BAB II
LANDASAN TEORI

Dalam penyusunan Tugas Akhir ini dibutuhkan beberapa sumber untuk dapat lebih memahami teori dari apa yang telah, dan akan dilaksanakan dalam penyusunan Tugas Akhir ini.

2.1. Aplikasi

Perangkat lunak aplikasi yaitu perangkat lunak yang digunakan untuk membantu pemakai komputer untuk melaksanakan pekerjaannya. Jika ingin mengembangkan program aplikasi sendiri, maka untuk menulis program aplikasi tersebut, dibutuhkan suatu bahasa pemrograman, yaitu *language software*, yang dapat berbentuk *assembler*, *compiler* ataupun *interpreter*. Jadi *language software* merupakan bahasanya dan program yang ditulis merupakan program aplikasinya. *Language software* berfungsi agar dapat menulis program dengan bahasa yang lebih mudah, dan akan menterjemahkannya ke dalam bahasa mesin supaya bisa dimengerti oleh komputer. Bila hendak mengembangkan suatu program aplikasi untuk memecahkan permasalahan yang besar dan rumit, maka supaya program aplikasi tersebut dapat berhasil dengan baik, maka dibutuhkan prosedur dan perencanaan yang baik dalam mengembangkannya.

Sekarang banyak sekali program-program aplikasi yang tersedia dalam bentuk paket-paket program. Ini adalah program-program aplikasi yang sudah ditulis oleh orang lain atau perusahaan-perusahaan perangkat lunak. Beberapa perusahaan perangkat lunak telah memproduksi paket-paket perangkat lunak yang mempunyai reputasi internasional. Program-program paket tersebut dapat diandalkan, dapat memenhi kebutuhan pemakai, dirancang dengan baik, relatip bebas dari kesalahan-kesalahan, *user friendly* (mudah digunakan), mempunyai dokumentasi manual yang memadai, mampu dikembangkan untuk kebutuhan mendatang, dan didukung perkembangannya. Akan tetapi, bila permasalahannya bersifat khusus dan unik, sehingga tidak ada paket-paket program yang sesuai untuk digunakan, maka dengan terpaksa harus mengembangkan program aplikasi itu sendiri.[9]

2.1.1. Definisi Sistem

Terdapat dua kelompok pendekatan didalam mendefinisikan sistem, yaitu yang menekankan pada prosedurnya dan yang menekankan pada komponennya. Pendekatan sistem yang lebih menekankan pada prosedur mendefinisikan sistem sebagai berikut :

Suatu sistem adalah suatu jaringan kerja dari prosedur-prosedur yang saling berhubungan, berkumpul bersama-sama untuk melakukan suatu kegiatan atau untuk menyelesaikan suatu sasaran yang tertentu.[1]

Prosedur didefinisikan oleh Richard F. Neuschel sebagai berikut.

Suatu prosedur adalah suatu urutan-urutan operasi klerikal (tulis-menulis), biasanya melibatkan beberapa orang didalam satu atau lebih departemen, yang diterapkan untuk menjamin penanganan yang seragam dari transaksi bisnis yang terjadi.[1]

Pendekatan sistem yang lebih menekan pada elemennya mendefinisikan sistem sebagai berikut :

Sistem adalah kumpulan elemen-elemen yang berinteraksi untuk mencapai suatu tujuan tertentu.

2.1.2. Definisi Informasi

Robert N. Anthony dan John Dearden menyebutkan keadaan dari sistem dalam hubungannya dengan keberakhirannya dengan istilah *entropy*. Informasi yang berguna bagi sistem akan menghindari proses *entropy* yang disebut dengan *negative entropy atau negentropy*.[2]

Apakah sebenarnya informasi itu, sehingga sangat penting artinya bagi suatu sistem? Informasi dapat didefinisikan sebagai berikut :

Informasi adalah data yang diolah menjadi bentuk yang lebih baik, berguna dan lebih berarti bagi yang menerimanya.

Sumber dari informasi adalah data. Data merupakan bentuk jamak dari bentuk tunggal data-item. Data adalah kenyataan yang menggambarkan suatu kejadian-kejadian dan kesatuan yang nyata.[4]

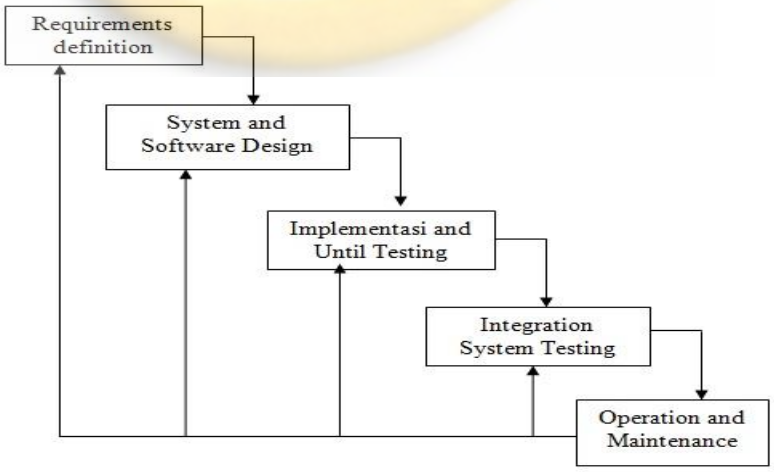
2.1.2.1. Metode Yang Digunakan

2.1.3. Metode Waterfall

Dalam perancangan aplikasi pada tugas akhir ini penulis menggunakan metode *Waterfall*. Metode *Waterfall* adalah metode yang menyarankan sebuah pendekatan yang sistematis dan sekuensial melalui tahapan-tahapan yang ada pada SDLC untuk membangun sebuah perangkat lunak.[3]

Metode ini mengambil kegiatan proses dasar sperti spesifikasi, validasi, dan evolusi, dan mempresentasikannya sebagai fase-fase proses yang berbeda seperti spesifikasi persaratan, perancangan perangkat lunak, implementasi, pengujian, dan seterusnya.

Metode ini merupakan metode pertama yang diterbitkan untuk proses pengembangan perangkat lunak diambil dari proses rekayasa lain (Royce, 1970). Metode ini diilustrasikan pada Gambar 2.1.[3]



Gambar II-1 *Waterfall Model*

Berikut adalah penjelasan dari tahap – tahap yang dilakukan dalam metode waterfall:

1. Tahap analisis dan definisi persyaratan. Pelayanan, batasan, dan tujuan sistem ditentukan melalui konsultasi dengan user sistem. Persyaratan ini kemudian didefinisikan secara rinci dan berfungsi sebagai spesifikasi sistem.
2. Tahap perancangan sistem dan perangkat lunak. Proses perancangan sistem membagi persyaratan dalam sistem perangkat keras atau perangkat lunak. Kegiatan ini menentukan arsitektur sistem secara keseluruhan. Perancangan perangkat lunak melibatkan identifikasi dan deskripsi abstraksi sistem perangkat lunak yang mendasar dan hubungan – hubungannya.
3. Tahap implementasi dan pengujian unit. Pada tahap ini, perancangan perangkat lunak direalisasikan sebagai serangkaian program atau unit program.
4. Tahap integrasi dan pengujian sistem. Unit program atau program individual diintegrasikan dan diuji sebagai sistem yang lengkap untuk menjamin bahwa persyaratan sistem telah dipenuhi. Setelah pengujian sistem, perangkat lunak dikirim kepada pelanggan.
5. Tahap operasi dan pemeliharaan. Biasanya (walaupun tidak seharusnya), ini merupakan fase siklus hidup yang paling lama. Sistem diinstal dan dipakai. Pemeliharaan mencakup koreksi dari berbagai error yang tidak ditemukan pada tahap – tahap terdahulu, perbaikan atas implementasi unit sistem dan pengembangan pelayanan sistem, sementara persyaratan – persyaratan baru ditambahkan.[3]

2.1.4. Object Oriented Programming (OOP)

Object Oriented Programming (OOP) adalah suatu metode pemrograman yang berbasiskan pada objek, secara singkat pengertian dari OOP adalah koleksi objek yang saling berinteraksi dan saling memberikan informasi satu dengan yang lainnya. Suatu program disebut dengan pemrograman berbasis obyek (OOP) karena terdapat :

1. Encapsulation (pembungkusan)

1)Variabel dan method dalam suatu obyek dibungkus agar terlindungi
2)Untuk mengakses, variabel dan method yang sudah dibungkus tadi perlu interface
3)Setelah variabel dan method dibungkus, hak akses terhadapnya bisa ditentukan.
4)Konsep pembungkusan ini pada dasarnya merupakan perluasan dari tipe data struktur

2. Inheritance (pewarisan)

1)Sebuah class bisa mewariskan atribut dan method-nya ke class yang lain
2)Class yang mewarisi disebut superclass
3)Sebuah subclass bisa mewariskan atau berlaku sebagai superclass bagi class yang lain disebut multilevel inheritance.
4)Keuntungan Penggunaan Pewarisan
5)Subclass memiliki atribut dan method yang spesifik yang membedakannya dengan superclass, meskipun keduanya mirip (dalam hal kesamaan atribut dan method).
6)Dengan demikian pada pembuatan subclass, programmer bisa menggunakan ulang source code dari superclass disebut dengan istilah reuse.
7)Class-class yang didefinisikan dengan atribut dan method yang bersifat umum yang berlaku baik pada superclass maupun subclass disebut dengan abstract class.

3. Polymorphism (polimorfisme – perbedaan bentuk)

Polimorfisme artinya penyamaran dimana suatu bentuk dapat memiliki lebih dari satu bentuk.

2.2. Unified Modeling Language (UML)

Unified Modeling Language (UML) adalah **“**keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek (OO).[5]

Selain itu UML adalah bahasa pemodelan yang menggunakan konsep orientasi object. UML dibuat oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson di bawah bendera *Rational Software Corp*. UML menyediakan notasi-notasi yang membantu memodelkan sistem dari berbagai perspektif. UML tidak hanya digunakan dalam pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.[5]

UML dideskripsikan oleh beberapa diagram, diantaranya:

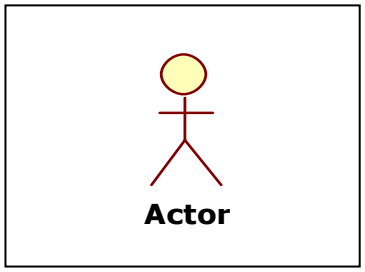
2.2.1. Use Case Diagram

Use Case Diagram digunakan untuk menggambarkan sistem dari sudut pandang pengguna sistem tersebut (*user*), sehingga pembuatan *use case diagram* lebih dititikberatkan pada fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian. Sebuah *use case diagram* merepresentasikan sebuah interaksi antara aktor dengan sistem.

Komponen-komponen yang terlibat dalam *use case* diagram :

1. Aktor

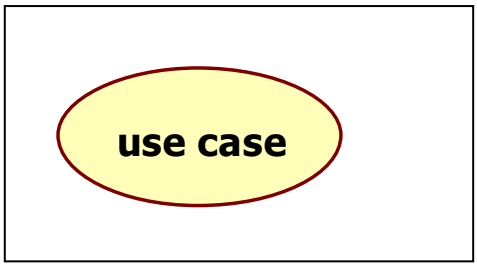
Pada dasarnya aktor bukanlah bagian dari *use case diagram*, namun untuk dapat terciptanya suatu *use case diagram* diperlukan aktor, dimana aktor tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat atau sistem lain) yang berinteraksi dengan sistem yang dibuat. Sebuah aktor mungkin hanya memberikan informasi inputan pada sistem, hanya menerima informasi dari sistem atau keduanya menerima dan memberi informasi pada sistem. Aktor hanya berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case*. Aktor digambarkan dengan *stick man*.[5]



Gambar II-2 Actor

2. Use Case

Gambaran fungsionalitas dari suatu sistem, sehingga pengguna sistem paham dan mengerti kegunaan sistem yang akan dibangun.



Gambar II-3 *Use Case*

Ada beberapa relasi yang terdapat pada *use case* diagram:

1. *Association*, menghubungkan link antar elemen.
2. *Generalization*, disebut juga pewarisan (*inheritance*), sebuah elemen dapat merupakan spesialisasi dari elemen lainnya.
3. *Dependency*, sebuah elemen bergantung dalam beberapa cara ke elemen lainnya.
4. *Aggregation*, bentuk *association* dimana sebuah elemen berisi elemen lainnya.

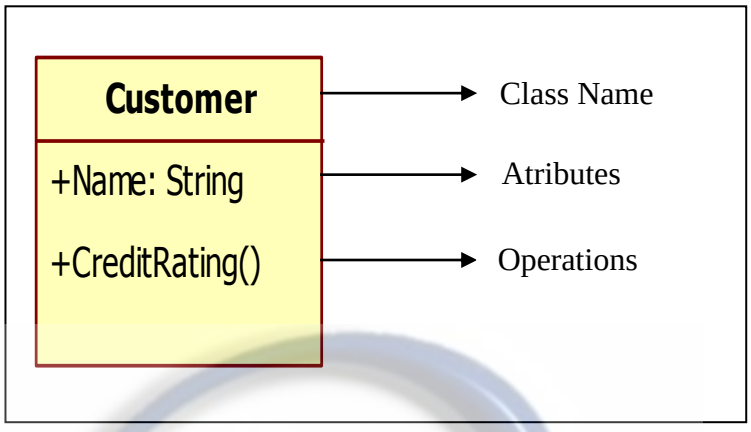
Tipe relasi yang mungkin terjadi pada *use case* diagram:

1. <<*include*>>, yaitu kelakuan yang harus terpenuhi agar sebuah *event* dapat terjadi, dimana pada kondisi ini sebuah *use case* adalah bagian dari *use case* lainnya.
2. <<*extends*>>, kelakuan yang hanya berjalan di bawah kondisi tertentu seperti menggerakkan peringatan.
3. <<*communicates*>>, merupakan pilihan selama asosiasi hanya tipe *relationship* yang dibolehkan antara aktor dan *use case.*

2.2.2. *Class* Diagram

Class adalah sebuah spesifikasi yang akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan

layanan untuk memanipulasi keadaan tersebut (metode/fungsi). *Class* diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti pewarisan, asosiasi, dan lain-lain.[5]



Gambar II-4 *Class Diagram*

Class memiliki tiga area pokok :

1. Nama (*Class Name*)
2. Atribut
3. Metode (*Operations*)

Pada UML, *class* digambarkan dengan segi empat yang dibagi beberapa bagian. Bagian atas merupakan nama dari *class*. Bagian yang tengah merupakan struktur dari *class* (atribut) dan bagian bawah merupakan sifat dari *class* (metode/operasi).

Atribut dan metode dapat memiliki salah satu sifat berikut :

1.*Private* , tidak dapat dipanggil dari luar *class* yang bersangkutan.

2.*Protected* , hanya dapat dipanggil oleh *class* yang bersangkutan dan *class* lain yang mewarisinya.

3.*Public* , dapat dipanggil oleh *class* lain.[5]

Hubungan antar *Class* **:**

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain.
2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas”).

3. Pewarisan, yaitu hubungan hirarki antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metode *class* asalnya serta bisa menambahkan fungsionalitas baru. Sehingga *class* tersebut disebut anak dari *class* yang diwarisinya.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

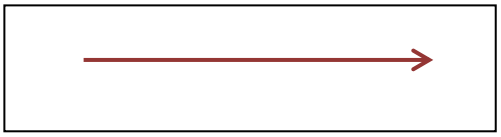
2.2.3. Statechart Diagram

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu objek dari suatu *class* dan keadaan yang menyebabkan *state* berubah. *Statechart* diagram tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

State adalah sebuah kondisi selama kehidupan sebuah objek atau ketika objek memenuhi beberapa kondisi, melakukan beberapa aksi atau menunggu sebuah *event*. *State* dari sebuah objek dapat dikarakteristikkan oleh nilai dari satu atau lebih atribut-atribut dari *class*. *State* dari sebuah objek ditemukan dengan pengujian/pemeriksaan pada atribut dan hubungan dari objek. Notasi UML untuk *state* adalah persegipanjang/bujur sangkar dengan ujung yang dibulatkan.[5]

Gambar II-5 *Start State* dan *Stop State*

Masing-masing diagram harus mempunyai satu dan hanya satu *start state* ketika objek mulai dibuat. Sebuah objek boleh mempunyai banyak *stop state*.



Gambar II-6 *State Transition*

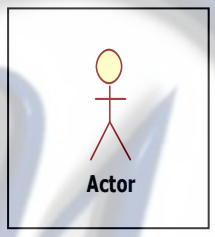
Sebuah *state transition* dapat mempunyai sebuah aksi dan/atau sebuah kondisi penjaga (*guard condition*) yang terasosiasi dengannnya, dan mungkin juga memunculkan sebuah *event*. Sebuah aksi adalah kelakuan yang terjadi ketika *state*

transition terjadi. Sebuah *event* adalah pesan yang dikirim ke objek lain di sistem. Kondisi penjaga adalah ekspresi *boolean* (pilihan Ya atau Tidak) dari nilai atribut-atribut yang mengijinkan sebuah *state transition* hanya jika kondisinya benar. Kedua aksi dan penjaga adalah kelakuan dari objek dan secara tipikal menjadi operasi.[5]

2.2.4. *Sequence* Diagram

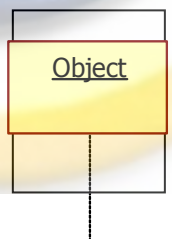
Menggambarkan interaksi antara sejumlah objek dalam urutan waktu. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antar objek yang terjadi pada titik tertentu dalam eksekusi sistem.

Dibawah merupakan simbol yang digunakan pada *sequence* diagram :



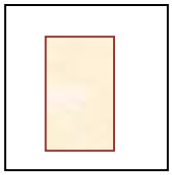
Gambar II-7 Aktor

Actor adalah pesan dari seseorang atau sistem lain yang bertukar informasi dengan sistem yang lainnya, kemudian lifeline berhenti atau mulai pada titik yang tepat.[5]



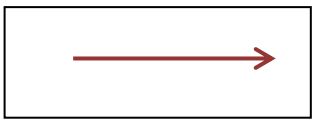
Gambar II-8 *Object lifeline*

Object lifeline menunjukkan keberadaan dari sebuah objek terhadap waktu. Yaitu objek dibuat atau dihilangkan selama suatu periode waktu diagram ditampilkan, kemudian lifeline berhenti atau mulai pada titik yang tepat.



Gambar II-9 *Activation*

Activation menampilkan periode waktu selama sebuah objek atau aktor melakukan aksi. Dalam *object lifeline*, *activation* berada diatas *lifeline* dalam bentuk kotak persegi panjang, bagian atas dari kotak merupakan inisialisasi waktu dimulainya suatu kegiatan dan yang dibawah merupakan akhir dari waktu.

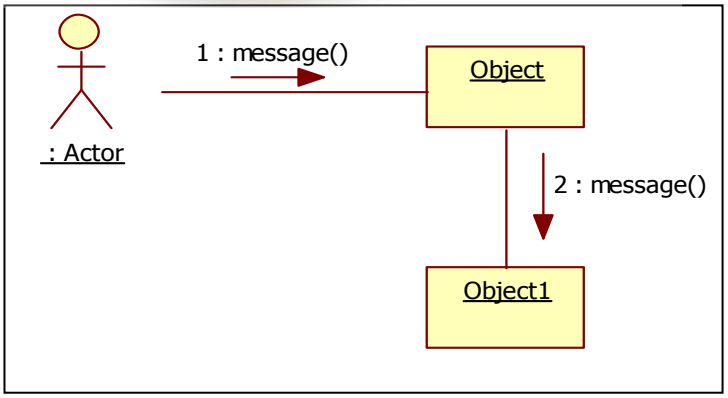


Gambar II-10 ***Message***

Message adalah komunikasi antar objek yang membawa informasi dan hasil pada sebuah aksi. *Message* menyampaikan dari *lifeline* sebuah objek kepada *lifeline* yang lain, kecuali pada kasus sebuah *message* dari objek kepada objek itu sendiri, atau dengan kata lain *message* dimulai dan berakhir pada *lifeline* yang sama.[5]

2.2.5. Collaboration Diagram

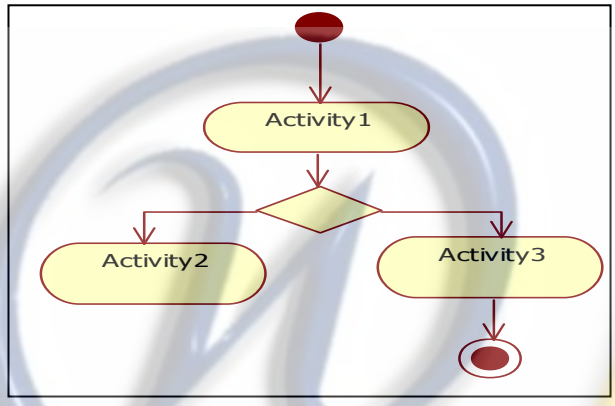
Diagram ini menggambarkan interaksi objek yang diatur objek sekelilingnya dan hubungan antara setiap objek dengan objek yang lainnya. Dalam menunjukkan pertukaran pesan, *collaboration diagram* menggambarkan objek dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagram*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*.



Gambar II-11 *Collaboration* Diagram

2.2.6. *Activity* Diagram

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktivitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya. Diagram ini sangat mirip dengan *flowchart* karena memodelkan *workflow* dari satu aktivitas ke aktivitas lainnya atau dari aktivitas ke status. Pembuatan *activity diagram* pada awal pemodelan proses dapat membantu memahami keseluruhan proses. *Activity diagram* juga digunakan untuk menggambarkan interaksi antara beberapa *use case*.[5]

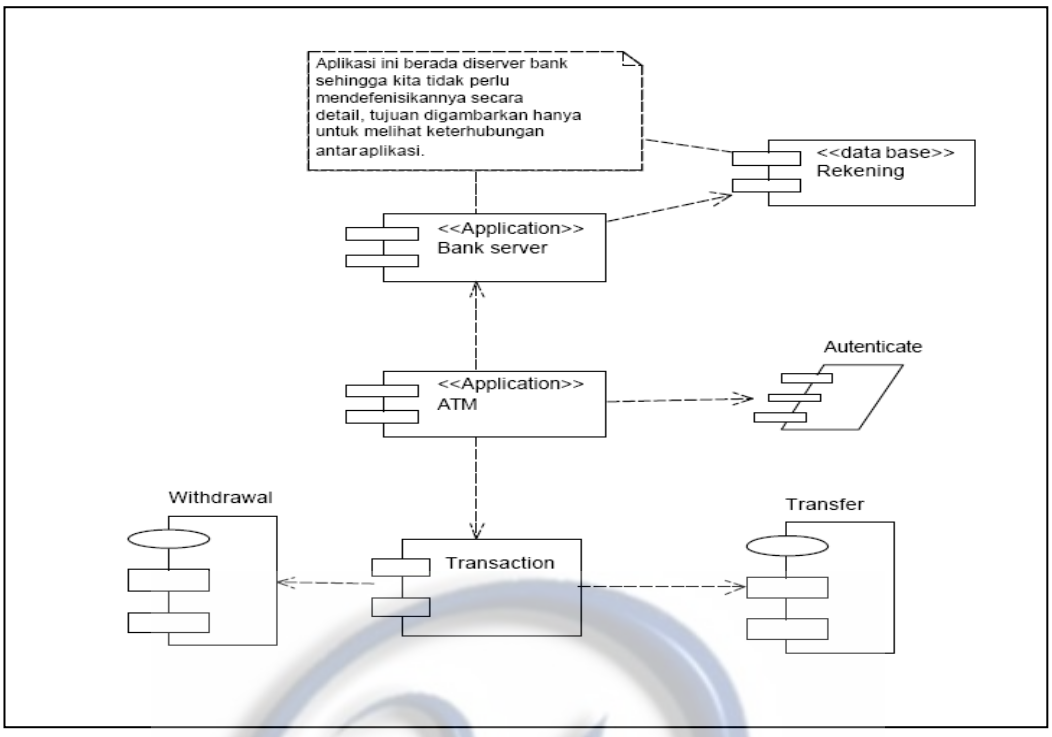


Gambar II-12 Diagram

2.2.7. Component Diagram

Menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*. Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil.[5]

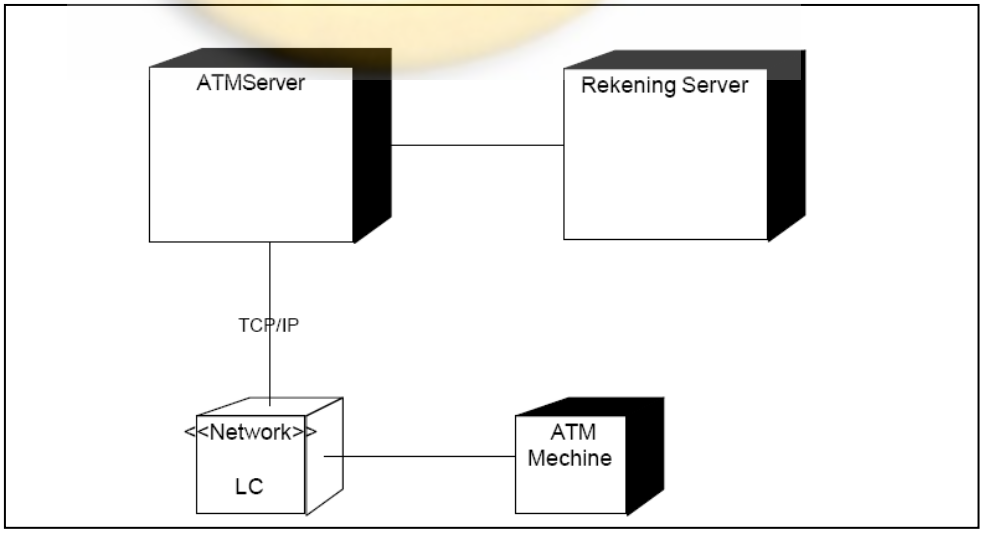
Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.



Gambar II-13 *Component* Diagram

2.2.8. Deployment Diagram

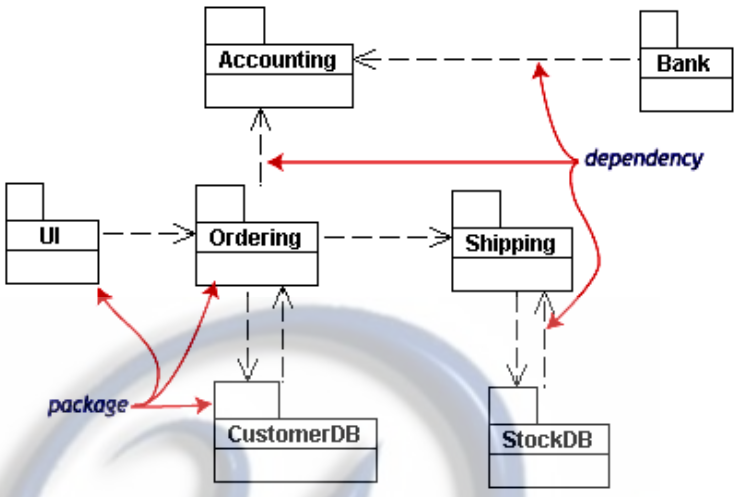
Menggambarkan arsitektur fisik dari perangkat keras dan perangkat lunak sistem, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executeable component* dan objek yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh node tertentu dan ketergantungan komponen.[5]



Gambar II-14 *Deployment* Diagram

2.2.9. Package Diagram

Adalah Penggambaran dan pengelompokan kelas-kelas yang terdapat pada perangkat lunak yang dibangun sesuai dengan fungsi atau subsistem aplikasi yang mempunyai ketergantungan satu sama lainnya.[5]



Gambar II-15 Package Diagram

2.3. Object Oriented Programming (OOP)

Object Oriented Progamming (OOP), artinya "pengelolaan program sepenuhnya diarahkan pada pembentukan objek (termasuk tipe data, kecuali tipe dasar : *int, float, double, char*), program akan lebih mudah dikembangkan karena sifatnya yang lebih modular".[9]

Pemrograman prosedural murni yang tidak menerapkan konsep *object oriented* (karena ada bahasa pemrograman prosedural yang juga sudah berorientasi objek, meskipun belum sepenuhnya) banyak menitikberatkan ke arah pembentukan fungsi-fungsi, sehingga di dalam program akan terdapat banyak sekali fungsi dan variabel yang menyulitkan pemrogram untuk mengelola dan mengembangkannya. Oleh karena itu, dengan memperhatikan kekurangan-kekurangan tersebut, maka dibentuklah bahasa yang menerapkan pendekatan *object oriented* untuk menyederhanakan fungsi-fungsi dan variabel-variabel ke dalam bentuk objek.

Dalam OOP dibutukan *memory* lebih besar dibandingkan dengan program prosedural (tradisional). Dua objek yang identik akan memerlukan dua area *memory* berbeda walaupun dari sisi data dan proses keduanya memiliki jumlah

dan jenis yang sama. Hal ini disebabkan karena data dan proses pada kedua objek tersebut dipisahkan oleh komputer. [9]

Secara garis besar yang menjadi ciri dari OOP adalah adanya proses abstraksi (*abstraction*), pengkapsulan (*encapsulation*), penurunan sifat (*inheritance*), dan polimorfisme (*polymorphism*) pada objek-objek yang dibentuk.[9]

1. **Rekayasa dan Pemodelan Sistem/ Informasi (*System/ Information Engineering and Modeling*).**

   Karena perangkat lunak selalu merupakan bagian dari sebuah sistem (bisbis) yang lebih besar, kerja dimulai dengan membangun syarat dari semua elemen sistem dan mengalokasikan beberapa subset dari kebutuhan ke perangkat lunak tersebut. Pandangan sistem ini penting ketika perangkat lunak harus berhubungan dengan elemen-elemen yang lain seperti perangkat lunak, manusia, dan database. Rekayasa dan analisis sistem menyangkut pengumpulan kebutuhan pada tingkat sistem dengan jumlah kecil analisis serta desain tingkat puncak. Rekayasa informasi mencakup juga pengumpulan kebutuhan pada tingkat bsisnis strategis dan tingkat area bisnis.[9]

2. **Analisa Kebutuhan Perangkat Lunak (*Software Requirements Analysis*)**

   Proses pengumpulan kebutuhan diintensifkan dan difokuskan, khususnya pada perangkat lunak. Untuk memehami sifat program yang dibangun, perekayasa perangkat lunak (analis) harus memahami domain informasi, tingkah laku, unjuk kerja, dan antar muka (*interface*) yang diperlukan. Kebutuhan baik untuk sistem maupun perangkat lunak didokumentasikan dan dilihat lagi dengan pelanggan.[9]

3. **Desain (*Design*)**

   Desain perangkat lunak sebenarnya adalah proses multi langkah yang berfokus pada empat atribut sebuah program yang berbeda :

   Struktur data, arsitektur perangkat lunak, representasi interface, dan detail (algoritma) prosedural. Proses desain menerjemahkan syarat/ kebutuhan dalam sebuah representasi perangkat lunak yang dapat diperkirakan demi kualitas sebelum dimulai pemunculan kode. Sebagaimana persyaratan, desain didokumentasi dan menjadi bagian dari konfigurasi perangkat lunak.[9]

4. **Pengkodean (*Coding*)**

   Desain harus diterjemahkan dalam bentuk mesin yang bisa dibaca. Langkah pembuatan kode melakukan tugas ini. Jika desai dilakukan dengan cara yang lengkap, pembuatan kode dapat diselesaikan secara mekanis.[9]

5. **Pengujian (Testing)**

   Sesudah kode dibuat, pengujian program dimulai. Proses pengujian berfokus pada logika internal pernagkat lunak, memastikan bahwa semua pernyataan sudah diuji, dan pada eksternal fungsional -- yaitu mengarahkan pengujian untuk menemukan kesalhan-kesalahan dan memsatikan bahwa input yang dibatasi akan memberikan hasil actual yang sesuai dengan hasil yang dibutuhkan.[9]

6. **Pemeliharaan (Support)**

   Perangkat lunak akan mengalami perubahan setelah disampaikan kepada pelanggan/ user (pengecualian yang mungki adalah kepada perangkat lunak yang dilekatkan). Perubahan akan terjadi karena kesalahan-kesalahan ditentukan, karena perangkat lunak harus disesuaikan untuk mengakomodasi perubahan-perubahan didalam lingkungan eksternalnya (contohnya perubahan yang dibutuhkan sebagai akibat perangkat peripheral atau sistem operasi yang baru), atau karena pelanggan membutuhkan perkembangan fungsional atau unjuk kerja. Pemeliharaan perangkat lunak mengaplikasikan lagi setiap fase program sebelumnya dan tidak membuat yang baru lagi.[9]

Mengapa model ini sangat populer ? Selain karena pengaplikasian menggunakan model ini mudah, kelebihan dari model ini adalah ketika semua kebutuhan sistem dapat didefinisikan secara utuh, eksplisit, dan benar di awal project, maka SE dapat berjalan dengan baik dan tanpa masalah. Meskipun seringkali kebutuhan sistem tidak dapat didefinisikan seeksplisit yang diinginkan, tetapi paling tidak, problem pada kebutuhan sistem di awal project lebih ekonomis dalam hal uang (lebih murah), usaha, dan waktu yang terbuang lebih sedikit jika dibandingkan problem yang muncul pada tahap-tahap selanjutnya.

Meskipun demikian, karena model ini melakukan pendekatan secara urut / sequential, maka ketika suatu tahap terhambat, tahap selanjutnya tidak dapat dikerjakan dengan baik dan itu menjadi salah satu kekurangan dari model ini.

Selain itu, ada beberapa kekurangan pengaplikasian model ini, antara lain adalah sebagai berikut:

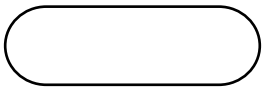
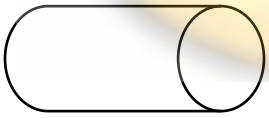
1. Ketika problem muncul, maka proses berhenti, karena tidak dapat menuju ke tahapan selanjutnya. Bahkan jika kemungkinan problem tersebut muncul akibat kesalahan dari tahapan sebelumnya, maka proses harus membenahi tahapan sebelumnya agar problem ini tidak muncul. Hal-hal seperti ini yang dapat membuang waktu pengerjaan SE.
2. Pendekatannya secara sequential, maka setiap tahap harus menunggu hasil dari tahap sebelumnya. Hal itu tentu membuang waktu yang cukup lama, artinya bagian lain tidak dapat mengerjakan hal lain selain hanya menunggu hasil dari tahap sebelumnya. Oleh karena itu, seringkali model ini berlangsung lama pengerjaannya.
3. Pada setiap tahap proses tentunya dipekerjakan sesuai spesialisasinya masing-masing. Oleh karena itu, ketika tahap tersebut sudah tidak dikerjakan, maka sumber dayanya juga tidak terpakai lagi. Oleh karena itu, seringkali pada model proses ini dibutuhkan seseorang yang "multi-skilled", sehingga minimal dapat membantu pengerjaan untuk tahapan berikutnya.

Masing-masing dari masalah tersebut bersifat rill, tetapi paradigma siklus kehidupan klasik memiliki tempat yang terbatas namun penting di dalam kerja rekayasa perangkat lunak. Paradigmaa itu memberikan *template* di mana metode analisis, desain, pengkodean, pengujian, dan pemeliharaan bisa dilakukan. Siklus kehidupan klasik tetap menjadi model bagi rekayasa perangkat lunak yang paling luas dipakai. Sekalipun memiliki kelemahan, secara signifikan dia lebih baik dari pada pendekatan yang sifatnya asal kepada pengembang perangkat lunak.

2.4. Flow Chart

Flowmap adalah penggambaran secara grafik dari langkah-langkah dan urutan-urutan prosedur dari satu program.[6]

Tabel II-1 Daftar Simbol-simbol dalam *Flowmap*

Simbol	Deskripsi
	Simbol yang digunakan untuk menunjukkan awal atau akhir dari suatu proses.
	Menunjukkan dokumen input dan output baik untuk proses manual mekanik atau computer.
	Menunjukkan pekerjaan manual
	Menunjukan pekerjaan yang terkomputerisasi
	Digunakan kalo melakukan pemilihan
	Digunakan buat database

2.5. Pemograman

2.5.1. Visual Studio 2010

Visual Basic diturunkan dari bahasa BASIC. Visual Basic terkenal sebagai bahasa pemograman yang mudah untuk digunakan terutama untuk membuat membuat aplikasi yang berjalan diatas *platform* Windows. [7]

Pada tahun 90an, Visual Basic menjadi bahasa pemograman yang paling popular dan menjadi pilihan utama untuk mengembangkan program berbasis

Windows. Versi Visual Basic terakhir sebelum berjalan di atas .NET Framework adalah VB6 (Visual Studio 1998).[7]

Visual Basic .NET dirilis pada bulan Februari tahun 2002 bersamaan dengan *platform* .NET Framwork 1.0. Kini sudah ada beberapa versi dari Visual Basic yang berjalan pada platform .NET, yaitu VB 2002 (VB7), VB 2005 (VB8), VB 2008 (VB9), dan VB 2010 (VB10) yang rilis bersamaan dengan Visual Studio 2010.[7]

Selain Visual Basic 2010, Visual Studio 2010 juga mendukung beberapa bahsa lain, yaitu C#, C++, F# (bahasa baru untuk *functional programming*), IronPhyton, dan IronRuby (bahasa baru untuk *dynamic programming*).[7]

2.5.2. Microsoft Access 2013

Microsoft merilis Microsoft Access 1.0 pada bulan November 1992 dan dilanjutukan dengan merilis versi 2.0 pada tahun 1993. Microsoft menentukan spesifikasi minimum untuk menjalankan Microsoft Access 2.0 adalah sebuah komputer dengan sistem operasi Microsoft Windows 3.0.

Perangkat lunak tersebut bekerja dengan sangat baik pada sebuah basis data dengan bayak *record* tapi terdapat beberapa kasus dimana data mengalami kerusakan. Sebagai contoh, pada ukuran basis data melebihi 700 *megabyte* sering mengalami masalah seperti ini (pada saat itu, memang hard disk yang beredar masih berada dibawah 700 *megabyte*).

Buku manual yang dibawanya memperingatkan bahwa beberapa kasus tersebut disebabkan driver perangkat yang kuno atau konfigurasi yang tidak benar. Nama kode (*codename*) yang digunakan oleh Access pertama kali adalah Cirrus yang dikembangkan sebelum Microsoft mengembangkan Microsoft Visual Basic, sementara mesin pembuat form antarmuka yang digunakan dengan Rubby.

Bill Gates melihat purwarupa (*prototype*) tersebut dan memutuskan bahwa komponen bahasa pemograman BASIC harus dikembangkan secara bersama-sama sebagai sebuah aplikasi terpisah tapi dapat diperluas. Proyek ini dinamakan dengan Thunder.

Kedua proyek tersebut dikembangkan secara terpisah dan mesin pembuat form yang digunakan oleh keduanya tidak saling cocok satu sama lainnya. Hal tersebut berakhir saat Microsoft merilis Visual Basic for Aplication (VBA).

Microsoft Acces (Microsoft Office Acces) adalah sebuah program aplikasi basis data computer relasional yang ditunjukan untuk kalangan rumahan dan perusahaan kecil hingga menengah.

Aplikasi ini merupakan anggota dari beberapa aplikasi Microsoft Office, selain tentunya Microsoft Word, Microsft Excel, dan Microsoft PowerPoint. Aplikasi ini menggunakan mesin basis data Microsoft Jet Database Engine, dan juga menggunakan tampilan grafis yang intuitif sehingga memudahkan pengguna.

Microsoft Access dapat menggunakan data yang disimpan di dalam format Microsoft Access, Microsoft Jet Database Engine, Microsoft SQL Server, Oracle Database, atau semua container basis data yang mendukung standar ODBC.

Para pengguna / programmer yang mahir dapat menggunakannya untuk mengembangkan perangkat lunak aplikasi yang kompleks, sementara para programmer yang kurang mahir dapat menggunakan untuk mengembangkan perangkat lunak aplikasi yang sederhana.

Access juga mendukung teknik-teknik pemograman berorientasi objek tetapi tidak dapat digolongkan kedalam perangkat bantu pemograman berorientasi objek.[8]