. Isi dari atribut mempunyai sesuatu yang dapat mengidentifikasikan isi elemen satu dengan yang lain. Macam-macam atribut sebagai berikut:

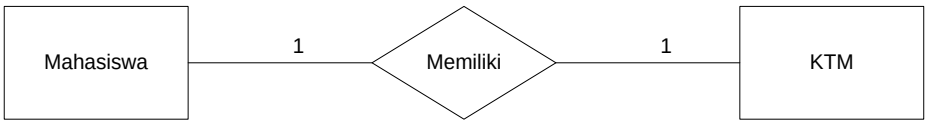
1) *Simple attribute* yaitu atribut yang terdiri atas satu komponen tunggal dengan keberadaannya yang independent dan tidak dapat dibagi lagi ke komponen yang lebih kecil. *Simple attribute* dikenal juga dengan nama *atomic attribute*.
2) *Composite attribute* yaitu atribut yang memiliki berbagai komponen dimana semua komponennya memiliki keberadaan yang independen.
3) *Single value attribute* yaitu sebuah atribut yang mempunyai nilai tunggal untuk setiap kejadian.
4) *Multi value attribute* yaitu sebuah atribut yang mempunyai beberapa nilai untuk setiap kejadian pada sebuah entitas.
5) *Derived attribute* yaitu atribut yang memiliki nilai yang dihasilkan dari satu atau beberapa atribut lainnya dan tidak harus dari satu entitas.

3. Derajat Relasi (*Relationship*)

Relasi didefinisikan sebagai hubungan yang terjadi antar *entity*. Representasi diagram relasi adalah sebuah garis lurus yang menghubungkan dua buah *entity*. Jenis-jenis atau hubungan yang biasa terjadi antar satu *entity* dengan *entity* lain dalam sebuah basis data meliputi :

1) *One to One* (Satu ke Satu)

   Hubungan relasi satu ke satu yaitu setiap entitas pada himpunan entitas A berhubungan paling banyak dengan satu entitas pada himpunan entitas B. Contoh hubungan antara *entity* Mahasiswa dengan Kartu Mahasiswa. Seorang mahasiswa hanya boleh memiliki satu kartu mahasiswa, satu kartu mahasiswa hanya dimiliki oleh satu orang mahasiswa.



Gambar II.2
One to One
Sumber : Yasin (2012:278)

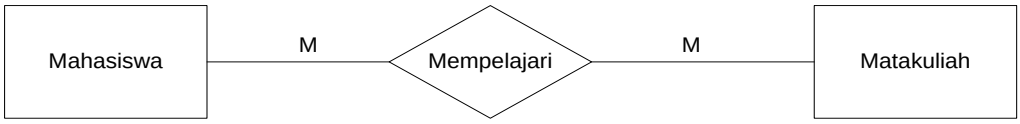
2) *One to Many* (Satu ke Banyak)

   Setiap entitas pada himpunan entitas A dapat berhubungan dengan banyak entitas pada himpunan B, tetapi setiap entitas pada entitas B dapat berhubungan dengan satu entitas pada himpunan entitas A. Contohnya hubungan yang terjadi antara entiti Dosen dengan Matakuliah. Seorang dosen dapat mengajar satu atau lebih matakuliah.

Gambar II.3
One to Many
Sumber : Yasin (2012:279)

3) *Many to Many* (Banyak ke Banyak)

Setiap entitas pada himpunan entitas A dapat berhubungan dengan banyak entitas pada himpunan entitas B. Contohnya hubungan yang terjadi antara Mahasiswa dengan Mata Kuliah. Satu mahasiswa dapat mengikuti lebih dari satu mata kuliah dan satu mata kuliah dapat diikuti oleh lebih dari satu mahasiswa.



Gambar II.4
Many to Many
Sumber : Yasin (2012:279)

b. Keterkaitan (*Cardinality Ratio*)

Menjelaskan batasan jumlah keterhubungan suatu entitas dengan entitas lainnya. Terdapat tiga jenis *cardinality ratio* yaitu :

1) Satu ke satu
2) Satu ke banyak
3) Banyak ke banyak

c. *Participation Constraint*

Menjelaskan apakah keberadaan suatu entitas tergantung pada hubungannya dengan entitas lain. Terdapat dua macam *Participation*

Constrain :

1) *Total Participation*

   Menjelaskan keberadaan suatu entitas tergantung pada hubungannya dengan entitas lain.

2) *Partial Participation*

   Menjelaskan keberadaan suatu entitas tidak tergantung pada hubungannya dengan entitas lain.

H. *LRS* (*Logical Record Structure*)

Menurut Frieyadie (2007:13) “LRS merupakan hasil dari pemodelan *Entity Relationship (ER)* beserta atributnya sehingga bisa terlihat hubngan-hubungan antar entitas”.

Dalam pembuatan LRS terdapat tiga hal yang dapat mempengaruhi, menurut Frieyadie (2007:14) yaitu:

1. Jika tingkat hubungan *(cardinality)* satu pada satu *(one-to-one)*, maka digabungkan dengan entitas yang lebih kuat *(strong entity)*, atau digabungkan dengan entitas yang memiliki atribut yang lebih sedikit.
2. Jika tingkat hubungan *(cardinality)* satu pada banyak *(one-to-many)*, maka hubungan relasi atau digabungkan dengan entitas yang tingkat hubungannya banyak.
3. Jika tingkat hubungan *(cardinality)* banyak pada banyak *(many-to-many)*, maka hubungan relasi tidak akan digabungkan dengan entitas manapun, melainkan menjadi *LRS*.

2.2. Penelitian Terkait

Sarifudin dkk (2013:1) Sekolah Dasar Negeri (SDN) Pacitan dalam mengolah data gaji karyawan masih menggunakan cara yang konvensional, yakni mencatat pada pembukuan sehingga sering terjadi kesalahan dalam menghitung jumlah gaji guru yang didasarkan pada berbagai rincian yang berbeda-beda, hal ini dapat menghambat proses kinerja bendahara instansi tersebut. Kesalahan dalam menghitung gaji dapat berakibat fatal karena dapat berpengaruh dalam pembuatan laporan keuangan. Selain itu bendahara juga kesulitan dalam pembuatan laporan keuangan, karena petugas harus melihat data pada pembukuan, kemudian mengolahnya menggunakan Ms. Excel.

Purnamasari (2013:83) "Pembangunan Sistem Informasi Pengolahan Data Pegawai dan Penggajian Pada Unit Pelaksana Teknis Taman Kanak-Kanak dan Sekolah Dasar Kecamatan Pringkulu". Pada saat ini, prosedur yang diterapkan pada pengolahan data pegawai dan penggajian pada Unit Pelaksana Teknis Taman Kanak-Kanak dan Sekolah Dasar Kecamatan Pringkuku masih menggunakan metode konvensional. Metode ini membutuhkan waktu yang lama dalam pengolahan data pegawai dan penggajian. Dengan adanya hal tersebut, Unit Pelaksana Teknis Taman Kanak-Kanak dan Sekolah Dasar Kecamatan Pringkuku dirasa perlu untuk merubah metode pengelolaan data pegawai yang mereka gunakan saat ini, yaitu metode konvensional menjadi metode terkomputerisasi dan otomatis. Adapun metode penelitian yang digunakan untuk menyelesaikan berbagai permasalahan yang terjadi adalah pustaka, observasi, wawancara, analisis data dan system, perancangan sistem, pembuatan program, pengujian program dan implementasi program.