**BAB II**

**LANDASAN TEORI**

1. **Pengertian Penerapan**

Menurut Kamus Besar Bahasa Indonesia (KBBI), pengertian penerapan adalah perbuatan menerapkan, sedangkan menurut beberapa ahli, penerapan adalah suatu perbuatan mempraktekkan suatu teori, metode, dan hal lain untuk mencapai tujuan tertentu dan untuk suatu kepentingan yang diinginkan oleh suatu kelompok atau golongan yang telah terencana dan tersusun sebelumnya. Penerapan adalah proses untuk memastikan terlaksananya suatu kebijakan dan tercapainya kebijakan tersebut. Penerapan juga dimaksudkan menyediakan sarana untuk membuat sesuatu dan memberikan hasil yang bersifat praktis terhadap sesama. Menurut Syaukani dkk, Penerapan adalah pelaksanaan serangkaian kegiatan dalam rangka untuk memberikan kebijakan publik sehingga kebijakan dapat membawa hasil, seperti yang diharapkan. ([https://www.dosenpendidikan.co.id/](https://www.dosenpendidikan.co.id/implementasi-adalah/) diakses pada tanggal 17 Januari 2020 pada pukul 00:53 WIB).

1. **Pengertian Analisa**

Menurut Gorys Keraf, analisa adalah sebuah proses untuk memecahkan masalah ke dalam bagian-bagian yang saling berkaitan satu sama lainnya. Sedangkan menurut Komarudin mengatakan bahwa analisis merupakan suatu kegiatan berfikir untuk menguraikan suatu keseluruhan menjadi komponen sehingga dapat mengenal tanda-tanda dari setiap komponen, hubungan satu sama lain dan fungsi masing-masing dalam suatu keseluruhan terpadu. Pengertian menurut kamus akuntansi yaitu bahwa analisa merupakan sebuah kegiatan evaluasi terhadap kondisi dari ayat-ayat yang berkaitan akuntansi dan alasan tentang perbedaan yang bisa muncul. Terakhir menurut Robert J. Schreiter (1991) mengatakan analisa merupakan membaca teks, dengan menempatkan tanda-tanda dalam interaksi yang dinamis dan pesan yang disampaikan.

Dari beberapa pengertian analisa diatas dapat ditarik kesimpulan bahwa analisa merupakan sekumpulan kegiatan, aktivitas dan proses yang saling berkaitan untuk memecahkan masalah atau memecakan komponen menjadi lebih detail dan digabungkan kembali lalu ditarik kesimpulan. Bentuk dari kegiatan analisa salah satunya yaitu merangkum data mentah menjadi sebuah informasi yang bisa disampaikan ke khalayak. Segala macam bentuk analisis menggambarkan pola-pola yang konsisten di dalam data, sehingga hasil analisa dapat dipelajari dan diterjemahkan dengan singkat dan penuh makna.

Belajar dari para ahli yang telah mendefinisikan pengertian analisa maka sebuah analisa data, proses dan hasil dari analisia biasanya dilakukan meliputi kegiatan mengorganisasikan data, mengelompokan data, mengklasifikasi data, memaparkan data, dan menarik kesimpulan dari keseluruhan data tersebut. Mengorganisasikan data yaitu berarti mengatur data-data yang telah diperoleh peneliti selama kegiatan penelitian berlangsung, sedangkan pengelompokan data yaitu mengelompokan data mana yang hendak dipakai dan data mana yang tidak dipakai. Mengklasifikasikan data juga mengelas-ngelaskan data sesuai kebutuhan. Memaparkan data yaitu menyampaikan hasil proses analisis data dan menarik kesimpulan atas informasi dari data yang telah disampaikan. (<https://pengertiandefinisi.com/pengertian-analisa-menurut-ahli/> diakses pada tanggal 28 Januari 2020 pada pukul 06:45 WIB).

1. **Data, Informasi, dan Pengetahuan (*Knowledge*)**

Perbedaan antara data dan informasi sering menjadi titik awal untuk memahami sistem informasi. Sebagai tambahan, pengetahuan juga diikutkan untuk dibahas, mengingat pada saat ini sistem pakar, sistem penunjang keputusan maupun *data mining* yang berbasiskan pengetahuan telah banyak digunakan.

**2.3.1 Data**

Data adalah segala fakta, angka, atau teks yang dapat diproses oleh computer. Saat ini, akumulasi pertumbuhan jumlah data berjalan dengan cepat dalam format dan basis data yang berbeda. Data-data tersebut, antara lain:

1. Data operasional atau transaksional, sperti penjualan, inventaris, penggajian, akuntansi, dan sebagainya.
2. Data nonoperasional, seperti industri penjualan (supermarket), peramalan, dan data ekonomi makro.
3. Metadata adalah data mengenai data itu sendiri, seperti desain logika basis data atau definisi kamus data.

Secara konseptual data adalah deskripsi tentang benda, kejadian, aktivitas, dan transaksi yang tidak mempunyai makna atau tidak berpengaruh secara langsung kepada pemakai. Misalnya, anda menjumpai deretan angka seperti berikut: 6.30 27 6.32 28 6.34 27. Anda mungkin merasakan bahwa deretan bilangan tersebut tidak memberikan makna apapun, itulah contoh data.

**2.3.2 Informasi**

Informasi merupakan pola, asosiasi, atau hubungan antara semua data yang dapat memberikan informasi. Sebagai contoh, analisis titik eceran (*retail point*) data transaksi penjualan dapat menghasilkan informasi mengenai produk apa yang sebaiknya dijual dan kapan menjualnya. Informasi dapat dikonversi menjadi pengetahuan mengenai pola-pola historis dan tren masa depan. Misalnya, ringkasan tentang penjualan eceran supermarket dapat dianalisis sehubungan dengan upaya promosi untuk memberikan pengetahuan mengenai perilaku konsumen dalam membeli. Dengan demikian, produsen atau pengecer dapat menemukan item yang paling rentan terhadap upaya promosi.

McFadden, dkk (1999) mendefinisikan informasi sebagai data yang telah diproses sedemikian rupa sehingga meningkatkan pengetahuan seseorang yang menggunakan data tersebut. Shannon dan Weaver, dua orang insinyur listrik melakukan pendekatan secara matematis untuk mendefinisikan informasi (Kroenke, 1992). Menurut mereka, informasi adalah jumlah ketidak pastian yang dikurangi ketika sebuah pesan diterima. Artinya, dengan adanya informasi, tingkat kepastian menjad meningkat. Menurut Davis (1999), informasi adalah data yang telah diolah menjadi sebuah bentuk yang berarti bagi penerimanya dan bermanfaat dalam pengambilan keputusan saat ini atau saat mendatang.



**Gambar 2.1** Transformasi Data Menjadi Informasi (Kadir, 2003)

**2.3.3 Pengetahuan (*Knowledge*)**

Pengetahuan (*knowledge*) adalah kombinasi dari naluri, gagasan, aturan, dan prosedur yang mengarahkan tindakan atau keputusan (Alter, 1992). Sebagai gambaran, informasi yang dipadukan dengan pengalaman masa lalu dan keahlian akan memberikan suatu pengetahuan yang tentu saja memiliki nilai yang tinggi. Sebuah gambaran tentang hubungan antara data, informasi, dan pengetahuan ditunjukan pada gambar berikut:



**Gambar 2.2** Hubungan Data, Informasi dan Pengetahuan (Kadir, 2003)

1. ***Data* *Mining***

Cara pandang dan pengetahuan yang berbeda membuat para ahli memberikan definisi berbeda tentang *data mining*. Sebagian ahli menyatakan bahwa *data mining* adalah langkah analisis terhadap proses penemuan pengetahuan di dalam basis data atau *knowledge discovery in databases* yang disingkat KDD (Fayaad et al. 1996). Pengetahuan bisa berupa pola data atau relasi antar data yang valid (yang tidak diketahui sebelumnnya). *Data mining* merupakan gabungan sejumlah disiplin ilmu computer (ACM 2006), (Clifton 2010), yang didefinisikan sebagai proses penemuan pola-pola baru dari kumpulan data yang sangat besar, meliputi metode-metode yang merupakan irisan dari *artificial intelligence, machine learning, statistics,* dan *database systems* (ACM 2006).

*Data mining* ditujukan untuk mengekstrak (mengambil intisari) pengetahuan dari sekumpulan data sehingga didapatkan struktur yang dapat dimengerti manusia serta meliputi basisdata dan managemen data, prapemrosesan data, pertimbangan model dan inferensi, ukuran ketertarikan, pertimbangan kompleksitas, pascapemrosesan terhadap struktur yang ditemukan, visualisasi, dan *online updating* (ACM 2006).

Tan (2006) mendefinisikan *data mining* sebagai proses untuk mendapatkan informasi yang berguna dari gudang basis data yang besar. *Data mining* juga dapat diartikan sebagai pengekstrakan informasi baru yang diambil dari bongkahan data besar yang membantu dalam pengambilan keputusan. Menurut Gartner Group *data mining* adalah suatu proses menemukan hubungan yang berarti, pola, dan kecenderungan dengan memeriksa dalam sekumpulan besar data yang tersimpan dalam penyimpanan dengan menggunakan teknik pengenalan pola seperti teknik statistik dan matematika (Larose, 2005).



**Gambar 2.3** Bidang Ilmu *Data Mining* (Kusrini, 2009)

*Data mining* bukanlah suatu bidang yang sama sekali baru, salah satu kesulitan untuk mendefinisikan *data mining* adalah kenyataan bahwa *data mining* mewarisi banyak aspek dan teknik dari bidang-bidang ilmu yang sudah mapan terlebih dulu. Gambar 2.3 menunjukan bahwa *data mining* memiliki akar yang panjang dari bidang ilmu seperti kecerdasan buatan (*artificial intellegent*), *machine learning,* statistik, database, dan juga *information retrieval* (Pramudiono 2006). Bidang lain yang juga mempengaruhi *data mining* adalah teknologi basis data, yang mendukung penyediaan penyimpanan yang efesien, pengindeksan, dan pemrosesan *query*. Teknik komputasi paralel sering digunakan untuk memberikan kinerja yang tinggi untuk ukuran set data yang besar, sedangkan komputasi terdistribusi dapat digunakan untuk menangani masalah ketika data tidak dapat disimpan di satu tempat.

* 1. **Kegunaan *Data Mining***

Kegunaan *data mining* dapat dibagi menjadi dua: deskriptif dan prediktif. Deskriptif berarti *data mining* digunakan untuk mencari pola-pola yang dapat dipahami manusia yang menjelaskan karakteristik data. Sedangkan prediktif berarti *data mining* digunakan untuk membentuk sebuah model pengetahuan yang akan digunakan untuk melakukan prediksi. Berdasarkan fungsionalitasnya, tugas-tugas *data mining* bisa dikelompokan ke dalam enam kelompok berikut ini (Fayyad et al. 1996):

1. Klasifikasi (*classification*): men-generalisasi struktur yang diketahui untuk diaplikasikan pada data-data baru. Misalkan, klasifikasi penyakit ke dalam sejumlah jenis, klasifikasi email ke dalam spam atau bukan.
2. Klasterisasi (*Clustering*): mengelompokan data, yang tidak diketahui label kelasnya, ke dalam sejumlah kelompok tertentu sesuai dengan ukuran kemiripannya.
3. Regresi (*regression*): menemukan suatu fungsi yang memodelkan data dengan galat (kesalahan prediksi) seminimal mungkin.
4. Deteksi anomali (*anomaly detection*): mengidentifikasi data yang tidak umum, bisa berupa *outlier* (pencilan), perubahan atau deviasi yang mungkin sangat penting dan perlu investigasi lebih lanjut.
5. Pembelajaran aturan asosiasi (*association rule learning*) atau pemodelan ketergantungan (*dependency modeling*): mencari relasi antar variabel.
6. Perangkuman (*summarization*): menyediakan representasi data yang lebih sederhana, meliputi visualisasi dan pembuatan laporan.
   1. **Tahapan KDD (*Knowledge Discovery in Database*)**
   2. ***Market Basket Analysis* atau Metode *Association Rule***

*Association rule* adalah teknik *data mining* untuk menemukan aturan asosiatif antara suatu kombinasi item. Contoh aturan asosiatif dari analisis pembelian di suatu pasar swalayan adalah dapat diketahuinya berapa besar kemungkinan seorang pelanggan membeli roti bersamaan dengan susu. Dengan pengetahuan tersebut, pemilik pasar swalayan dapat mengatur penempatan barangnya atau merancang kampanye pemasaran dengan memakai kupon diskon untuk kombinasi barang tertentu. Analisis asosiasi sering disebut *market basket analysis*. *Asociation rule* juga dikenal sebagai salah satu teknik *data mining* yang menjadi dasar dari berbagai teknik *data mining* lainnya. Secara khusus salah satu tahap *association rule* yang menarik perhatian banyak peneliti untuk menghasilkan algoritma yang efesien adalah analisis pola frekuensi tinggi (*frequent pattern mining*).

Penting atau tidaknya suatu aturan asosiatif dapat diketahui dengan dua parameter, yaitu *support* dan *confidence*. *Support* (nilai penunjang) adalah presentase kombinasi item tersebut dalam database, sedangkan *confidence* (nilai kepastian) adalah kuatnya hubungan antar item dalam *association rule*. Berikut rumus untuk menghitung *support* (nilai penunjang) dan *confidence* (nilai kepastian):

*Support* (A) =

*Confidence* P (B|A) =

*Association rule* biasanya dinyatakan dalam bentuk:

{roti, mentega} {susu} (support = 40%, confidence = 50)

Aturan tersebut berarti 50% dari transaksi di *database* yang memuat *item* roti dan mentega juga memuat *item* susu. Sedangkan 40% dari seluruh transaksi yang ada di *database* memuat ketiga *item* itu. Dapat juga diartikan, seorang konsumen yang membeli roti dan mentega punya kemungkinan 50% untuk juga membeli susu. Aturan ini cukup signifikan karena mewakili 40% dari catatatn transaksi selama ini. Dengan begitu *association rule* dapat didefinisikan sebagai suatu proses untuk menemukan semua aturan asosiasi yang memenuhi syarat minimum untuk *support* (*minimum support*) dan syarat minimum untuk *confidence* (*minimum confidence*).

1. **Algoritma *FP-Growth* (*Frequent Pattern Growth*)**

*Fp-growth* adalah salah satu alternatif algoritma yang dapat digunakan untuk menentukan himpunan data yang paling sering muncul (*frequent itemset*) dalam sebuah kumpulan data. *Fp-growth* menggunakan pendekatan yang berbeda dari paradigma pada *apriori*, dengan kata lain algoritma *fp-growth* merupakan pengembangan dari algoritma *apriori*. Sehingga kekurangan dari algoritma *apriori* dapat diperbaiki oleh algoritma *fp-growth*.

Pada algoritma *apriori* diperlukan *generate candidate* untuk mendapatkan *frequent itemsets*. Akan tetapi, di algoritma *FP-growth generate candidate* tidak dilakukan karena *FP-growth* menggunakan konsep pembangunan *tree* dalam pencarian *frequents itemsets*. Hal tersebutlah yang menyebabkan algoritma *FP-growth* lebih cepat dari algoritma *apriori*. Karakteristik pada algoritma *FP-growth* adalah struktur data yang digunakan adalah *tree* yang disebut *FP-tree*, dengan menggunakan *FP-tree*, algoritma *FP-growth* dapat langsung mengekstrak *frequent itemset* dari *FP-tree*.

1. **Struktur *FP-Tree***

*FP-tree* dibangun dengan memetakan setiap data transaksi ke dalam setiap lintasan tertentu dalam *FP-tree,* karena dalam setiap transaksi yang dipetakan mungkin ada transaksi yang memiliki *item* yang sama, maka lintasannya memungkinkan untuk saling menimpa. Semakin banyak data transaksi yang memiliki *item* yang sama, maka proses pemampatan dengan struktur data *FP-tree* semakin efektif. Kelebihan dari *FP-tree* adalah hanya memerlukan dua kali pemindaian data transaksi yang terbukti sangat efisien.

(Samuel) *Fp-tree* adalah sebuah pohon dengan definisi sebagai berikut:

1. *Fp-tree* dibentuk oleh sebuah akar yang diberi label *null*, sekumpulan pohon yang beranggotakan *item-item* tertentu dan sebuah table *frequent header*.
2. Setiap simpul dalam *Fp-tree* mengandung tiga informasi penting yaitu label *item* menginformasikan jenis *item* yang direpresentasikan simpul tersebut, *support count* merepresentasikan jumlah lintasan transaksi yang melalui simpul tersebut, dan *pointer* penghubung yang menghubungkan simpul-simpul dengan label *item* sama antar lintasan ditandai dengan garis panah putus-putus.
3. **Tahapan Algoritma *Fp-Growth***

*FP-growth* merupakan salah satu algoritma yang termasuk dalam *association rule* mining. Penggalian *itemset* yang *frequent* dengan menggunakan algoritma *FP-growth* akan dilakukan dengan cara membangkitkan struktur data *tree* atau disebut dengan *FP-tree*. (Samuel) Algoritma *FP-growth* dibagi menjadi 3 tahapan utama yaitu sebagai berikut:

1. Tahap pembangkitan *Conditional Pattern Base* merupakan subdatabase yang berisi *prefix path* (lintasan *prefix*) dan *suffix pattern* (pola akhiran). Pembangkitan *condtional pattern base* didapatkan melalui *Fp-tree* yang telah dibangun sebelumnya.
2. Tahap pembangkitan *conditional FP-tree,* pada tahap ini *support count* dari setiap *item* pada setiap *conditional pattern base* dijumlahkan, lalu setiap *item* yang memiliki jumlah *support count* lebih besar sama dengan minimum *support count* akan dibangkitkan dengan *conditional Fp-tree*
3. Tahap pencarian *frequent itemset* apabila *conditional Fp-tree* merupakan lintasan tunggal (*single path*), maka didapatkan *frequent itemset* dengan melakukan kombinasi *item* untuk setiap *conditional Fp-tree*. Jika bukan lintasan tunggal, maka dilakukan pembangkitan *Fp-growth* secara rekursif.
4. **Definisi Pembelian**

Pembelian merupakan kegiatan utama untuk menjamin kelancaran transaksi penjualan yang terjadi dalam suatu perusahaan. Dengan adanya pembelian, perusahaan dapat secara mudah menyediakan sumber daya yang diperlukan organisasi secara efisien dan efektif. Adapun pengertian pembelian menurut para ahli sebagai berikut :

Menurut Soemarso (2007:08) dalam buku *Akuntansi Suatu Pengantar,* Pembelian (*purchase*) adalah akun yang digunakan untuk mencatat semua pembelian barang dagang dalam satu periode.

Dari definisi diatas dapat disimpulkan bahwa pembelian merupakan kegiatan yang dilakukan untuk pengadaan barang yang dibutuhkan perusahaan dalam menjalankan usahanya dimulai dari pemilihan sumber sampai memperoleh barang.

1. **Definisi Penjualan**

Penjualan (*sales*) adalah suatu aktivitas atau bisnis menjual produk barang atau jasa. Penjualan merupakan fungsi pemasaran yang sangat penting dan menentukan bagi perusahaan dalam mencapai tujuan perusahaan yakni memperoleh laba guna menjaga kelangsungan perusahaan.

Menurut (Bayu Swastha, 2009) dalam buku *Azas-azas Marketing,* penjualan adalah ilmu dan seni mempengaruhi pribadi yang dilakukan oleh penjual untuk mengajak orang lain bersedia membeli barang atau jasa yang ditawarkan. Jadi dalam bukunya menerangkan bahwa penjualan yaitu proses menawarkan barang atau produk kepada konsumen dengan cara merayu konsumen tersebut.

Sedangkan menurut (Moekijat, 1990) dalam buku berjudul *Kamus Manajemen,* pengertian penjualan adalah suatu kegiatan yang ditujukan untuk mencari pembeli, mempengaruhi, dan memberi petunjuk agar pembelian dapat menyesuaikan kebutuhannya dengan produksi yang ditawarkan serta mengadakan perjanjian mengenai harga yang menguntungkan kedua belah pihak.

1. **SDLC (*System Development Life Cycle*)**

SDLC atau *Software Development Life Cycle* atau sering disebut juga *System Development Life Cycle* adalah proses mengembangkan atau mengubah suatu sistem perangkat lunak dengan menggunakan model-model dan metodologi yang digunakan orang untuk mengembangkan sistem-sistem perangkat lunak sebelumnya (berdasarkan *best practice* atau cara-cara yang sudah teruji baik). Seperti halnya proses metamorphosis pada kupu-kupu, untuk menjadi kupu-kupu yang indah maka dibutuhkan beberapa tahap untuk dilalui, sama halnya dengan membuat perangkat lunak, memiliki daur tahapan yang dilalui agar menghasilkan perangkat lunak yang berkualitas.

1. **Tahapan SDLC (*System Development Life Cycle*)**

Tahapan-tahapan yang ada pada SDLC secara global adalah sebagai berikut:

1. Inisiasi (*initiation*)

Tahapan ini biasanya ditandai dengan pembuatan proposal proyek perangkat lunak.

1. Pengembangan konsep sistem

Mendefinisikan lingkup konsep termasuk dokumen lingkup sistem, analisis manfaat biaya, manajemen rencana, dan pembelajaran kemudahan sistem.

1. Perencanaan (*planning*)

Mengembangkan rencana manajemen proyek dan dokumen perencanaan lainnya. Menyediakan dasar untuk mendapatkan sumber daya (*resources*) yang dibutuhkan untuk memperoleh solusi.

1. Analisis kebutuhan (*requirements analysis*)

Menganalisis kebutuhan pemakai sistem perangkat lunak (*user*) dan mengembangkan kebutuhan *user*. Membuat dokumen kebutuhan fungsional.

1. Desain (*design*)

Mentransformasikan kebutuhan detail menjadi kebutuhan yang sudah lengkap, dokumen desain sistem fokus pada bagaimana dapat memenuhi fungsi-fungsi yang dibutuhkan.

1. Pengembangan (*development*)

Mengonversi desain ke sistem informasi yang lengkap termasuk bagaimana memperoleh dan melakukan instalasi lingkungan sistem yang dibutuhkan; membuat basis data dan mempersiapkan prosedur kasus pengujian; mempersiapkan berkas atau file pengujian, pengodean, pengompilasian, memperbaiki dan membersihkan program; peninjauan pengujian.

1. Integrase dan pengujian (*integration and test*)

Mendemonstrasikan sistem perangkat lunak bahwa telah memenuhi kebutuhan yang dispesifikasikan pada dokumen kebutuhan fungsional. Dengan diarahkan oleh staf penjamin kualitas (*quality assurance*) dan *user*. Menghasilkan laporan analisis pengujian.

1. Implementasi (*implementation*)

Termasuk pada persiapan implementasi, implementasi perangkat lunak pada lingkungan produksi (lingkungan pada *user*) dan menjalankan resolusi dari permasalahan yang teridentifikasi dari fase integrase dan pengujian.

1. Operasi dan pemeliharaan (*operations and maintenance*)

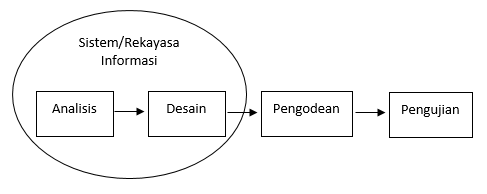
Mendekripsikan pekerjaan untuk mengoperasikan dan memelihara sistem informasi pada lingkungan produksi (lingkungan pada *user*), termasuk implementasi akhir dan masuk pada proses peninjauan.

1. Disposisi (*disposition*)

Mendeskripsikan aktifitas akhir dari pengembangan sistem dan membangun data yang sebenarnya sesuai dengan aktivitas *user*.

1. **Model Air Terjun (*Waterfall*)**

Metode SDLC air terjun (*waterfall*) sering juga disebut model sekuensial linier (*sequential linear*) atau alur hidup klasik (*classic life cycle*). Model air terjun menyediakan pendekatan alur hidup perangkat lunak secara sekuensial atau terurut dimulai dari analisis desain, pengodean, pengujian, dan tahap pendukung (*support*). Berikut adalah gambar dari model air terjun:



**Gambar 2.6** Ilustrasi Model *Waterfall* (M. Shalahuddin, 2016).

Berikut penjelasan tahapan pengembangan sistem dengan metode *waterfall*:

1. 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*.

1. Desain

Desain perangkat lunak adalah proses multi langkah yang fokus pada desain pembuatan program perangkat lunak termasuk struktur data, arsitektur perangkat lunak, representasi antarmuka, dan prosedur pengodean. Tahap ini mentranslasikan kebutuhan perangkat lunak dari tahap analisis kebutuhan ke representasi desain agar dapat diimplementasikan menjadi program pada tahap selanjutnya.

1. 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.

1. Pengujian

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

1. Pemeliharaan (*Maintenance*)

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

1. **Perangkat Lunak dan Bahasa Pemrograman**
2. ***Cascading Style Sheet* (CSS)**
3. ***Hyper Text Markup Language* (HTML)**
4. ***Hypertext Preprocesor* (PHP)**
5. ***Database* (Basis Data)**
6. ***Database Management System* MySQL**
7. ***Web Server* XAMPP**
8. **Pemodelan dan UML (*Unified Modeling Language*)**

Salah satu pemodelan yang saat ini paling banyak digunakan adalah UML. UML (*Unified Modeling Language*) merupakan salah satu standar Bahasa yang banyak digunakan di dunia industri untuk mendefinisikan *requirement*, membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek.

1. **Pemodelan**

Pemodelan adalah gambaran dari realita yang simple dan dituangkan dalam bentuk pemetaan dengan aturan tertentu. Pemodelan dapat menggunakan bentuk yang sama dengan realitas misalnya jika seorang arsitek ingin memodelkan sebuah gedung yang akan dibangun maka dia akan memodelkannya dengan membuat sebuah maket (tiruan) arsitektur yang akan dibangun di mana maket itu akan dibuat semirip mungkin dengan desain gedung yang akan dibangun agar arsitektur gedung yang diingikan dapat terlihat. Pada dunia pembangunan perangkat lunak sistem informasi juga perlu pemodelan. Pemodelan perangkat lunak digunakan untuk mempermudah langkah berikutnya dari pengembangan sebuah system informasi sehingga lebih terencana. Seperti halnya maket, pemodelan pada pembangunan perangkat lunak digunakan untuk memvisualkan perangkat lunak yang akan dibuat.

Perangkat pemodelan adalah suatu model yang digunakan untuk menguraikan sistem menjadi bagian-bagian yang dapat diatur dan mengomunikasikan ciri konseptual dan fungsional kepada pengamat.

Peran perangkat pemodelan :

1. Komunikasi

Perangkat pemodelan dapat digunakan sebagai alat komunikasi antara pemakai dengan analis sistem maupun *developer* dalam pengembangan sistem.

1. Eksperimen

Pengembangan sistem yang bersifat “*trial and error*”.

1. Prediksi

Model meramalkan bagaimana suatu sistem akan bekerja.

1. **Pengenalan UML**

Pada perkembangan teknik pemrograman berorientasi objek, muncullah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung. UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientas objek.

Seperti yang kita ketahui bahwa banyak hal di dunia sistem informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan dan konteksnya. Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunaannya. Jadi belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan. Secara analogi jika dengan bahasa kita gunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar.

1. **Diagram UML**

Pada UML 2.3 terdiri dari 13 macam diagram yang dikelompokan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada gambar dibawah.



**Gambar 2.X** Diagram UML

Berikut penjelasan singkat dari pembagian kategori tersebut.

1. *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. *Behavior diagrams* kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahaan yang terjadi pada sebuah sistem.
3. *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem.
4. ***Class Diagram***

Diagram kelas menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun system. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

1. Atribut merupakan variable-variabel yang dimiliki oleh kelas.
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Diagram kelas dibuat agar pembuat program atau *programmer* membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak sinkron serta kelas-kelas yang ada pada struktur sistem dapat melakukan fungsi-fungsi sesuai kebutuhan sistem.

Berikut adalah simbol-simbol yang ada pada diagram kelas:

Tabel 2.2 Simbol-simbol *class diagrams*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| Kelas   |  | | --- | | nama\_kelas | | +atribut | | +operasi() | | Kelas pada struktur sistem |
| Antar muka/*interface*  **nama\_interace** | Sama dengan konsep *interface* dalam pemrograman berorientasi objek |
| Asosiasi/*association* | Relasi antarkelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity* |
| Asosiasi berarah / *directed association* | Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai *multiplicity* |
| Generalisasi | Relasi antarkelas dengan makna generalisasi-spesialisasi (umum khusus) |
| Kebergantungan/*dependency* | Relasi antarkelas dengan makna kebergantungan antar elas |
| Agregasi/*aggregation* | Relasi antarkelas dengan makna semua bagian (*whole part*) |

1. ***Object Diagram***

Diagram objek menggambarkan struktur sistem dari segi penamaan objek dan jalannya objek dalam system. Pada diagram objek harus dipastikan semua kelas yang sudah didefinisikan pada diagram kelas harus dipakai objeknya, karena jika tidak, pendefinisian kelas itu tidak dapat dipertanggungjawabkan. Diagram objek juga berfungsi untuk mendefinisikan contoh nilai atau isi dari atribut tiap kelas. Hubungan *link* pada diagram objek merupakan hubungan memakai dan dipakai dimana dua buah objek akan dihubungkan oleh *link* jika ada objek yang dipakai oleh objek lainnya. Berikut adalah smbol-simbol yang ada pada diagram objek:

Table 2.3 Simbol-simbol *object diagram*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| Objek   |  | | --- | | nama\_objek : nama\_kelas | | atribut = nilai | | Objek dari kelas yang berjalan saat sistem dijalankan |
| *Link* | Sama dengan konsep *interface* dalam pemrograman berorientasi objek |

1. ***Componenet Diagram***

Diagram komponen dibuat untuk menunjukan organisasi dan ketergantungan diantara kumpulan komponen dalam sebuah sistem. Diagram komponen fokus pada komponen sistem yang dibutuhkan dan ada di dalam sistem. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut:

1. *Source code* program perangkat lunak
2. Komponen *executable* yang dilepas ke *user*
3. Basis data secara fisik
4. Sistem yang harus beradaptasi dengan sistem lain
5. *Framework system, framework* pada perangkat lunak merupakan kerangka kerja yang dibuat untuk memudahkan pengembangan dan pemeliharaan aplikasi, contohnya *Struts* dari *Apache* yang menggunakan prinsip desain *Model*-*View*-*Controller* (MVC), dimana *source code* program dikelompokan berdasarkan fungsinya.

Berikut adalah simbol-simbol yang ada pada diagram komponen:

Tabel 2.4 Simbol-simbol *component diagram*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| Package  package | *Package* merupakan sebuah bungkusan dari satu atau lebih komponen |
| Komponen  **nama\_komponen** | Komponen sistem |
| Kebergantungan/*dependency* | Kebergantungan antar komponen, arah panah mengarah pada komponen yang dipakai |
| Antar muka/*interface*  **nama\_interace** | Sama dengan konsep *interface* pada pemrograman berorientasi objek, yaitu sebagai antarmuka komponen agar tidak mengakses langsung komponen |
| *Link* | Relasi antar komponen |

1. ***Composite Structure Diagram***

Diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (*runtime*) dari *instance* yang saling terhubung. Dapat menggambarkan struktur di dalam kelas atau kolaborasi. Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait untuk menjalankan fungsi mesin tersebut, menggambarkan aliran data *router* pada jaringan komputer, dan lain-lain.

Berikut adalah simbol-simbol yang ada pada diagram *composite structure*:

Tabel 2.5 Simbol-simbol *composite structure diagram*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| *Property*  roleName : TypeName | *Property* adalah satu set dari suatu *instance*.  *roleName* : peran / nama / identitas dari *property* (opsional)  *TypeName* : tipe kelas dari *property* (harus ada) |
| *Connector*  nama connector : tipe connector | *Connector* adalah cara komunikasi dari 2 buah *instance*.  *connName* : nama *connector* (opsional)  *connType* : tipe *connector* (opsional) |
| Port    portName : EntityName[n]  Pemakaiannya adalah sebagai berikut:  property    property | *Port* adalah cara yang digunakan dalam diagram *composite structure* tanpa menampilkan detail internal dari suatu sistem.  *Port* digambarkan dalam bentuk kotak kecil yang menempel atau di dalam suatu *property*.  *Port* digambarkan menempel *property* jika fungsi tersebut dapat diakses *public*.  Sedangkan *port* digambarkan didalam suatu *property* jika fungsi tersebut bersifat *protected*. |
| Kelas  class | Kelas; jika yang akan dijabarkan strukturnya adalah sebuah kelas |

1. ***Package Diagram***

*Package diagram* menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML. Hampir semua diagram dalam UML dapa dikelompokan menggunakan  *package diagram*.

Berikut ini simbol-simbol yang ada pada *package diagram*:

**Tabel 2.6** Simbol-simbol *package diagram*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| *Package*  package | *Package* merupakan sebuah bungkusan dari satu atau lebih kelas atau elemen diagram UML lainnya. |
| Elemen dalam *package* digambarkan di dalam *package*  Perpustakaan  Pustaka  Anggota  Peminjaman | |
| Elemen dalam *package* digambarkan di luar *package*  Perpustakaan  Pustaka  atribut  metode  Peminjaman  atribut  metode  Anggota  atribut  metode | |

1. ***Deployment Diagram***

Diagram *deployment* atau *deployment diagram* menunjukan konfigurasi komponen dalam proses eksekusi aplikasi. Diagram *deployment* juga dapat digunakan untuk memodelkan hal-hal berikut:

1. Sistem tambahan (*embedded system*) yang menggambarkan rancangan *device, node,* dan *hardware*.
2. Sistem *client*/*server* misalnya seperti gambar berikut:

Client

**browser**

Server

**components** business process

**printer**

**php server**

**Gambar 2.x** Diagram *deployment* sistem *client*/*server*

1. Sistem terdistribusi murni
2. Rekayasa ulang aplikasi

Berikut adalah symbol-simbol yang ada pada diagram *deployment*:

**Tabel 2.7** Simbol-simbol *package diagram*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| Package  package | *Package* merupakan sebuah bungkusan dari satu atau lebih *node* |
| *Node*  nama\_node | Biasanya mengacu pada perangkat keras (*hardware*), perangkat lunak yang tidak dibuat sendiri, jika di dalam *node* disertakan komponen untuk mengkonsistenkan rancangan maka komponen yang diikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen |
| Kebergantungan/*dependency* | Kebergantungan antar *node*, arah panah mengarah pada *node* yang dipakai |
| *Link* | Relasi antar *node* |

1. ***Use Case Diagram***

*Use case* atau diagram *use case* merupakan pemodelan untuk kelakukan (*behavior*) terhadap sistem informasi atau aplikasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut.

Berikut simbol-simbol yang ada pada diagram *use case*:

**Tabel 2.8** Simbol-simbol *use case diagram* (M. Shalahuddin, 2016)

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| *Use case* | Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor; biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama *use case* |
| Aktor/*Actor*    **nama aktor** | Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor |
| Asosiasi / *association* | Komunikasi antara aktor dan *use case* yang berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor |
| Ekstensi / *extend*  << extend >> | Relasi *use case* tambahan ke sebuah *use case*  dimana *use case* yang ditambahkan dapat berdiri sendiri walaupun tanpa *use case* tambahan itu; mirip dengan prinsip *inheritance* pada pemrograman berorientasi objek; biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan, misal:  << extend >>    << extend >>  Arah panah mengarah pada *use case* yang ditambahkan; biasanya *use case* yang menjadi *extend*-nya merupakan jenis yang sana dengan *use case* yang menjadi induknya |
| Generalisasi / *generalization* | Hubungan generalisasi dan spesialisasi (umum - khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya, misalnya:  arah panah mengarah pada *use case* yang menjadi generalisasinya (umum) |
| Menggunakan / *include* / *uses* | Relasi *use case* tambahan ke sebuah *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat dijalankan *use case* ini  Ada dua sudut pandang yang cukup besar mengenai *include* di *use case*:   1. *Include* berarti *use case* yang ditambahkan akan selalu dipanggil saat *use case* tambahan dijalankan, misal:   << include >>   1. *Include* berarti *use case* yang tambahan akan selalu melakukan pengecekan apakah *use case* yang ditambahkan telah dijalankan sebelum *use case* tambahan dijalankan, misal:   << include >>  Kedua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi yang dibutuhkan. |

1. ***Activity Diagram***

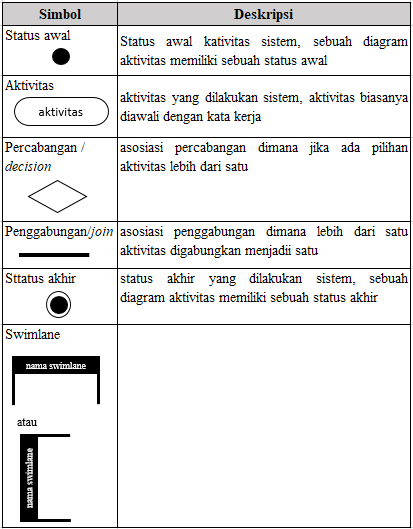
Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut:

1. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan
2. Urutan atau pengelompokan tampilan dari sistem/*user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya
4. Rancangan menu yang ditampilkan pada perangkat lunak

Berikut adalah simbol-simbol yang ada pada diagram aktivitas:

**Tabel 2.9** Simbol-simbol *activity diagram* (M. Shalahuddin, 2016)



1. ***State Machine Diagram***

*State machine diagram* atau dalam bahasa indonesia disebut diagram mesin status atau sering juga disebut diagram status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek. Jika diagram sekuen digunakan untuk interaksi di dalam sebuah objek. Perubahan tersebut digambarkan dalam suatu graf berarah. *State machine diagram* merupakan pengembangan dari diagram *Finite State Automata* dengan penambahan beberapa fitur dan konsep baru. *State machine diagram* cocok digunakan untuk menggambarkan alur interaksi pengguna sistem. Berikut ini adalah contoh gambar diagram mesin status.



**Gambar 2.x** Contoh *state machine diagram*

Berikut ini komponen-komponen dasar yang ada dalam *state machine diagram*:

**Tabel 2.10** Simbol-simbol *state machine diagram*

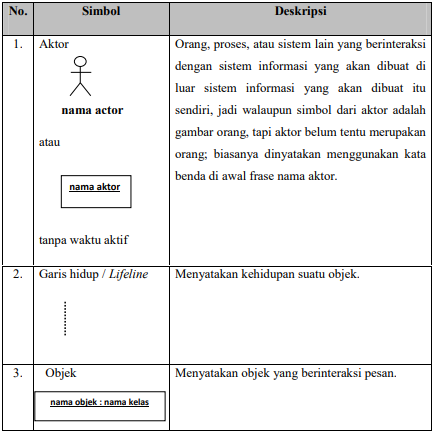
|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| Start / status awal (*initial state*) | *Start* atau *initial state* adalah *state* atau keadaan awal pada saat sistem mulai hidup. |
| *End* / Status awal (*final state*) | *End* atau *final state* adalah *state* keadaan akhir dari daur hidup suatu sistem. |
| *Event*  **event** | *Event* adalah kegiatan yang menyebabkan berubahnya status mesin. |
| *State*  state | Sistem pada waktu tertentu. *State* dapat berubah jika ada *event* tertentu yang memicu perubahan tersebut. |

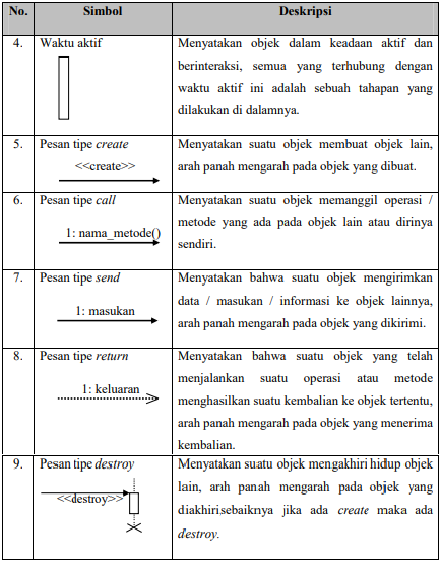
1. ***Sequence Diagram***

Diagram sekuen mengambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case*.

Berikut adalah simbol-simbol yang ada pada diagram sekuen:

**Tabel 2.11** Simbol-simbol *sequence diagram*





1. ***Communication Diagram***

*Communication diagram* pada UML versi 2.x adalah penyederhanaan dari diagram kolaborasi (*collaboration diagram*) pada UML versi 1.x. *Collaboration diagram* sudah tidak muncul lagi pada UML versi 2.x. Diagram komunikasi sebenarnya adalah diagram kolaborasi tetapi dibuat untuk tiap sekuen. Diagram komunikasi menggambarkan interaksi antar objek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari diagram kelas, diagram sekuen, dan diagram *use case* untuk mendeskripsikan gabungan antara struktur statis dan tingkah laku dinamis dari suatu sistem.

Diagram komunikasi mengelompokan *message* pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi/metode yang dijalankan antara objek yang satu dan objek yang lainnya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada semua diagram sekuen. Penomoran metode dapat dilakukan berdasarkan urutan dijalankannya metode/operasi diantara objek yang satu dengan objek lainnya atau objek itu sendiri.

Berikut adalah symbol-simbol yang ada pada *communication diagram:*

**Tabel 2.12** Simbol-simbol *communication diagram*

|  |  |
| --- | --- |
| **Simbol** | **Deskripsi** |
| Objek  nama\_objek : nama\_kelas | Objek yang melakukan interaksi pesan |
| *Link* | Relasi antar objek yang menghubungkan objek satu dengan lainnya atau dengan dirinya sendiri  nama\_objek : nama\_kelas |
| Arah pesan / *stimulus* | Arah pesan yang terjadi, jika pada suatu *link* ada dua arah pesan yang berbeda maka arah juga digambarkan dua arah pada dua sisi *link* |

1. ***Timing Diagram***

*Timing diagram* merupakan diagram yang focus pada penggambaran terkait batasan waktu. *Timing diagram* digunakan untuk menggambarkan tingkah laku sistem dalam periode waktu tertentu. *Timing diagram* biasanya digunakan untuk mendeskripsikan operasi dari alat digital karena penggambaran secara visual akan lebih mudah diahami daripada dengan kata-kata. Berikut ini adalah aliran sirkuit yang menggambarkan operasi *timing diagram*. Gambar dibawah menyatakan aliran listrik. Status OPEN = o artinya *switch* dalam posisi terbuka (tidak terhubung), sedangkan CLOSED = 1 adalah posisi *switch* terhubung. Lampu akan menyala, status ON = 1, jika kedua *switch* terhubung.

OUTPUT LIGHT X

1

ON = 1 OFF = 0

INPUT SW A

INPUT SW B

OPEN = 0 CLOSED = 1

**Gambar 2.x** Contoh aliran sirkuit

Berikut ini contoh gambar pemanfaatan *timing diagram* dalam menggambarkan seluruh status dalam gambar aliran sirkuit di atas:

Aliran waktu pada *timing diagram* dibaca dari kiri ke kanan. Pada gambar *timing diagram* di atas dapat terlihat bahwa lampu akan menyala jika kedua *switch* atau INPUT A dan INPUT B dalam posisi CLOSED = 1. Jika slah satu *switch* tidak terhubung maka lampu tidak akan menyala.

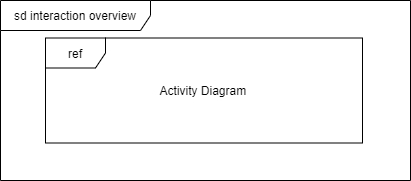
1. ***Interaction Overview Diagram***

*Interaction overview diagram* mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. *Interaction overview diagram* adalah bentuk aktivitas diagram yang setiap titik merepresentasikan diagram interaksi. Interaksi diagram dapat meliputi diagram sekuen, diagram komunikasi, *interaction overview diagram* dan*, timing diagram*. Hampir semua notasi pada *interaction overview diagram* sama dengn notasi pada diagram aktivitas. Tambahan pada *interaction overview diagram* adalah i*nteraction occurrence* dan *interaction element*.

1. *Interaction Occurrence*

*Interaction occurrence* atau kejadian interaksi adalah referensi untuk diagram interaksi yang ada. Sebuah *interaction occurrence* ditujukan sebagai *frame* referensi (*frame* dengan tulisan “*ref*” di pojok kiri atas). Nama diagram yang sedang direferensikan ditunjukan pada tengah *frame*.

Berikut gambar contoh dari *interaction occurrence*:



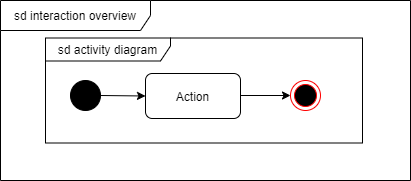
**Gambar 2.x** Contoh *interaction occurance*

sd di atas kependekan dari *sequence diagram*.

1. *Interaction Element*

*Interaction element* atau element interaksi mirip *interaction occurrence*. Perbedaannya di dalam *interaction element* menampilkan isi diagram yang direferensikan secara langsung, sedangkan *interaction occurrence* hanya menampilkan diagram yang direferensikan.

Berikut adalah contoh dari *interaction element*:



**Gambar 2.x** Contoh *interaction element*

1. **Navigasi Diagram**
2. ***Blackbox Testing***
3. **Tinjauan Pustaka**
4. **Pola Pemikiran Pemecahan Masalah**