BAB 2

LANDASAN TEORI

2.1 Sistem Informasi Data

Data adalah fakta-fakta mentah atau deskripsi dasar dari konsep-konsep, kejadian-kejadian, kegiatan-kegiatan, dan transaksi yang dapat ditangkap, direkam, disimpan, dan dikelompokkan, tetapi tidak terorganisasi dalam membawa arti tertentu (Turban et al, 2003, p15). Jadi data merupakan bentuk dasar yang perlu diolah lebih lanjut untuk membentuk sebuah informasi.

Informasi adalah kumpulan fakta-fakta (data) yang sudah terorganisasi dalam suatu cara sehingga dapat berarti bagi penerima (Turban et al, 2003, p15). Dengan demikian informasi merupakan bagian penting dalam pengambilan keputusan.

Sistem adalah kumpulan komponen atau elemen yang saling berhubungan (berinteraksi) yang ditampilkan sebagai salah satu kesatuan dan dirancang untuk mencapai tujuan tertentu (Britton et al, 2002, p2). Berdasarkan pengertian tersebut, maka sistem harus dirancang sedemikian rupa agar dapat bekerja secara baik sehingga tujuan dari dibentuknya sistem dapat dicapai.

2.2 *Database*

2.2.1 Sistem Tradisional Berbasiskan *File*

Sistem berbasiskan *file* adalah sekumpulan program aplikasi yang memberikan pelayanan bagi para *end-users* seperti pembuatan laporan. Setiap program akan mendefinisikan dan mengatur datanya masing-masing (Connolly dan Begg, 2002, p7). Sistem berbasiskan *file* dapat berjalan baik pada aplikasi

yang memiliki data yang tidak banyak. Sistem ini merupakan awal dari proses komputerisasi sistem penyimpanan (*filing*) secara manual. Sistem ini juga berjalan baik pada jumlah data yang banyak selama aplikasi dipakai hanya terbatas pada menyimpan dan mengambil data tersebut. Akan tetapi, sistem berbasiskan *file* akan mengalami kesulitan bila harus dilakukan manipulasi data pada jumlah yang banyak.

Berikut adalah keterbatasan penggunaan pendekatan sistem berbasiskan *file*, yaitu :

1. Pemisahan dan isolasi data

   Untuk mendapatkan suatu informasi, terkadang *user* perlu mengambil data dari satu *file* yang kemudian data tersebut digunakan untuk mendapatkan data yang berbeda pada *file* yang lain. Hal ini menyebabkan kesulitan dalam pengaksesan data.

2. Duplikasi data

   Duplikasi data dapat menyebabkan :

   a. Memerlukan tempat penyimpanan yang besar.
   b. Memakan lebih banyak waktu dan uang untuk menggunakan data lebih dari sekali.
   c. Duplikasi data menghilangkan integritas data sehingga konsistensi data menjadi tidak terjamin.

3. Ketergantungan data

   Karena struktur dan penyimpanan fisik *file* data didefinisikan dalam kode aplikasi maka perubahan pada struktur yang sudah ada menjadi sulit dilakukan.

4. *Query* permanen dari program aplikasi

   Sistem berbasiskan *file* sangat bergantung pada pengembang aplikasi yang sudah menulis *query* yang dibutuhkan. *Query* yang dibuat bersifat permanen dan tidak terdapat fasilitas untuk *query* yang tidak direncanakan.

5. Perbedaan format *file* yang disimpan

   Karena setiap bagian dari sistem mengatur datanya sendiri. Ada kemungkinan data yang disimpan memiliki perbedaan format dalam penyimpanan. Ketika suatu bagian ingin mengakses data pada bagian lain. Data tersebut harus disesuaikan terlebih dahulu formatnya agar dapat diproses lebih lanjut. Hal ini akan menghabiskan waktu yang lama dan biaya yang mahal.

2.2.2 Pengertian *database*

Database adalah koleksi data yang saling berhubungan dan didesain sedemikian rupa untuk memenuhi kebutuhan informasi dari suatu organisasi (Connolly dan Begg, 2002, p14). Sehingga *database* memiliki peran yang penting dalam penyediaan informasi untuk sebuah sistem.

Relational database adalah sebuah kumpulan dari relasi yang telah dinormalisasi dengan nama relasi yang jelas (Connolly dan Begg, 2002, p74). *Relational database* merupakan suatu tipe *database* yang berdasarkan model *relational*, dimana semua data dapat dilihat oleh pengguna, disusun dalam bentuk tabel-tabel dan semua operasi pada *database* berkerja pada tabel-tabel tersebut. Relasi antar-tabel pada *relational database* sudah melalui tahap normalisasi dengan nama relasi yang berbeda-beda.

Ada 3 jenis relasi antar-*records* dalam tabel (Connolly dan Begg, 2002, p344), yaitu :

1. Relasi *one-to-one* adalah relasi antara satu *record* dengan satu *record* dalam tabel lain yang saling berhubungan.
2. Relasi *one-to-many* adalah relasi antara satu *record* dengan lebih dari satu *record* dalam tabel lain sehingga saling berhubungan.
3. Relasi *many-to-many* adalah relasi antara banyak *record* dengan lebih dari satu *record* dalam tabel lain yang saling berhubungan.

2.2.3 *Database Management System* (DBMS)

DBMS (*Database Management System*) adalah sebuah sistem *software* yang memungkinkan pengguna untuk mendefinisikan, membuat, memelihara, dan mengatur akses ke dalam *database* (Connolly dan Begg, 2002, p16). DBMS merupakan sebuah *software* yang berinteraksi dengan pengguna program aplikasi dan *database*.

Sebuah DBMS menyediakan beberapa fasilitas berikut :

1. Data *Definition Language* (DDL)

   DDL adalah sebuah bahasa yang mengijinkan *Database Administrator* atau pengguna untuk menggambarkan dan memberi nama dari *entities, attribute,* dan *relationships* yang dibutuhkan untuk aplikasi bersama dengan semua kepercayaan yang berhubungan dan batasan keamanan (Connolly dan Begg, 2002, p40).

2. Data *Manipulation Language* (DML)

DML adalah sebuah bahasa yang menyediakan sekumpulan operasi yang mendukung operasi manipulasi data di dalam *database* (Connolly dan Begg, 2002, p41).

3. Menyediakan kontrol akses ke dalam *database*, sebagai contoh :
   a. *Security system*, dimana mencegah pengguna yang tidak mempunyai hak untuk mengakses *database*.
   b. *Integrity system*, dimana menjaga konsistensi dari data.
   c. *Concurrency control system*, dimana mengijinkan akses yang terbagi dalam *database*.
   d. *Recovery control system*, dimana mengembalikan kondisi *database* sebelum kegagalan *hardware* atau *software*.
   e. *User-accessible catalog*, dimana berisi deskripsi dari data dalam *database*.

SQL adalah suatu bahasa yang dirancang untuk sistem operasi pengaksesan data pada struktur *relational database* yang mentransformasikan *input* menjadi *output* yang diinginkan pengguna (Connolly dan Begg, 2002, p111). Operasi pengaksesan data meliputi penyisipan data (*insert*), pengubahan data (*update*), pengambilan data (*select*), dan penghapusan data (*delete*). Perintah-perintah di atas dilakukan atas permintaan dari pengguna.

2.3 *System Development Life Cycle* (SDLC)

Sebelumnya pengembangan *software* dilakukan oleh *programmer* dengan cara menuliskan *code* untuk menyelesaikan suatu masalah. Namun sekarang sistem sangat besar dan kompleks dan dibagi berdasarkan beberapa bagian

seperti perancangan, analisis, *programmer*, pengetesan dan pengguna yang harus bekerja bersama-sama untuk menciptakan jutaan baris *code* agar perusahaan dapat berjalan dengan baik (Kay, R.). Untuk mengatur semua bagian yang ada maka dibuatlah *System Development Life Cycle* (SDLC).

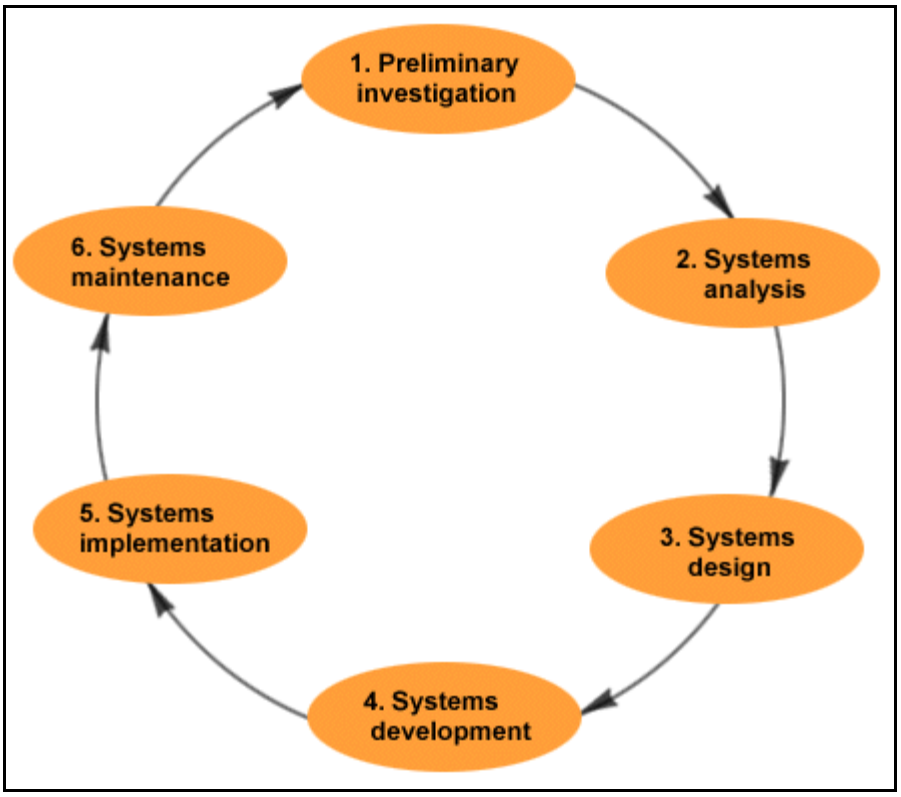
SDLC adalah tahapan-tahapan pekerjaan yang dilakukan oleh analis sistem dan *programmer* dalam membangun sebuah sistem informasi. SDLC juga sangat berguna untuk merencanakan, memutuskan dan mengontrol proses pengembangan sistem informasi. Langkah-langkah yang digunakan meliputi :

1. Melakukan survei dan menilai kelayakan proyek pengembangan sistem informasi.
2. Mempelajari dan menganalisis sistem informasi yang sedang berjalan.
3. Menentukan permintaan pemakai sistem informasi.
4. Memilih solusi atau pemecahan masalah yang paling baik.
5. Menentukan perangkat keras (*hardware*) dan perangkat lunak (*software*) yang digunakan.
6. Merancang sistem informasi baru.
7. Membangun sistem informasi baru.
8. Mengkomunikasikan dan mengimplementasikan sistem informasi baru.
9. Memelihara dan melakukan perbaikan/peningkatan sistem informasi baru bila diperlukan.

Dengan siklus SDLC, proses membangun sistem dibagi menjadi beberapa langkah dan pada sistem yang besar, masing-masing langkah dikerjakan oleh tim yang berbeda.

Dalam sebuah siklus SDLC, terdapat enam langkah. Jumlah langkah SDLC pada referensi lain mungkin berbeda, namun secara umum adalah sama. Langkah-langkah tersebut adalah (Anonim1) :

1. Analisis sistem, yaitu membuat analisis aliran kerja manajemen yang sedang berjalan.
2. Spesifikasi kebutuhan sistem, yaitu melakukan perincian mengenai apa saja yang dibutuhkan dalam pengembangan sistem dan membuat perencanaan yang berkaitan dengan proyek sistem.
3. Perancangan sistem, yaitu membuat desain aliran kerja manajemen dan desain pemrograman yang diperlukan untuk pengembangan sistem informasi.
4. Pengembangan sistem, yaitu tahap pengembangan sistem informasi dengan menulis program yang diperlukan.
5. Pengujian sistem, yaitu melakukan pengujian terhadap sistem yang telah dibuat.
6. Implementasi dan pemeliharaan sistem, yaitu menerapkan dan memelihara sistem yang telah dibuat.



Gambar 2.1 Siklus SDLC (Anonim1)

Siklus SDLC dijalankan secara berurutan, mulai dari langkah pertama hingga langkah keenam. Setiap langkah yang telah selesai harus dikaji ulang, kadang-kadang pengkajian ulang dilakukan bersama *expert user*, terutama dalam langkah spesifikasi kebutuhan dan perancangan sistem untuk memastikan bahwa langkah telah dikerjakan dengan benar dan sesuai harapan. Jika tidak maka langkah tersebut perlu diulangi lagi atau kembali ke langkah sebelumnya.

Pengkajian ulang yang dimaksud adalah pengujian yang sifatnya *quality control*, sedangkan pengujian di langkah kelima bersifat *quality assurance*. *Quality control* dilakukan oleh orang dalam tim untuk membangun kualitas, sedangkan *quality assurance* dilakukan oleh orang di luar tim untuk menguji

kualitas sistem. Semua langkah dalam siklus harus terdokumentasi. Dokumentasi yang baik akan mempermudah pemeliharaan dan peningkatan fungsi sistem.

Berikut ini merupakan kelebihan dan kekurangan dari SDLC :

Tabel 2.1 Kelebihan dan Kekurangan SDLC

Kelebihan	Kekurangan
Adanya pengaturan pada system.	Waktu untuk membangun sistem menjadi lebih besar.
Memonitoring projek-projek besar.	Biaya pembangunan system meningkat.
Memiliki langkah-langkah yang detail	Sistem harus didefinisikan diawal.
Biaya lebih terarah dan target dapat dicapai dengan baik.	Sistem menjadi kaku.
Memiliki dokumentasi	Sulit dalam memperkirakan biaya, proyek menjadi terlalu besar.
Memberikan definisi *input* kepada pengguna dengan baik.	*Input* dari pengguna terkadang terbatas.
Mudah dalam pemeliharaan.	
Menggunakan standarisasi pengembangan dan desain.	

Ada beberapa model SDLC yang telah dibuat yaitu :

1. *Waterfall*
2. *Fountain*
3. *Spiral*
4. *Build and fix*
5. *Rapid prototyping*
6. *Incremental*

Pada skripsi ini kami menggunakan *waterfall* model. Model ini merupakan model dasar yang sangat baik untuk hampir semua pengembangan system dan juga merupakan model yang paling tua. Model ini merupakan

rangkaian dari kegiatan dimana hasil dari satu kegiatan menjadi input bagi kegiatan selanjutnya. Sehingga tidak ada titik balik pada model ini, setiap kegiatan yang dilakukan akan terus berurutan dari awal hingga akhir.

Setiap kegiatan karakteristik yang berbeda, berikut ini merupakan kegiatan-kegiatan yang dilakukan :

1. ***Software Requirements Analysis***.

   Proses pencarian kebutuhan difokuskan pada *software*. Untuk mengetahui sifat dari program yang akan dibuat, maka para *software engineer* harus mengerti tentang *domain* informasi dari software, misalnya fungsi yang dibutuhkan, *user interface* dan lain-lain. Dari dua aktivitas tersebut (pencarian kebutuhan sistem dan *software*) harus didokumentasikan dan ditunjukkan kepada pelanggan.

2. ***Design***.

   Proses ini digunakan untuk mengubah kebutuhan-kebutuhan diatas menjadi representasi ke dalam bentuk "*blueprint*" *software* sebelum pembuatan program dimulai. Desain harus dapat mengimplementasikan kebutuhan yang telah disebutkan pada tahap sebelumnya. Seperti dua aktivitas sebelumnya, maka proses ini juga harus didokumentasikan sebagai konfigurasi dari *software*.

3. ***Coding***.

   Untuk dapat dimengerti oleh mesin, dalam hal ini adalah komputer, maka desain tadi harus diubah bentuknya menjadi bentuk yang dapat dimengerti oleh mesin, yaitu ke dalam bahasa pemrograman melalui proses *coding*.

Tahap ini merupakan implementasi dari tahap desain yang secara teknis nantinya dikerjakan oleh *programmer*.

4. ***Testing / Verification***.

   Sesuatu yang dibuat haruslah diujicobakan. Demikian juga dengan *software*. Semua fungsi-fungsi *software* harus diujicobakan, agar *software* bebas dari *error*, dan hasilnya harus benar-benar sesuai dengan kebutuhan yang sudah didefinisikan sebelumnya.

5. ***Maintenance***.

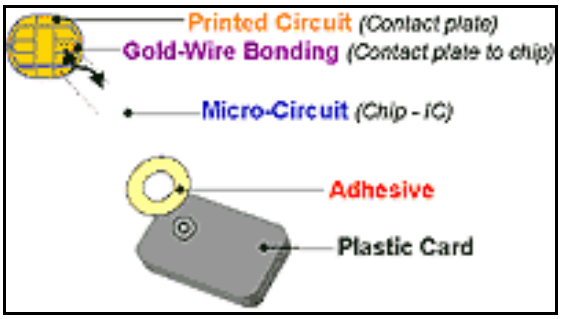
   Pemeliharaan suatu *software* diperlukan, termasuk di dalamnya adalah pengembangan, karena *software* yang dibuat tidak selamanya hanya seperti itu. Ketika dijalankan mungkin saja masih ada *errors* kecil yang tidak ditemukan sebelumnya, atau ada penambahan fitur-fitur yang belum ada pada *software* tersebut. Pengembangan diperlukan ketika adanya perubahan dari eksternal perusahaan seperti ketika ada pergantian sistem operasi, atau perangkat lainnya.

2.4 Teknologi *Smart Card* dan *Java Card*

2.4.1 Pengenalan *Smart Card*

Smart card adalah sebuah kartu plastik yang digabung dengan *Integrated Circuit* (IC) yang menyerupai ukuran kartu kredit dan bersifat *tamper resistant*, yaitu usaha ilegal pengambilan data dari dalam kartu tidak dimungkinkan (Ortiz, C. E.). *Smart card* didesain untuk menyimpan data yang bersifat pribadi dengan tingkat keamanan yang tinggi dan kartu mudah untuk dibawa kemana saja (*portable*). Penyimpanan dan pemrosesan informasi dalam *smart card* dilakukan

melalui sirkuit elektronik yang digabungkan dalam silikon pada bahan plastik (umumnya PVC atau ABS) dari kartu.



Gambar 2.2 Komponen *Smart Card* (Bezakova et al)

Integrated Circuit (*chip*) berukuran sangat kecil dan *printed circuit* berbentuk plat emas yang tipis. *Printed circuit* ini memberikan kontak elektrik dengan lingkungan luar dan juga melindungi chip dari kerusakan mekanik dan gangguan elektrik (Bezakova et al). Kartu dapat ditanamkan hologram untuk menghindari pemalsuan.

Untuk pemrosesan dan penyimpanan data yang aman digunakan *public-key* atau *shared-key algorithm*. Beberapa *smart card* memiliki *cryptographic coprocessors* yang terpisah yang mendukung algoritma seperti RSA(*Rivest-Shamir-Adleman's algorithm*), AES (*Advanced Encryption Standard*) dan *Triple* DES (*Triple Data Encryption Standard*).

Cara komunikasi yang dipakai *smart card* adalah *half duplex*, yaitu jenis komunikasi dua arah, namun tidak dilakukan secara bersamaan. Fungsi mengirim dan menerima harus dilakukan secara bergantian. Data yang dikirimkan dan diterima dari *smart* card disimpan dalam *buffer* yang terdapat dalam RAM *smart card*.

2.4.2 Standar Internasional *Smart Card*

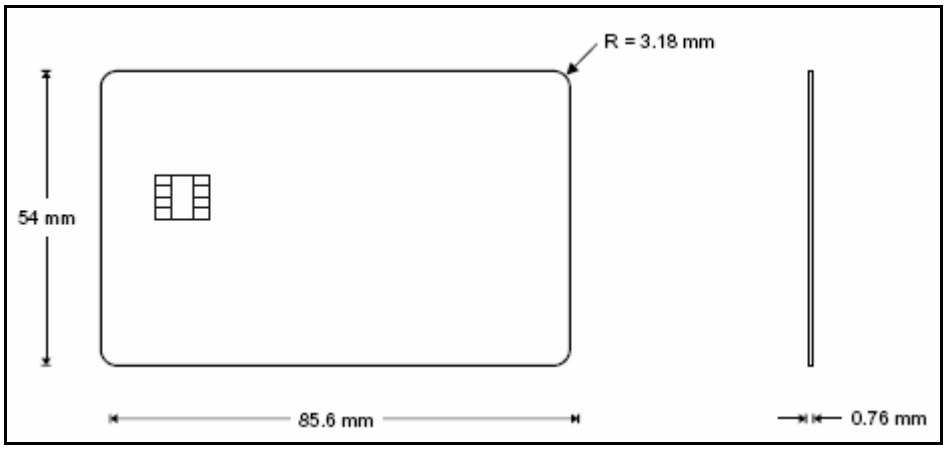
Smart card merupakan sebuah komponen dalam sebuah sistem yang kompleks. *Interface* yang berhubungan dengan kartu dalam sebuah sistem harus memiliki kesesuaian dan spesifikasi. *Smart card* memungkinkan pemecahan masalah dalam sebuah sistem dengan mengabaikan sistem lainnya. Ini berarti, akan diperlukan *smart card* yang berbeda-beda untuk setiap sistem. Oleh karena itu, diperlukan standarisasi untuk memungkinkan sebuah kartu multifungsional dapat dikembangkan (Rankl dan Effing, 2003, p9).

Standar ISO/IEC digunakan untuk menentukan standarisasi dari *smart card*. Kepanjangan dari ISO adalah *International Organization for Standarization*, sedangkan kepanjangan IEC adalah *International Electrotechnical Commission*.

ISO/IEC 7816 dan ISO/IEC 7810 adalah standar yang menentukan hal-hal berikut ini :

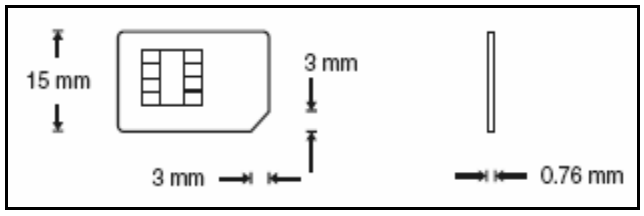
1. Bentuk fisik kartu.
2. Posisi dan ukuran dari konektor elektrikal kartu.
3. Karakteristik dari sirkuit elektonik.
4. Protokol komunikasi, termasuk format perintah yang dikirim ke kartu dan tanggapan dari kartu.
5. Ketahanan kartu.
6. Fungsionalitas kartu.

Ukuran kartu *smart card* dengan format ID-1 berdasarkan ISO/IEC 7810 adalah 85,60mm x 53,98mm dan memiliki ketebalan kartu 0,76mm ± 0,08mm serta jari-jari sudutnya 3,18mm (Rankl dan Effing, 2003, p28-29).



Gambar 2.3 Ukuran standar Smart Card format ID-1(Rankl dan Effing, 2003, p29)

Ukuran lainnya adalah format ID-000 (berdasarkan sistem telepon *mobile* berbasis GSM) yaitu 25mm x 15mm dengan ketebalan 0,76mm ± 0,08mm. Bagian kanan bawah kartu dipotong dengan sudut sebesar 45°. Jari-jari sudut kartu adalah 1mm ± 0,10mm dan panjang sisi sudut yang dipotong sebesar 3mm ± 0,03mm(Rankl dan Effing, 2003, p29-30).



Gambar 2.4 Ukuran standar *Smart Card* format ID-000 (Rankl dan Effing, 2003, p30)

2.4.3 Tipe *Smart Card*

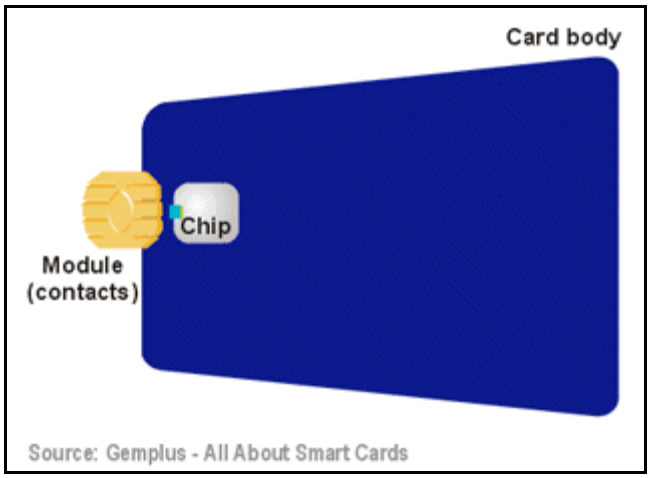
Secara umum ada dua tipe dari *smart card* yaitu *Memory card* dan *Microprocessor card* (Anonim2). *Memory card* hanya menyimpan dan memproteksi data secara lokal, namun tidak mengandung sebuah prosesor untuk melakukan perhitungan komputer pada data. Sedangkan *microprocessor card* memiliki *memory* dan *microprocessor* yang dapat melakukan kalkulasi pada data dan menyimpan data dalam kartu secara aman.

Biasanya, *memory card* dapat menyimpan data sebesar 1K sampai 4K dan keuntungan dari *memory card* terletak pada teknologinya yang sederhana. Sedangkan *microprocessor card*, menawarkan sekuritas yang ditingkatkan dan kemampuan multifungsional. Data yang disimpan dalam *microprocessor card* tidak dapat diakses secara langsung melalui aplikasi di luar kartu. *Microprocessor* mengontrol data dan *memory* mengatur pengaksesan data menurut kondisi yang diberikan seperti *password*, enkripsi dan lainnya. Keuntungan *microprocessor card* adalah dapat diintegrasikan ke lebih dari satu aplikasi.

Secara fisik, *smart card* dibedakan menjadi dua jenis yaitu :

1. *Contact Smart Card*

   Contact Smart Card bekerja dengan cara berkomunikasi secara fisik antara *card reader* dan *smart card's pin contact* yang berbentuk segiempat berukuran ±1 cm. *Contact smart card* tidak membutuhkan baterai dan akan aktif ketika terhubung dengan *card reader*. Saat terhubung dengan *reader*, maka *chip* menunggu perintah *request* dari *client/host* dari aplikasi untuk membaca informasi dari *chip* atau menulis informasi ke *chip*.



Gambar 2.5 *Contact Smart Card* (Ortiz, C. E.)

Gambar 2.6 *Smart Card's pin contact*(Anonim2)

Berikut adalah fungsi dari masing-masing *contact* menurut ISO/IEC 7816, yaitu :

a. VCC (*Supply Voltage*)

   Sebagai penyedia tegangan listrik, biasanya sebesar 3 atau 5 volt dengan toleransi maksimum ±10%.

b. RST (*Reset Input*)

   Berfungsi mengirimkan sinyal untuk me-*reset microprocessor*.

c. CLK (*Clock Input)*

   Berfungsi sebagai *timing* atau *clocking signal* yang mengatur frekuensi waktu atau kecepatan mikroprosessor.

d. GND (*Ground*)

Merupakan *reference voltage* yang berlawanan dengan Vcc dimana energi potensialnya diukur. Nilainya dianggap 0 volt.

e. VPP (*Programming Voltage*)

Berfungsi menyediakan tegangan listrik yang berbeda dari VCC. C6 biasa digunakan untuk aplikasi lain seperti konektivitas USB.

f. I/O (*Input*/*Output*)

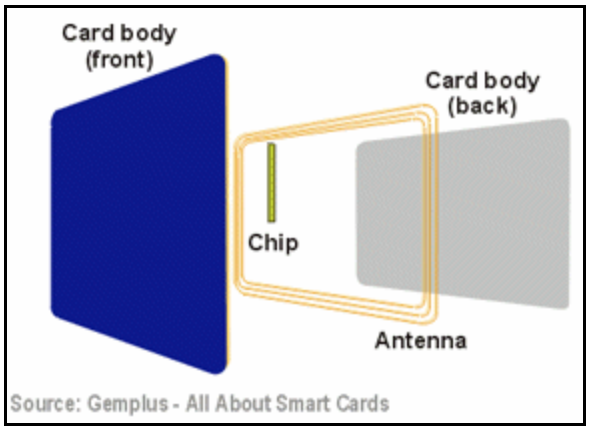
Berfungsi memonitor semua komunikasi yang keluar maupun yang masuk pada kartu.

g. C4 dan C8

Contact C4 dan C8 merupakan *contact* tambahan yang digunakan bila diperlukan untuk operasi pada kartu atau penggunaan *interface* di masa akan datang, seperti USB. Kedua contact ini dapat dihilangkan untuk mengurangi biaya produksi. Oleh karena itu, *Smart card* ada yang memiliki 6 atau 8 *pin contact.*

2. *Contactless Smart Card*

Contactless Smart Card berkomunikasi dengan terminal melalui sinyal frekuensi radio. Komunikasi *contactless smart card* berdasarkan dengan teknologi yang sama dengan *Radio Frequency Identification* (RFID). *Contactless smart card* tidak memiliki baterai, sehingga kartu ini memiliki induktor yang menangkap sinyal frekuensi radio sebagai sumber daya elektronik kartu. *Contactless card* memerlukan jarak tertentu untuk melakukan pertukaran data dengan *card reader*.



Gambar 2.7 *Contactless Smart Card* (Ortiz, C. E.)

Standar komunikasi *contactless smart card* adalah ISO/IEC 14443. Berikut adalah standarisasi ISO/IEC *Contactless Smart Card* :

Tabel 2.2 Standarisasi ISO/IEC *Contactless Smart Card* (Rankl dan Effing, 2003, p101)

Standar	Tipe *contactless smart card*	Jarak komunikasi
ISO/IEC 10 536	*Close-coupling card*	± 1 cm
ISO/IEC 14 443	*Proximity coupling card* (PICC)	± 10 cm
ISO/IEC 15 693	*Vicinity coupling card* (VICC)	± 1 m

Tabel 2.3 Perbandingan Teknologi *Smart Card* dengan RFID (Anonim3)

	RFID	***Smart Card***
Jarak Komunikasi	± 30 cm (pendek) ± 3 m (menengah) ± 10 m (jauh)	± 1 cm (pendek) ± 10 cm (menengah) ± 1 m (jauh)
Authentikasi (pembuktian keaslian)	Rendah, oleh karena itu mudah dipalsukan.	tinggi, karena adanya enkripsi sehingga memiliki tingkat keamanan yang tinggi.
Penyimpanan data	sedikit , ± 2KB	besar, ada yang mencapai 72 KB

2.4.4 Sistem Memori *Smart Card*

Smart card terdiri dari tiga buah jenis memory, yaitu (Chen, 2000, p16):

1. ROM (*Read-Only Memory)*

Digunakan untuk menyimpan program yang bersifat tetap dari kartu, dan tidak dibutuhkan tenaga untuk menyimpan data. ROM tidak dapat ditulis

kembali setelah kartu dibuat, dan berisikan sistem operasi. ROM dari *smart card* berisikan data dan aplikasi user yang bersifat permanen.

2. EEPROM (*Electrical Erasable Programable Read Only Memory).*

   Hampir serupa dengan ROM yang dapat menyediakan data ketika tenaga dari memori dimatikan. Perbedaannya adalah isi dari memori ini dapat dimodifikasi selama penggunaan kartu yang normal. EEPROM dapat menerima setidaknya 100.000 kali penulisan, dan dapat menyimpan data selama 10 tahun. Membaca dari EEPROM sama cepatnya dengan membaca dari RAM, namun menulis pada EEPROM 1000 kali lebih lama dibandingkan menulis dari RAM.

3. RAM (*Random Access Memory).*

   Digunakan sebagai tempat kerja sementara untuk menyimpan dan memodifikasi data. RAM merupakan memori yang bersifat *nonpersistent* yaitu informasi yang disimpan pada RAM akan dihilangkan ketika tenaga dari kartu hilang.

2.4.5 Protokol *Application Protocol Data Units* (APDU)

Seperti yang telah dispesifikasikan dalam ISO 7816-4, protokol APDU adalah sebuah protokol pada tingkatan aplikasi antara s*mart card* dengan *host-application*. Memiliki dua buah struktur, yaitu (Chen, 2000, p18-20) :

1. Pesan APDU yang dikirim pada kartu *smart card* oleh host application melalui *Card Acceptance Device* (CAD), yang dikenal sebagai *Command* APDU (C-APDU).

Mandatory header				Optional Body		
CLA	INS	P1	P2	Lc	Data Field	Le

Gambar 2.8 *Command* APDU

2. Pengiriman respon balik kepada *host-application* dari kartu, yang dikenal sebagai *Response* APDU (R-APDU).

Optional Body	Mandatory trailer	
Data Field	SW1	SW2

Gambar 2.9 *Response* APDU

2.4.6 Protokol *Transmission Protocol Data Units* (TPDU)

APDU dikirimkan melalui *transport protocol* yang didefinisikan pada ISO 7816-3. Struktur penukaran data antara *host* dan kartu menggunakan *transport protocol* yang disebut dengan *Transmission Protocol Data Units* (TPDU) (Chen, 2000, p20).

Dua buah *transport protocol* utama yang digunakan adalah protokol T=0 dan protokol T=1. Protokol T=0 bersifat *byte oriented* dimana unit terkecil yang diproses dan dikirimkan oleh protokol berbentuk sebuah byte. Sedangkan protokol T=1 merupakan protokol yang bersifat *block oriented*, dimana yang dikirimkan adalah serangkaian *byte*.

2.4.7 Teknologi *Java Card*

Java Card adalah teknologi yang memungkinkan aplikasi berbasis *Java* (*Applet*) dapat berjalan secara aman pada *Smart Card* dan alat-alat sejenisnya (Anonim4). *Java Card* merupakan program terkecil yang ditargetkan oleh *Java*

untuk digabung ke dalam *embedded device. Java Card* dipergunakan secara luas pada kartu SIM (digunakan pada kartu GSM *handphone*) dan kartu ATM.

Teknologi *Java Card* dikembangkan untuk menyimpan informasi yang bersifat *personal* di dalam *Smart Card*. Keamanan terlihat dalam beberapa aspek, yaitu:

1. Enkapsulasi data

   Data disimpan ke dalam aplikasi *Java Card* dan dieksekusi di dalam lingkungan yang terisolasi (*Java Card VM*), terpisah dari sistem operasi dan *hardware*.

2. *Applet firewall*

   Aplikasi - aplikasi yang berbeda dipisahkan antara satu dan yang lain oleh *applet firewall* yang membatasi dan mengecek akses elemen data antara satu *applet* ke *applet* lainnya.

3. Kriptografi

   Mendukung algoritma enkripsi seperti DES, 3DES, AES, RSA. Dan mendukung kriptografi lainnya seperti *signing, key generation* and *key exchange.*

4. *Applet*

   Adalah sebuah aplikasi konfigurasi yang hanya memproses perintah yang masuk dan memberikan balasan dengan mengirimkan data atau status respon ke *interface device.*

Elemen dari aplikasi *Java Card* terdiri dari *back-end application and systems, host (off-card) application, interface device (card reader), applet* dalam kartu, data yang diperlukan dan *software* yang mendukung (Ortiz, C. E.). Semua

elemen ini bersama-sama menyusun sebuah aplikasi *end-to-end* yang aman. Berikut ini adalah arsitektur dari aplikasi *Java Card* :

1. *Back-End Application and Systems*

   Adalah aplikasi dan sistem yang tidak berinteraksi secara langsung dengan pengguna, tapi secara tidak langsung mendukung *host application*. *Back-end application* mendukung *applet Java* di dalam kartu, sebagai contoh memberikan konektivitas dengan sistem keamanan, memberikan pelayanan seperti informasi pembayaran elektronik yang disimpan dalam *database*.

2. *Reader-Side Host (off-card) Application*

   Adalah aplikasi yang berada pada *desktop* atau terminal seperti PC, telepon genggam, mesin ATM. *Host application* mengatur komunikasi antara pengguna, *applet Java Card*, dan *back end application*.

3. *Reader-Side Card Acceptance Device* (CAD)

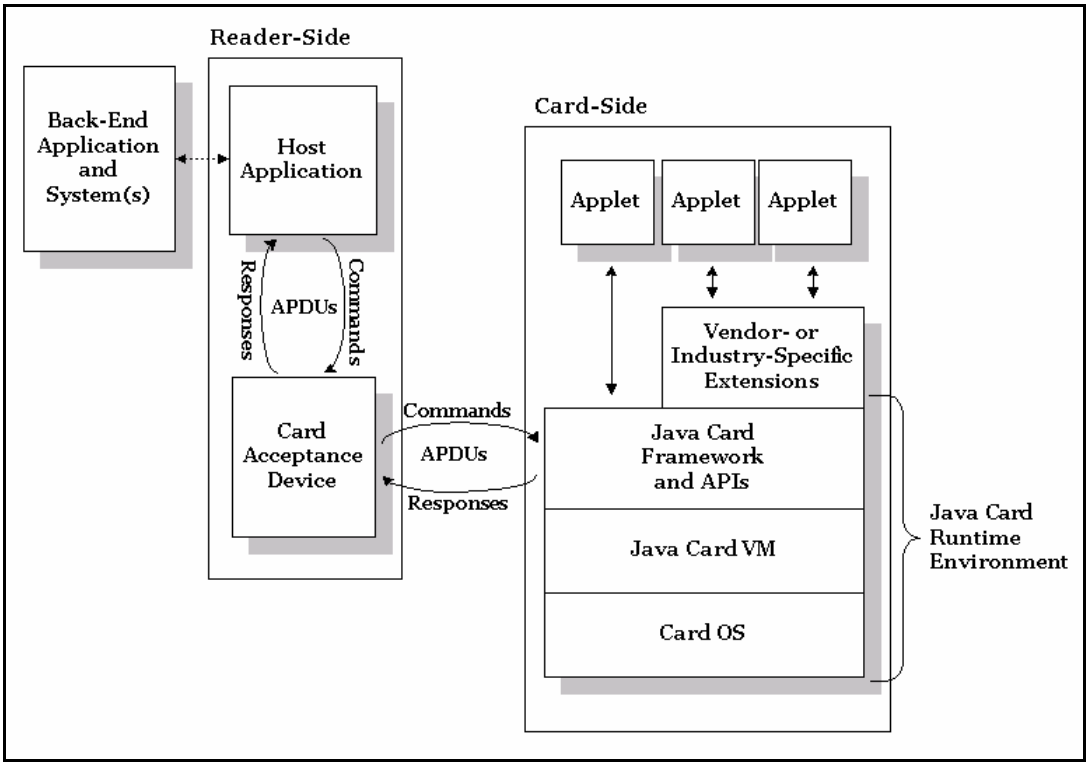
   Adalah alat yang menjadi jembatan antara *host application* dan *Java Card device*. CAD dapat berupa sebuah *card reader* yang dipasang pada sebuah komputer melalui *serial port* atau diintegrasikan kepada sebuah terminal seperti mesin pembayaran elektronik pada restoran atau tempat perbelanjaan.

   Alat ini meneruskan perintah APDU (*Application Protocol Data Unit*) dari *host application* ke kartu, dan meneruskan umpan balik dari kartu ke *host application*.

4. *Card-Side Applets* dan lingkungannya

   Java Card platform adalah lingkungan yang mendukung *multiple application*. Dalam hal ini, dalam satu kartu bisa memiliki satu atau lebih *Java Card applet* yang berjalan bersamaan dengan perangkat lunak yang

mendukungnya yaitu sistem operasi pada kartu dan *Java Card Runtime Environment* (JCRE). JCRE terdiri dari *Java Card VM* (*Virtual Machine*), *Java Card Framework and APIs*, dan beberapa pengembangan *APIs*.



Gambar 2.10 Arsitektur Aplikasi *Java Card* (Anonim4)

2.4.8 Keuntungan Teknologi *Java Card*

Keuntungan penggunaan *Java Card* adalah sebagai berikut (Chen, 2000, p8-9) :

1. Kemudahan dalam pengembangan aplikasi.

   Bahasa *Java* memudahkan para pengembang sehingga tidak memerlukan lagi melakukan pemrograman menggunakan bahasa mesin 6805 dan 8051. Selain itu, pengembang *smart card* mendapatkan keuntungan

dengan dukungan dari beberapa *vendor* seperti Borland, IBM, Microsoft, Sun, dan Symantec.

2. Keamanan.

   Keamanan adalah hal yang terpenting dalam *smart card*. Fitur keamanan yang dibangun pada *Java* berjalan dengan baik pada lingkungan *smart card*. Sebagai tambahan, *applet-applet* pada *Java Card platform* dipisahkan oleh *applet firewall*.

3. Tidak tergantung kepada perangkat keras.

   Teknologi *Java Card* dapat berjalan pada berbagai prosesor *smart card* (8 bit, 16 bit atau 32 bit). *Applet* yang siap digunakan dimasukkan ke dalam *Java smart card* tanpa memerlukan *compile* ulang.

4. Kemampuan menyimpan dan mengatur berbagai aplikasi.

   Sebuah *Java smart card* dapat menyimpan berbagai *applet*, seperti program pembayaran elektronik, authentikasi, pelayanan kesehatan dari berbagai macam *provider*. Karena *Java Card* memiliki mekanisme *applet firewall*, maka *applet* yang satu tidak dapat mengakses *applet* yang lain.

5. Sesuai dengan standarisasi *smart card* yang ada.

   Teknologi *Java Card* berdasarkan pada standar internasional ISO 7816, sehingga mendukung sistem dan aplikasi *smart card* yang sesuai dengan ISO 7816.

2.5 *Unified Modelling Language* (UML)

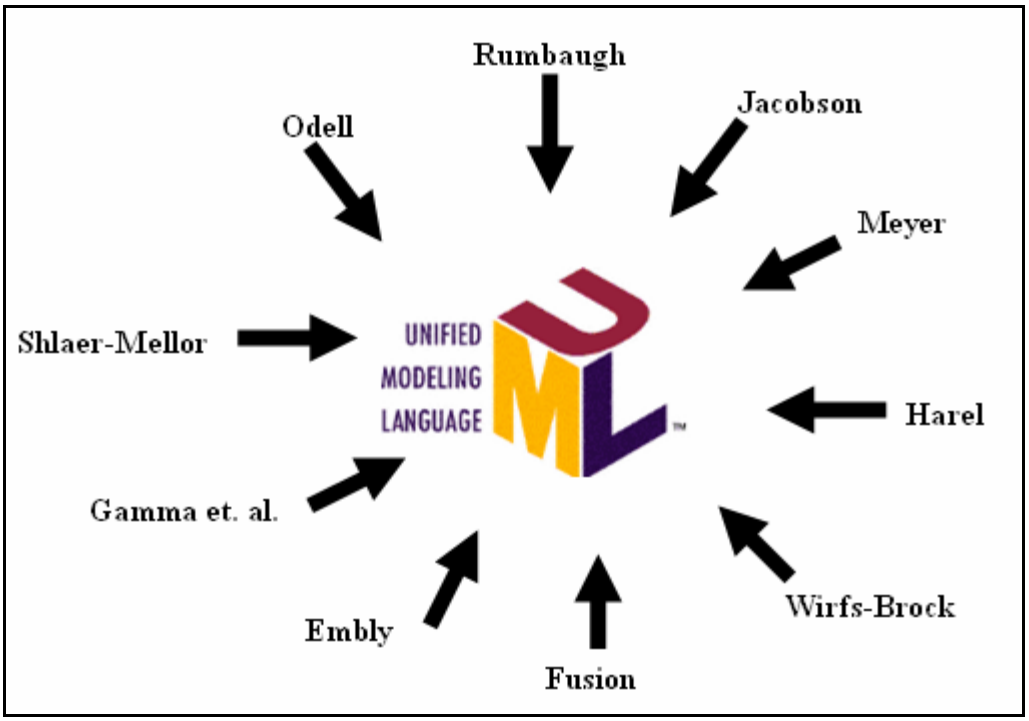
2.5.1 Sejarah UML

UML adalah sebuah bahasa yang telah menjadi standar dalam industri untuk menvisualisasi, menspesifikasi, merancang dan mendokumentasi sistem piranti lunak (Booch et al, 1999, p14). UML memberikan standar penulisan sebuah sistem *blue print*, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa program yang spesifik, skema *database*, dan komponen-komponen yang diperlukan dalam sistem *software*.

Pendekatan analisa dan rancangan dengan menggunakan model *Object Oriented* (OO) mulai diperkenalkan sekitar pertengahan 1970 hingga akhir 1980 dikarenakan pada saat itu aplikasi *software* sudah meningkat dan mulai kompleks. Jumlah yang menggunakaan metode OO mulai diujicobakan dan diaplikasikan antara 1989 hingga 1994, seperti halnya oleh Grady Booch dengan metode yang dikenal dengan OOSE (*Object-Oriented Software Engineering*), serta James Rumbaugh dari *General Electric*, dikenal dengan OMT (*Object Modelling Technique*).

Kelemahan saat itu disadari oleh Booch maupun Rumbaugh adalah tidak adanya standar penggunaan model yang berbasis OO, kemudian Booch, Rumbaugh dan Jacobson mulai mendiskusikan untuk mengadopsi masing-masing pendekatan metoda OO untuk membuat suatu model bahasa yang seragam yang disebut UML (*Unified Modeling Language*) dan dapat digunakan oleh seluruh dunia.

Secara resmi bahasa UML dimulai pada bulan Oktober 1994, ketika Rumbaugh bergabung dengan Booch untuk membuat sebuah *project* pendekatan metode yang seragam dari masing-masing metoda mereka.



Gambar 2.11 UML Menjadi Standar Bahasa Pemodelan (Anonim5)

2.5.2 Bagian UML

2.5.2.1 Class Diagram

Class diagram adalah diagram yang menunjukkan sekumpulan dari kelas-kelas, *interfaces*, dan kolaborasi-kolaborasi serta hubungannya (Booch et al, 1999, p107). *Class diagram* digunakan untuk memvisualisasikan, menspesifikasikan, mendokumentasikan model struktural dan juga membangun sistem yang dapat dieksekusi.

Pada *class diagram* terdapat simbol-simbol :

1. Simbol "+" untuk menandakan *public*.

2. Simbol "-" untuk menandakan *private.*
3. Simbol "#" untuk menandakan *protective.*

Class diagram direpresentasikan dalam bentuk kotak yang terbagi atas tiga bagian yaitu nama *class*, atribut, dan perilaku (*behavior*), seperti di bawah ini :

Class Name
Attribute: *Type=Initial Value*
Operation(*arg list*): *return type*

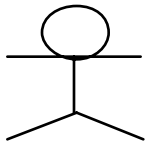
Gambar 2.12 Contoh *Class Diagram*

2.5.2.2 Use Case Diagram

Use case diagram menggambarkan sekumpulan *use case* dan aktor serta hubungannya (Booch et al, 1999, p234). *Use Case Diagram* memvisualisasikan tingkah laku dari suatu sistem dan menggambarkan interaksi antara aktor dengan sistem. Di bawah ini dijelaskan bagian *use case diagram*:

1. Aktor

   Sebuah aktor mewakili sekumpulan peranan yang saling berhubungan di dalam sistem dimana aktor tersebut berinteraksi dengan *use case* (Booch et al, 1999, p221). Aktor dapat berupa orang ataupun sistem yang otomatis berjalan. Notasi aktor dengan nama aktor tersebut dibawahnya:

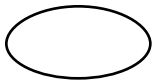


Aktor

2. *Use Case*

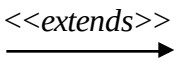
Sebuah *use case* menjelaskan sekumpulan dari *sequence,* dimana setiap *sequence* mewakili interaksi dari hal-hal di luar sistem (aktornya) dengan sistem itu sendiri (Booch et al, 1999, p220). Sehingga sebuah *use case* menunjukkan sebuah keperluan fungsional dari keseluruhan sistem.

Notasi *use case*:

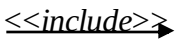


Untuk menghubungkan antara aktor dengan *use case* digunakan simbol garis yang disebut sebagai *relationship*.

Suatu *use case* dapat memiliki deskripsi teknik, yaitu: *extends*, dan *include.*



Extends berarti memperluas *use case* dasar dengan menambah *behavior-behavior* baru tanpa mengubah *use case* dasar itu sendiri. Titik di mana *use case* diperluas disebut sebagai *extension point.*



Sebuah *use case* dapat meng-*include* fungsionalitas dari *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Dengan adanya *use case diagram* maka akan membantu dalam menyusun kebutuhan sebuah sistem dan mengkomunikasikannya dengan klien.

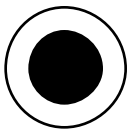
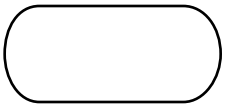
2.5.2.3 *Sequence Diagram*

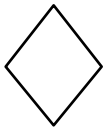
Sequence diagram menggambarkan sekumpulan objek dan interaksinya, termasuk pesan yang dikirim terhadap urutan waktu (Booch et al, 1999, p245). *Sequence diagram* menunjukkan sekumpulan objek dan pesan yang dikirim dan diterima oleh objek tersebut. *Sequence diagram* memiliki dua buah karakteristik yaitu :

1. Setiap objek memiliki *lifeline* yang digambarkan dengan garis putus-putus vertikal dan garis ini menunjukkan daur hidup dari sebuah objek.
2. Terdapat fokus kontrol yang digambarkan dengan sebuah persegi panjang yang tipis dan tinggi. Fokus kontrol ini menunjukkan periode waktu selama sebuah objek melakukan sebuah *event.*

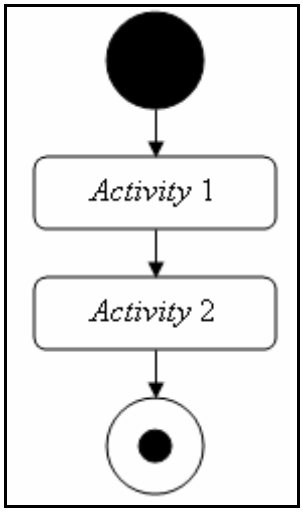
2.5.2.4 *Activity Diagram*

Activity diagram memodelkan aliran dari suatu aktivitas ke aktivitas berikutnya dalam suatu proses (Booch et al, 1999, p258). Komponen utama dalam *activity diagram* adalah:

	*Initial state*, yaitu menyatakan awal dimulainya suatu aktivitas.
	*Final state*, yaitu menyatakan berakhirnya suatu aktivitas.
	*State*, menggambarkan aktivitas yang merepresentasikan kinerja dari suatu operasi.

	*Control Flow*, menyatakan *relationship* diantara 2 *state*. *Control flow* mengidentifikansi kontrol yang dikirim dari *state* pertama ke *state* kedua setelah aktivitas pada *state* pertama selesai dijalankan.
	*Decision*, menggambarkan kontrol dari aliran yang bersifat kondisional.

Contoh penggunaan *Activity Diagram*:



Gambar 2. 13 Contoh *Activity Diagram*

Activity Diagram menekankan aliran kontrol dari suatu aktivitas ke aktivitas yang lain. Sehingga *activity diagram* dapat digunakan untuk menunjukkan aliran aktivitas sistem yang dirancang dari awal hingga aliran berakhir.

2.5.2.5 *Component Diagram*

Component Diagram menunjukkan organisasi dan hubungan ketergantungan antara satu set komponen dalam sebuah sistem (Booch et al, 1999, p393). Dengan *component diagram*, dapat digambarkan hubungan statis antara komponen-komponen fisik dan menspesifikasikan detailnya untuk membangun sebuah sistem.