# **BAB III**

# **LANDASAN TEORI**

1. **Pengertian Aplikasi**

Dalam Kamus Besar Bahasa Indonesia (KBBI) aplikasi merupakan penerapan rancang system untuk mengolah data yang menggunakan aturan atau ketentuan Bahasa pemprograman tertentu yang bertujuan dibuat untuk mengerjakan dan melaksanakan tugas khusus dari pengguna.

Sedangkan menutut Jogiyanto aplikasi merupakan penggunaan dalam suatu komputer (*computer*), intruksi (*instruction*) atau pernyataan (*statement*) yang disusun sedemian rupa sehingga komputer dapat memproses *input* menjadi *output*. Dan menurut Ibisa merupakan alat bantu untuk mempermudah dan mempercepat proses pekerjaan dan bukan merupakan beban bagi penggunanya.

Menurut Simarmata(2010), Aplikasi Web adalah sebuah sistem informasi yang mendukung interaksi pengguna melalui antarmuka berbasis Web. Fitur-fitur aplikasi Web biasanya berupa data persistence, mendukung transaksi dan komposisi halaman Web dinamis yang dapat dipertimbangkan sebagai hibridisasi, antara hipermedia dan sistem informasi.

1. ***Inventory* atau Persediaan**

Dalam perusahaan dagang maupun manufaktur, persediaan sangat dan termasuk bagian aktivasi lancer yang aktif.

Suatu aktifa yang meliputi barang-barang milik perusahaan dengan maksud untuk dijual dalam periode usaha tertentu atau persediaan barang – barang yang masih dalam pengerjaan atau proses produksi ataupun persediaan bahan baku yang menunggu penggunaannya dalam proses produksi (Benny Alexandri 2009)

Sedangkan menurut rstotno (2009) persediaan merupakan teknik untuk manajemen material yang berkaitan dengan perusahaan.

Secara umum pengertian persediaan adalah suatu sistem yang mengatur semua jumlah barang dagang yang selalu mengalami perubahan jumlah dan nilai melalui proses penjualan atau transaksi.

1. ***United Modeling Language* (*UML*)**

Menurut Adi Nugroho (2010), UML (*Unified Modeling Language*) adalah bahasa pemodelan untuk sistem atau perangkat lunak yang berparadigma berorientasi objek. Pemodelan (*modeling*) sesungguhnya digunakan untuk menyederhanakan permasalahan-permasalahan yang kompleks sedemikian rupa sehingga lebih mudah dipahami dan dipelajari. *UML* juga menyediakan standar notasi ataupun diagram yang digunakan untuk pemodelan sistem,

1. ***Use* *Case* *Diagram***

Menurut Adi Nugroho (2010), *Use Case Diagram* digunakan untuk memodelkan fungsionalitas - fungsionalitas sistem/perangkat lunak dilihat dari pengguna yang ada di luar sistem (yang sering dinamakan sebagai aktor). *Use Case* pada dasarnya merupakan unit fungsionalitas koheren yang diekspresikan sebagai transaksi-transaksi yang terjadi antara aktor dan sistem. Kegunaan dari *use case diagram* adalah untuk mendaftarkan aktor-aktor dan *use case – use case* dan memperlihatkan aktor-aktor mana yang berpartisipasi dalam masing-masing *use case*. Berikut notasi-notasi yang digunakan dalam *use case diagram*:

Tabel 3.1 Notasi-Notasi *Use Case Diagram*

|  |  |  |  |
| --- | --- | --- | --- |
| **NOTASI** | **KEGUNAAN** | | **SIMBOL** |
| *Actor* | Menggambarkan semua objek diluar sistem (bukan hanya pengguna sistem/ perangkat lunak) yang berinteraksi dengan sistem yang dikembangkan. | |  |
| *Use Case* | Menggambarkan fungsionalitas yang dimiliki sistem. | |  |
| *Collaboration* | Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi). | |  |
| *Assosiation* | Lintasan komunikasi antara *actor* dengan *use case.* | |  |
| *Include* | Penambahan perilaku ke suatu *use case* dasar yang secara eksplisit mendeskripsikan penambahan tersebut. | | <<include>> |
| *Extend* | Penambahan perilaku ke suatu *use case* dasar. | | <<extend>> |
| *Generalization* | Relasi antara pengklasifikasi yang memiliki deskripsi yang bersifat labih umum dengan berbagai pengklasifikasi yang lebih spesifik, digunakan dalam struktur pewarisan. | |  |
| *Dependency* | | Relasi antar dua elemen model. |  |

1. ***Class* *Diagram***

Menurut Adi Nugroho (2010), dalam notasi *UML*, himpunan kelas-kelas beserta hubungan / relasi / asosiasi antar kelas biasanya digambarkan menggunakan sebuah *diagram UML* yang dinamakan diagram kelas (*class diagram*). Jika kita perhatikan lebih jauh, sesungguhnya diagram kelas memiliki dua kegunaan / fungsi yang sangat penting, yaitu:

1. Mempresentasikan keadaan statis kelas-kelas yang terlibat dalam sistem. Kelas-kelas ini bisa saja merupakan kelas-kelas dalam bahasa pemrograman dan kelas-kelas persisten yang hadir dalam bentuk tabel-tabel yang ada di sistem basis data relasional.
2. Hubungan antar kelas dalam sistem atau perangkat lunak yang sedang kita kembangkan dapat terlihat dengan mudah.

Berikut notasi-notasi yang digunakan dalam *class diagram UML*:

Tabel 3.2 Notasi-Notasi *Class Diagram*

|  |  |  |
| --- | --- | --- |
| **NOTASI** | **KETERANGAN** | **SIMBOL** |
| *Class* | *Class* adalah balok-balok pembangun pada pemrograman berorientasi objek. Sebuah *class* digambarkan sebagai sebuah kotak yang terbagi menjadi 3 bagian. Bagian atas adalah bagian nama dari *class*. Bagian tengah mendefinisikan atribut *class*. Bagian bawah mendefinisikan *method* dari sebuah *class*. |  |

Tabel 3.2 Notasi-Notasi *Class Diagram* (Lanjutan)

|  |  |  |
| --- | --- | --- |
| **NOTASI** | **KETERANGAN** | **SIMBOL** |
| *Assosiation* | Sebuah asosiasi merupakan sebuah *relationship* paling umum antara 2 *class*, dan dilambangkan oleh sebuah garis yang menghubungkan antar 2 *class*. Garis ini dapat melambangkan tipe - tipe *relationship* dan juga dapat menampilkan hukum-hukum multiplisitas pada sebuah *relationship*. | 1..n  owned  by 1 |
| *Composition* | Jika sebuah *class* tidak bisa berdiri sendiri dan harus menjadi bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *composition* terhadap *class* tempatnya bergantung tersebut. |  |
| *Dependency* | Kadang kala class menggunakan *class* yang lain. Hal ini disebut *dependecy*. Umumnya *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain. |  |
| *Aggregation* | *Aggregation* mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut relasi “bagian dari”. Sebuah *aggregation* digambarkan sebagai sebuah garis dengan sebuah jajaran genjang yang tidak berisi / tidak solid. |  |
| *Generalization* | Sebuah *generalization* dilambangkan dengan sebuah panah dengan kepala panah yang tidak solid yang mengarah ke arah “*parent*”-nya / induknya. |  |

1. *Activity* *Diagram*

Menurut Adi Nugroho (2010), diagram aktivitas (*activity diagram*) sesungguhnya merupakan bentuk khusus dari state machine yang bertujuan untuk memodelkan komputasi-komputasi dan aliran - aliran kerja yang terjadi dalam sistem / perangkat lunak yang sedang dikembangkan. State pada diagram aktivitas merepresentasikan state dari komputasi yang dieksekusi, bukan state dari suatu objek biasa.

Biasanya, suatu *diagram* aktivitas mengasumsikan komputasi - komputasi dilaksanakan tanpa adanya interupsi-interupsi eksternal berbasis *event* terjadi padanya.

Suatu *diagram* aktivitas memuat di dalamnya *activity state* dimana suatu *activity state* merepresentasikan eksekusi pernyataan dalam suatu prosedur atau kinerja suatu aktivitas dalam suatu aliran kerja. Alih-alih menunggu selesainya atau event seperti yang terjadi pada state tunggu, *activity state* menunggu selesainya komputasi. Saat suatu aktivitas selesai maka akan berlanjut ke *activity state* berikutnya yang terlihat pada diagram aktivitas. Penyelesaian transisi dalam suatu diagram aktivitas biasanya akan terpicu saat aktivitas sebelumnya selesai.

*Activity state* biasanya tidak memiliki transisi-transisi yang diakibatkan oleh *event-event* eksplisit, tapi mungkin dibatalkan oleh transisi-transisi pada state yang melingkupinya. Berikut adalah notasi - notasi / simbol-simbol yang digunakan pada *activity diagram* :

Tabel 3.3 Notasi-notasi *Activity Diagram*

|  |  |  |
| --- | --- | --- |
| **NOTASI** | **KETERANGAN** | **SIMBOL** |
| *Initial* | Titik awal untuk memulai suatu aktivitas. |  |
| *Final* | Titik akhir untuk mengakhiri aktivitas. |  |
| *Activity* | Menandakan sebuah aktivitas. |  |
| *Decision* | Pilihan untuk menggambil keputusan. |  |
| *Fork* | Menunjukkan kegiatan yang dilakukan secara paralel. |  |
| *Join* | Untuk menggabungkan beberapa kegiatan secara paralel menjadi satu. |  |

1. *Sequence* *Diagram*

Menurut Adi Nugroho (2010), *Sequence* *Diagram* memperlihatkan interaksi sebagai *diagram* dua matra (dimensi). Matra vertikal adalah sumbu waktu, waktu bertambah dari atas ke bawah. Matra horizontal memperlihatkan peran pengklasifikasian yang merepresentasikan objek - objek mandiri yang terlibat dalam kolaborasi. Masing - masing pengklasifikasian direpresentasikan sebagai kolom-kolom vertikal dalam *sequence diagram* yang sering disebut sebagai garis waktu (*life line*). Selama objek ada, peran digambarkan menggunakan garis tegas. Selama aktivitas prosedur pada objek aktif, garis waktu digambarkan sebagai garis ganda. Pesan-pesan digambarkan sebagai suatu tanda panah dari garis waktu suatu objek ke garis waktu objek lainnya.

Panah-panah menggambarkan aliran pesan antar peran pengklasifikasian digambarkan dalam urutan waktu kejadiannya dari atas ke bawah. Berikut selengkapnya notasi-notasi yang digunakan dalam *sequence* *diagram*:

Tabel 3.4 Notasi-notasi *Sequence Diagram*

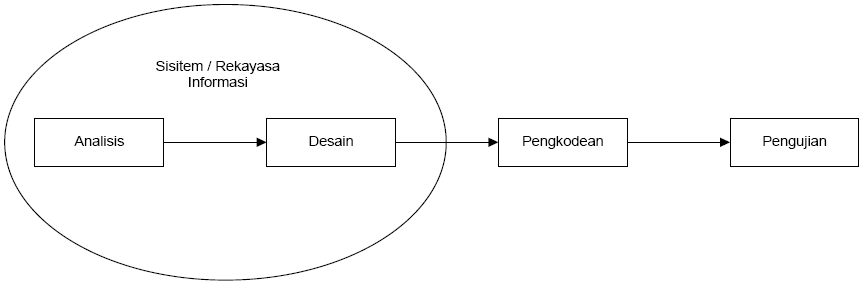
|  |  |  |
| --- | --- | --- |
| **NOTASI** | **KETERANGAN** | **SIMBOL** |
| *Object*  (Partisipan) | Objek atau biasa disebut partisipan merupakan *intance* dari sebuah *class* dan dituliskan tersusun secara horizontal. Digambarkan sebagai sebuah *class* (kotak) dengan nama objek di dalamnya yang diawali dengan titik koma. |  |
| *Actor* | *Actor* juga dapat bekomunikasi dengan objek, maka *actor* juga dapat diurutkan sebagai kolom. |  |
| *Life Line* | *Life line* mengindikasikan keberadaan sebuah *object* dalam baris waktu. Notasi untuk *life line* adalah garis putus-putus vertikal yang ditarik dari sebuah *object*. |  |

Tabel 3.4 Notasi-notasi *Sequence Diagram* (Lanjutan)

|  |  |  |
| --- | --- | --- |
| **NOTASI** | **KETERANGAN** | **SIMBOL** |
| *Activation* | Activation dinotasikan sebagai sebuah kotak persegi empat yang digambarkan pada sebuah *life line*. *Activation* mengindikasikan sebuah *object* yang akan melakukan sebuah aktivasi. |  |
| *Boundary* | *Boundary* terletak antara sebuah sistem dengan sekelilingnya. Semua *form*, laporan-laporan, antar muka diperangkat keras seperti *printer* dan *scanner* dan antar muka ke sistem lainnya adalah termasuk ke dalam kategori. |  |
| *Control* | *Control* berhubungan dengan fungsionalitas seperti pemanfaatan sumber daya, pemrosesan terdistribusi, atau penanganan kesalahan. |  |
| *Entity* | *Entity* digunakan untuk menangani informasi yang mungkin akan disimpan secara permanen. *Entity* bisa juga merupakan sebuah tabel pada struktur basis data. |  |
| *Message* | *Message*, digambarkan dengan anak panah horizontal antar *activation*. *Message* mengindikasikan komunikasi antara objek-objek. |  |
| *Self Message* | *Self message* mengindikasikan komunikasi kembali ke dalam sebuah objek itu sendiri. |  |

1. ***Waterfall***

Menurut Rosa dab Shalahudin, (2013) Model Air Terjun (*Waterfall*) disebut juga model Sekuensial Linier (*Sequential Linear*) atau alur hidup klasik (*Clasic Life Cycle*). Model air terjun menyediakan pendekatan alur hidup perangkat lunak secara sekuensial atau terurut dimulai dari analisis, desain, pengkodean, pengujian dan tahap pendukung.



Gambar 1.1 Metode *Waterfall*

(Rosa A. S dan M. Shalahuddin, 2013)

1. Analisa Sistem

Analisa sistem bertujuan untuk menentukan ruang lingkup, unit organisasi kegiatan dan skala sumber daya yang diperlukan untuk mensepesifikasi kebutuhan perangkat lunak agar dapat dipahami perangkat lunak seperti apa yang dibutuhkan oleh pengguna. Spesifikasi kebutuhan perangkat lunak pada tahap ini perlu untuk didokumentasikan.

1. Desain Sistem

Pada tahap desain sistem ini adalah proses multi langkah yang focus pada desain pembuatan program untuk menentukan proses, 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.

1. Pengkodean program (*Coding*)

*Coding* merupakan unsur yang sangat penting dalam pembuatan aplikasi. Desain harus diterjemahkan kedalam program perangkat lunak untuk menjadikan fungsi-fungsi atau fitur yang sesuai dengan desain yang telah dibuat pada tahap desain.

1. Pengujian (*Testing system*)

Tahap pengujian sistem focus pada perangkat lunak secara segi *logic* 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.

1. Pemeliharaan (*Maintanance*)

*Maintenance*, pada tahap ini perangkat lunak sudah terbentuk dengan baik, namun tidak menutup kemungkinan untuk perbaikan, perubahan ketika sudah dikirim ke pengguna, 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 tahap analisis spesifikasi untuk perubahan perangkat lunak baru.