BAB II

LANDASAN TEORI

2.1. Sekilas Tentang Aplikasi POS

Point Of Sales atau disingkat POS dapat diterjemahkan bebas menjadi sistem kasir, yaitu aktivitas yang berorientasi pada penjualan yang terjadi pada bidang usaha retail. Mengapa POS ini menjadi terlihat sangat penting? Hal ini semata-mata adalah karena POS merupakan terminal tempat uang diterima dari pelanggan ke toko retail. Bagi pemilik usaha, uang masuk adalah indikator yang paling mudah untuk mengukur pendapatan, disebut dengan omset.

POS juga menjadi penting karena seiring dengan berkembangnya usaha, sistem kasir akan dijalankan bukan oleh pemilik, namun oleh karyawan. Karena itu pemilik wajib tahu apa yang dikerjakan oleh kasir, dan berapa uang yang didapatkan secara tepat.

Para pengusaha retail lama mungkin telah terbiasa dengan mesin kasir elektronik atau disebut *Cash Register*. *Cash Register* adalah sistem kasir sederhana yang bisa mengetahui omset hari ini. Mesin ini juga bisa mengetahui aktivitas uang masuk oleh masing-masing karyawan kasir jika dipakai oleh lebih dari satu orang.

Seiring dengan pesatnya perkembangan komputer, *cash register* mulai ditinggalkan dengan alasan sebagai berikut:

1. Kapasitas penyimpanan yang terbatas.
2. Hanya dapat melakukan transaksi POS sederhana.

3. Tidak dapat digunakan untuk kepentingan lainnya.
4. Sistem POS terlalu sederhana. Kalaupun ada *cash register* yang mampu mendukung sistem stok (persediaan), harganya sangat mahal dibandingkan computer
5. Tidak dapat di *upgrade*. Misalnya, tidak mendukung sistem Barcode. Kalaupun ada, harganya juga mahal.
6. Biaya *maintenance* tinggi. Jika terjadi kerusakan modul, *sparepart* jarang tersedia dan harganya mahal.

Karena itu dewasa ini, sistem POS atau kasir telah digantikan oleh sistem komputer kasir. Walaupun tahap implementasi awal terlihat lebih rumit, namun fitur yang disediakan jauh melebihi *cash register*.

Saat ini *cash register* hanya cocok dipakai untuk *counter-counter* sederhana yang tidak mempunyai terlalu banyak barang. Misalnya depot di *food court, open space counter* di mall, atau usaha retail dengan frekuensi penjualan yang rendah. Satu-satunya keunggulan *cash register* adalah bentuknya yang *compact* dan mudah dipindah-pindah.

2.2. Sekilas Tentang Pertimbangan Investasi

Investasi adalah suatu istilah dengan beberapa pengertian yang berhubungan dengan keuangan dan ekonomi. Istilah tersebut berkaitan dengan akumulasi suatu bentuk aktiva dengan suatu harapan mendapatkan keuntungan dimasa depan. Terkadang, investasi disebut juga sebagai penanaman modal.

Berdasarkan teori ekonomi, investasi berarti pembelian (dan berarti juga produksi) dari kapital/modal barang-barang yang tidak dikonsumsi tetapi digunakan untuk produksi yang akan datang (barang produksi). Contohnya, membangun rel kereta api, pabrik, pembukaan lahan, atau seseorang sekolah di suatu universitas. Untuk lebih jelasnya, ROI (singkatan bahasa Inggris: *return on investment*) atau ROR (singkatan bahasa Inggris: *rate of return*) – dalam bahasa Indonesia disebut laba atas investasi – adalah rasio uang yang diperoleh atau hilang pada suatu investasi, relatif terhadap jumlah uang yang diinvestasikan. Jumlah uang yang diperoleh atau hilang tersebut dapat disebut bunga atau laba/rugi. Investasi uang dapat dirujuk sebagai aset, modal, pokok, basis biaya investasi. ROI biasanya dinyatakan dalam bentuk persentase dan bukan dalam nilai desimal.

ROI tidak memberikan indikasi berapa lamanya suatu investasi. Namun demikian, ROI sering dinyatakan dalam satuan tahunan atau disetahunkan dan sering juga dinyatakan untuk suatu tahun kalendar atau fiskal.

ROI digunakan untuk membandingkan laba atas investasi antara investasi-investasi yang sulit dibandingkan dengan menggunakan nilai moneter. Sebagai contoh, suatu investasi senilai 1000 rupiah yang menghasilkan bunga 50 rupiah jelas memberikan lebih banyak uang daripada investasi senilai 100 rupiah yang memberikan bunga 20 rupiah. Tapi investasi 100 rupiah memberikan ROI yang lebih besar.

Fungsi investasi pada aspek tersebut dibagi pada investasi non-residential (seperti pabrik, mesin, dll) dan investasi residential (rumah baru). Investasi adalah suatu fungsi pendapatan dan tingkat bunga, dilihat dengan kaitannya I= (Y,i). Suatu pertambahan pada pendapatan akan mendorong investasi yang lebih besar, dimana tingkat bunga yang lebih tinggi akan menurunkan minat untuk investasi sebagaimana hal tersebut akan lebih mahal dibandingkan dengan meminjam uang. Walaupun jika suatu perusahaan memilih untuk menggunakan dananya sendiri untuk investasi, tingkat bunga menunjukkan suatu biaya kesempatan dari investasi dana tersebut daripada meminjamkan untuk mendapatkan bunga.

PDB (Produk Domestik Bruto) diartikan sebagai nilai keseluruhan semua barang dan jasa yang diproduksi didalam wilayah tersebut dalam jangka waktu tertentu (biasanya per tahun). PDB berbeda dari PNB (Produk Nasional Bruto) karena memasukkan pendapatan faktor produksi dari luar negeri yang bekerja di negara tersebut. Sehingga PDB hanya menghitung total produksi dari suatu negara tanpa memperhitungkan apakah produksi itu dilakukan dengan memakai faktor produksi dalam negeri atau tidak. Sebaliknya, PNB memperhatikan asal usul faktor produksi yang digunakan

PDB Nominal (disebut PDB Atas Dasar Harga Berlaku) merujuk kepada nilai PDB tanpa memperhatikan pengaruh harga. Sedangkan PDB riil (atau disebut PDB Atas Dasar Harga Konstan) mengoreksi angka PDB nominal dengan memasukkan pengaruh dari harga.

PDB dapat dihitung dengan memakai dua pendekatan, yaitu pendekatan pengeluaran dan pendekatan pendapatan, Dimana konsumsi adalah pengeluaran yang dilakukan oleh rumah tangga, investasi oleh sektor usaha, pengeluaran pemerintah oleh pemerintah, dan ekspor dan impor melibatkan sektor luar negeri. Sementara pendekatan pendapatan menghitung pendapatan yang diterima faktor produksi:

PDB = sewa + upah + bunga + laba

Di mana sewa adalah pendapatan pemilik faktor produksi tetap seperti tanah, upah untuk tenaga kerja, bunga untuk pemilik modal, dan laba untuk pengusaha. Secara teori, PDB dengan pendekatan pengeluaran dan pendapatan harus menghasilkan angka yang sama. Namun karena dalam praktek menghitung PDB dengan pendekatan pendapatan sulit dilakukan, maka yang sering digunakan adalah dengan pendekatan pengeluaran.

Return on Investment atau Rasio pengembalian atas investasi merupakan rasio perbandingan antara laba setelah pajak dengan total aktiva yang dimiliki oleh perusahaan (Martono dan Harjito, 2001:60).

Munawir (2004:89) *Return on Investment* atau *Return on Assets* (ROA) menunjukkan kemampuan perusahaan menghasilkan laba dari aktiva yang dipergunakan. Dengan mengetahui rasio ini, akan dapat diketahui apakah perusahaan efisien dalam memanfaatkan aktivanya dalam kegiatan operasional perusahaan. Rasio ini juga memberikan ukuran yang lebih baik atas profitabilitas

perusahaan karena menunjukkan efektifitas manajemen dalam menggunakan aktiva untuk memperoleh pendapatan.

Analisa *Return On Investment* (ROI) dalam analisa keuangan mempunyai arti yang sangat penting sebagai salah satu teknik analisa keuangan yang bersifat menyeluruh/komprehensif. Analisa Return On Investment (ROI) ini sudah merupakan tehnik analisa yang lazim digunakan oleh pimpinan perusahaan untuk mengukur efektivitas dari keseluruhan operasi perusahaan. Return On Investment (ROI) itu sendiri adalah salah satu bentuk dari ratio profitabilitas yang dimaksudkan untuk dapat mengukur kemampuan perusahaan dengan keseluruhan dana yang ditanamkan dalam aktiva yang digunakan untuk operasi perusahaan untuk menghasilkan keuntungan atau profitabilitas (Munawir, 1995:89) . Rasio ini dapat dihitung dengan rumus :

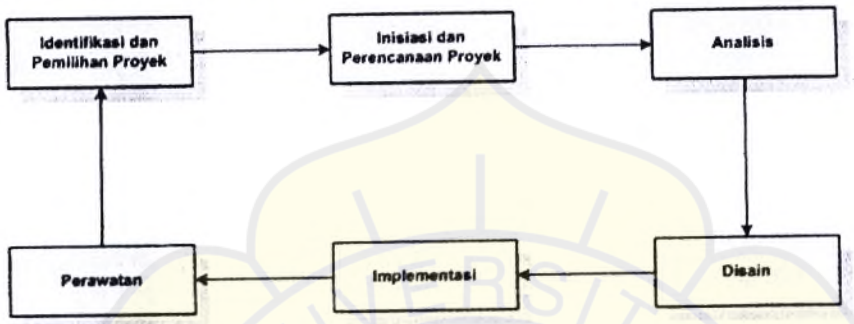
$$ROI = \frac{\text{Laba Bersih Setelah Pajak}}{\text{Modal}} \times 100\%$$

2.3. Konsep Rekayasa Perangkat Lunak

2.3.1. Definisi Rekayasa Perangkat lunak

Isi dari sub-bab ini tidak termasuk dalam standar kompetensi bidang keahlian RPL. Namun penulis memandang perlu disampaikan agar pembaca dapat mengetahui bagaimana sebenarnya rekayasa perangkat lunak dilakukan dan metode-metode apa saja yang biasa digunakan.

Pada rekayasa perangkat lunak, banyak model yang telah dikembangkan untuk membantu proses pengembangan perangkat lunak. Model-model ini pada umumnya mengacu pada model proses pengembangan sistem yang disebut *System Development Life Cycle (SDLC)* seperti terlihat pada gambar dibawah ini.



Gambar 2.2. *System Development Life Cycle (SDLC).* (Harris, 2003)

Setiap model yang dikembangkan mempunyai karakteristik sendiri-sendiri. Namun secara umum ada persamaan dari model-model ini, yaitu:

1. Kebutuhan terhadap definisi masalah yang jelas. Input utama dari setiap model pengembangan perangkat lunak adalah pendefinisian masalah yang jelas. Semakin jelas akan semakin baik karena akan memudahkan dalam penyelesaian masalah.
2. Tahapan-tahapan pengembangan yang teratur. Meskipun model-model pengembangan perangkat lunak memiliki pola yang berbeda-beda, biasanya model-model tersebut mengikuti pola umum *analysis – design – coding – testing - maintenance*.
3. *Stakeholder* (pemegang saham) berperan sangat penting dalam keseluruhan tahapan pengembangan. *Stakeholder* dalam rekayasa perangkat lunak dapat

berupa pengguna, pemilik, pengembang, pemrogram dan orang-orang yang terlibat dalam rekayasa perangkat lunak tersebut.

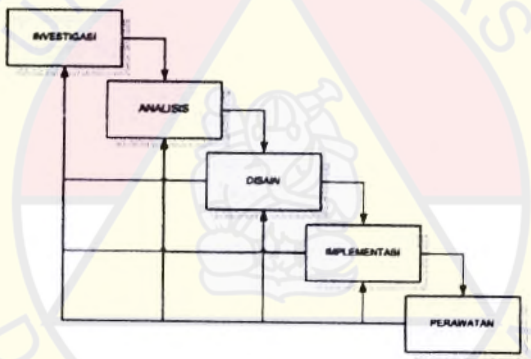
4. Dokumentasi merupakan bagian penting dari pengembangan perangkat luna . Masing-masing tahapan dalam model biasanya menghasilkan sejumlah tulisan, diagram, gambar atau bentuk-bentuk lain yang harus didokumentasi dan merupakan bagian tak terpisahkan dari perangkat lunak yang dihasilkan
5. .Keluaran dari proses pengembangan perangkat lunak harus bernilai ekonomis . Nilai dari sebuah perangkat lunak sebenarnya agak susah dinilai dalam rupiah. Namun efek dari penggunaan perangkat lunak yang telah dikembangkan haruslah memberi nilai tambah bagi organisasi. Hal ini dapat berupa penurunan biaya operasi, efisiensi penggunaan sumberdaya, peningkatan keuntungan organisasi, peningkatan *image* organisasi dan lain-lain.

Ada banyak model pengembangan perangkat lunak, antara lain *The Waterfall Model, Joint Application Development (JAD), Information Engineering (IE), Rapid Application Development (RAD)* termasuk di dalamnya *Prototyping, Unified Process (UP), Structural Analysis and Design (SAD)* dan *Framework For The Application of System Thinking (FAST).* Pada sub-bab ini akan dibahas tiga model pengembangan yaitu *The Waterfall Model, Prototyping,* dan *Unified Processs (UP).*

2.3.2. Metode Rekayasa Perangkat Lunak

2.3.2.1. Metode WaterFall

Model siklus hidup (*life cycle model*) adalah model utama dan dasar dari banyak model. Salah satu model yang cukup dikenal dalam dunia rekayasa perangkat lunak adalah *The Waterfall Model*. Ada 5 tahapan utama dalam *The Waterfall Model* seperti terlihat pada Gambar 2.3. Disebut *waterfall* (air terjun) karena memang diagram tahapan prosesnya mirip dengan air terjun yang bertingkat.



Gambar 2.3. Metode WaterFall, (Harris, 2003)

Tahapan-tahapan dalam *The Waterfall Model* secara ringkas adalah sebagai berikut:

1. Tahap investigasi dilakukan untuk menentukan apakah terjadi suatu masalah atau adakah peluang suatu sistem informasi dikembangkan. Pada tahapan ini studi kelayakan perlu dilakukan untuk menentukan apakah sistem informasi yang akan dikembangkan merupakan solusi yang layak
2. Tahap analisis bertujuan untuk mencari kebutuhan pengguna dan organisasi

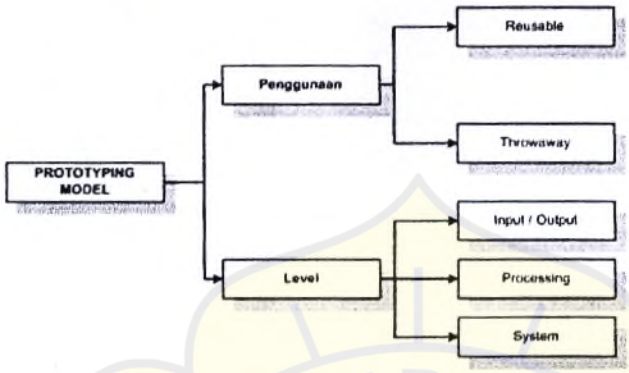
serta menganalisa kondisi yang ada (sebelum diterapkan sistem informasi yang baru).

3. Tahap desain bertujuan untuk menentukan spesifikasi detil dari komponen-komponen sistem informasi (manusia, *hardware, software, network* dan data) dan produk-produk informasi yang sesuai dengan hasil tahap analisa.
4. Tahap implementasi merupakan tahapan untuk mendapatkan atau mengembangkan *hardware* dan *software* (pengkodean program), melakukan pengujian, pelatihan dan perpindahan ke sistem baru.
5. Tahapan perawatan (*maintenance*) dilakukan ketika sistem informasi sudah dioperasikan. Pada tahapan ini dilakukan monitoring proses, evaluasi dan perubahan (perbaikan) bila diperlukan.

2.3.2.2. Metode Prototype

Prototyping adalah s*alah satu pendekatan dalam rekayasa perangkat lunak yang secara langsung mendemonstrasikan bagaimana sebuah perangkat lunak atau komponen-komponen perangkat lunak akan bekerja dalam lingkungannya sebelum tahapan konstruksi aktual dilakukan* (Howard, 1997).

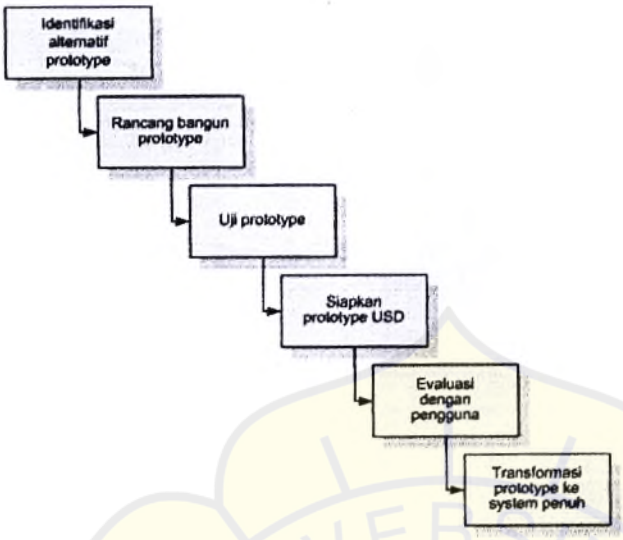
Prototyping model dapat diklasifikasikan menjadi beberapa tipe seperti terlihat pada gambar 2.4



Gambar 2.4. Klasifikasi prototyping model (Harris, 2003)

- *Reusable prototype : Prototype* yang akan ditransformasikan menjadi produk final.
- *Throwaway prototype : Prototype* yang akan dibuang begitu selesai menjalankan maksudnya.
- *Input/output prototype : Prototype* yang terbatas pada antar muka pengguna (*user interface*).
- *Processing prototype : Prototype* yang meliputi perawatan file dasar dan proses-proses transaksi.
- *System prototype : Prototype* yang berupa model lengkap dari perangkat lunak.

Tahap-tahap dalam *prototyping* boleh dikatakan merupakan tahap-tahap yang dipercepat. Strategi utama dalam *prototyping* adalah kerjakan yang mudah terlebih dahulu dan sampaikan hasilnya kepada pengguna sesegera mungkin. Harris (2003) membagi prototyping dalam enam tahapan seperti terlihat pada gambar 2.5.



Gambar 2.5. Tahapan-tahapan prototyping model (Harris, 2003)

Tahapan-tahapan secara ringkas dapat dijelaskan sebagai berikut:

1. Identifikasi kandidat *prototyping*. Kandidat dalam kasus ini meliputi *user interface* (menu, dialog, input dan output), file-file transaksi utama, dan fungsi-fungsi pemrosesan sederhana.
2. Rancang bangun *prototype* dengan bantuan *software* seperti *word processor, spreadsheet, database* , pengolah grafik, dan *software* CASE (*Computer-Aided System Engineering*).
3. Uji *prototype* untuk memastikan *prototype* dapat dengan mudah dijalankan untuk tujuan demonstrasi.
4. Siapkan *prototype* USD *(User's System Diagram)* untuk mengidentifikasi bagian-bagian dari perangkat lunak yang di- *prototype* -kan.
5. Evaluasi dengan pengguna untuk mengevaluasi *prototype* dan melakukan

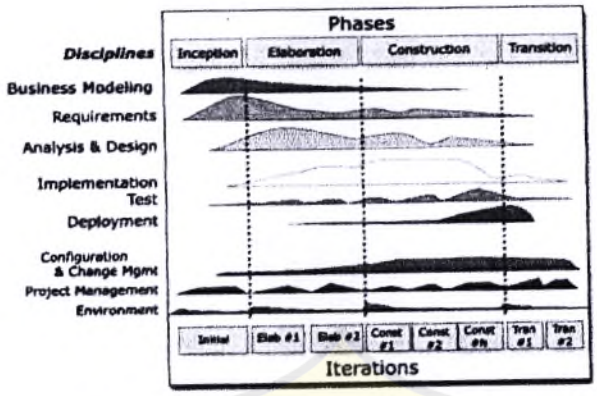
perubahan jika diperlukan.

6. Transformasikan *prototype menjadi* perangkat lunak yang beroperasi penuh dengan melakukan penghilangan kode-kode yang tidak dibutuhkan, penambahan program-program yang memang dibutuhkan dan perbaikan serta pengujian perangkat lunak secara berulang.

2.3.2.3. Metode Unified Process dan Unified Modeling Language

*Unified Process (*UP*)* atau kadang disebut sebagai *Unified Software Development Process* (USDP) adalah kerangka proses pengembangan yang bersifat *use-case-driven*, berpusat pada arsitektur perangkat lunak, interaktif dan tumbuh-kembang (Alhir, 2005). Kerangka pengembangan ini termasuk baru dalam metodologi pengembangan perangkat lunak. UP dapat diaplikasikan pada berbagai skala proyek, mulai dari skala kecil sampai dengan skala besar.

Daur hidup UP secara umum akan tampak seperti pada bagan di Gambar 2.6. Bagan ini biasa disebut sebagai *hump chart*. Pada bagan ini terlihat ada empat tahap pengembangan yaitu *inception, elaboration, construction* dan *transition.* Selain itu tampak pula sejumlah aktivitas (*disciplines*) yang harus dilakukan sepanjang pengembangan perangkat lunak, yaitu, *business modelling, requirements, analysis and design, implementation, test* . Tahap dan aktivitas tersebut akan dilakukan secara iteratif (Ambler, 2005).



Gambar 2.6. RUP *Life Cycle* (Ambler, 2005).

Penjelasan singkat untuk empat tahapan dalam UP adalah sebagai berikut :

1. *Inception.* Tahapan ini merupakan tahapan paling awal dimana aktivitas penilaian terhadap sebuah proyek perangkat lunak dilakukan. Tujuannya adalah untuk mendapatkan kesepakatan dari *stakeholder* sehubungan dengan tujuan dan dana proyek.
2. *Elaboration.* Tujuan dari tahap ini adalah untuk mendapatkan gambaran umum kebutuhan, persyaratan dan fungsi-fungsi utama perangkat lunak. Hal ini penting untuk mengetahui secara lebih baik resiko-resiko proyek, baik meliputi resiko arsitektur perangkat lunak, perencanaan, maupun implementasi. Pada tahap ini telah dimulai rancang bangun perangkat lunak secara iterative melalui aktivitas seperti *business modelling, requirements, analysis* dan *design* meskipun baru pada tahap awal.
3. *Construction.* Tujuan dari tahapan ini adalah membangun perangkat lunak sampai dengan saat perangkat lunak tersebut siap digunakan. Titik berat tahapan ini adalah pada penentuan tingkat prioritas kebutuhan / persyaratan, melengkapi spesifikasinya, analisis lebih dalam, desain solusi yang memenuhi

kebutuhan dan persyaratan, pengkodean dan pengujian perangkat lunak. Jika dimungkinkan versi awal dari perangkat lunak diuji cobakan untuk mendapatkan masukan dari pengguna.

4. *Transition.* Tahap ini difokuskan pada bagaimana menyampaikan perangkat lunak yang sudah jadi pada pengguna. Perangkat lunak akan secara resmi diuji oleh baik oleh penguji yang kompeten maupun oleh pengguna. Beberapa aktivitas seperti pemindahan pusat data dan pelatihan pengguna dan staf pendukung harus dilakukan pada tahap ini.

2.4. Tools Membangun Aplikasi

2.4.1. Microsoft Visual Studio.Net

Visual Studio.NET adalah sebuah *tools* pengembangan perangkat lunak untuk membangun aplikasi ASP Web, layanan XML Web, aplikasi *desktop*, dan aplikasi *mobile*. Visual Basic.NET, Visual C++.NET, Visual C#.NET, dan Visual J#.NET; semuanya menggunakan *Integrated Development Environment* (IDE) atau lingkungan pengembangan terintegrasi yang sama; yang membolehkan mereka untuk saling berbagi *tools* dan fasilitas dalam pembuatan solusi yang memadukan beberapa bahasa *(mixed language solutions)*. Selain itu, bahasa-bahasa ini mempengaruhi fungsionalitas dari .NET Framework, dan menyediakan pengaksesan ke kunci teknologi yang menyederhanakan proses pengembangan dari aplikasi ASP Web dan layanan XML Web.

Setelah itu, Microsoft pun berkonsentrasi dalam mengembangkan Microsoft.NET Framework 2.0, dan tentunya alat bantu untuk membangun program di atasnya. Hingga pada tahun 2005, mereka merilis versi terbaru dari Visual Basi .NET, yang kali ini disebut dengan Visual Basic 2005 (dengan membuang kata ".NET"), bersama-sama dengan beberapa aplikasi pengembangan lainnya.

Untuk rilis 2005 ini, Microsoft menambahkan beberapa fitur baru, di antaranya adalah :

- *Edit and Continue.* Fitur ini sebelumnya terdapat di dalam Visual Basic, akan tetapi dihapus di dalam Visual Basic.NET. Dengan keberadaan fitur ini, para *programmer* dapat memodifikasi kode pada saat program dieksekusi dan melanjutkan proses eksekusi dengan kode yang telah dimodifikasi tersebut.
- Evaluasi ekspresi pada saat waktu desain.
- Munculnya *Pseudo-NAMESPACE* "My", yang menyediakan :
  - Akses yang mudah terhadap beberapa area tertentu dari dalam .NET *Framework* yang tanpanya membutuhkan kode yang sangat signifikan.
  - Kelas-kelas yang dibuat secara dinamis (khususnya *My.Forms*).
- Peningkatan yang dilakukan terhadap konverter kode sumber dari Visual Basic ke Visual Basic .NET.
- Penggunaan kata kunci (*keyword*) *Using*, yang menyederhanakan penggunaan objek-objek yang membutuhkan pola *Dispose* untuk membebaskan sumber daya yang sudah tidak terpakai.

- *Just My Code*, yang menyembunyikan kode *reusable* yang ditulis oleh alat bantu *Integrated Development Environment* (IDE) Visual Studio .NET.
- Pengikatan sumber data (*Data Source Binding*), yang mampu mempermudah pengembangan aplikasi basis data berbasis klien *server*.

Fungsi-fungsi yang tersebut di atas (khususnya *My*) ditujukan untuk memfokuskan Visual Basic.NET sebagai sebuah *platform* pengembangan aplikasi secara cepat dan "menjauhkannya" dari bahasa C#.

Bahasa Visual Basic 2005 memperkenalkan fitur-fitur baru, yakni:

- Bawaan .NET Framework 2.0:
  - *Generics*
  - *Partial Class*, sebuah metode yang dapat digunakan untuk mendefinisikan beberapa bagian dari sebuah kelas di dalam sebuah berkas, lalu menambahkan definisinya di lain waktu, sangat berguna khususnya ketika mengintegrasikan kode pengguna dengan kode yang dibuat secara otomatis.
  - *Nullable Type*
- Komentar XML yang dapat diproses dengan menggunakan beberapa alat bantu seperti NDoc untuk membuat dokumentasi secara otomatis.
- *Operator overloading*
- Dukungan terhadap tipe data bilangan bulat tak bertanda *(unsigned integer)* yang umumnya digunakan di dalam bahasa lainnya.

2.4.2. Microsoft SQL server 2005

Beberapa definisi tentang *Database* :

Menurut Gordon C. Everest *Database* adalah koleksi atau kumpulan data yang mekanis, terbagi/*shared*, terdefinisi secara formal dan dikontrol terpusat pada organisasi.

Menurut C.J. Date *Database* adalah koleksi "data operasional" yang tersimpan dan dipakai oleh sistem aplikasi dari suatu organisasi.

- Data input adalah data yang masuk dari luar system.
- Data output adalah data yang dihasilkan system.
- Data operasional adalah data yang tersimpan pada system.

Menurut Toni Fabbri *Database* adalah sebuah sistem file-file yang terintegrasi yang mempunyai minimal *primary key* untuk pengulangan data.

Menurut S. Attre *Database* adalah koleksi data-data yang saling berhubungan mengenai suatu organisasi / *enterprise* dengan macam-macam pemakaiannya. Jadi sistem *database* adalah sistem penyimpanan data memakai komputer.

Pada dasarnya pengertian dari SQL Server itu sendiri adalah bahasa yang dipergunakan untuk mengakses data dalam basis data *relation*. Bahasa ini secara *defacto* adalah bahasa standar yang digunakan dalam manajemen basis data relasional. Saat ini hampir semua *server* basis data yang ada mendukung bahasa ini dalam manajemen datanya. SQL *server* 2005 merupakan salah satu produk dari *Relational Database Management System* (RDBMS).

Komponen SQL Server 2005

SQL Server 2005 terdiri atas beberapa komponen sebagai berikut:

a. *Relational Database Engine* : komponen utama atau jantung SQL Server 2005.

b. *Analysis Services* : basis dari solusi intelijen bisnis yang ampuh (*powerful*), dan mendukung aplikasi-aplikasi OLAP (*online analytical processing*), serta *data minning*.

c. *Data Transformation Service* (DTS): sebuah mesin untuk membuat solusi ekspor dan impor data, serta untuk mentransformasi data ketika data tersebut ditransfer.

d. *Notification Services* : sebuah *framework* untuk solusi dimana pelanggan akan dikirimi notifikasi ketika sebuah event muncul.

e. *Reporting Services* : service yang akan mengambil data dari SQL Server, dan menghasilkan laporan-laporan.

f. *Service Broke r*: sebuah mekanisme antrian yang akan menangani komunikasi berbasis pesan diantara service.

g. *Native HTTP Suppor* : dukungan yang memungkinkan SQL server 2005 yang (jika di *install* pada Windows Server 2003) akan merespon *request* terhadap HTTP *endpoint*, sehingga memungkinkan pembangunan sebuah *web service* untuk SQL Server tanpa menggunakan IIS.

h. *SQL Server Agent* : akan mengotomatiskan perawatan *database* dan mengatur *task*, *event* dan *alert*.

i. .NET CLR (*Common Language Runtime*) : memungkinkan pembuatan solusi menggunakan *managed code* yang ditulis dalam salah satu bahasa .NET.

j. *Replication*: serangkaian teknologi untuk menjalin dan mendistribusikan data dan obyek *database* dari sebuah *database* ke *database* lain, dan melakukan sinkronisasi untuk menjaga konsistensinya.

k. *Full-Text Search*: memungkinkan pengindeksan yang cepat dan flexibel untuk *query* berbasis kata kunci (terhadap data teks yang disimpan dalam *database*).

Fitur Untuk Administrasi Database

a. Database Mirroring

Perluas log pengiriman kemampuan dengan solusi *mirroring database*. Anda akan dapat menggunakan *mirroring database* untuk meningkatkan ketersediaan sistem SQL Server Anda dengan mendirikan *failover* otomatis ke server siaga.

b. Online Restore

Dengan SQL Server 2005, *database* administrator dapat melakukan *restore* operasi sementara langsung dari SQL Server yang berjalan. Online mengembalikan meningkatkan ketersediaan SQL Server karena hanya data yang dipulihkan tidak tersedia, sisa database tetap online dan tersedia.

c. Online Indexing Operations

Opsi Indeks online memungkinkan modifikasi *concurrent* (*update*, menghapus, dan menyisipkan) ke meja yang mendasari atau *data clustered index*

dan setiap indeks terkait selama data indeks bahasa eksekusi (DDL). Sebagai contoh, sementara *clustered index* sedang dibangun kembali, anda dapat terus memperbarui data yang mendasari dan melakukan *query* terhadap data.

d. Fast Recovery

Sebuah pilihan pemulihan baru yang lebih cepat meningkatkan ketersediaan *database* SQL Server. Administrator dapat menyambung kembali ke *database* pulih setelah log transaksi telah diperpanjang ke depan.

e. Standards-Based Information Access

Setiap objek, sumber data, atau komponen usaha intelijen bisa terkena menggunakan protokol berbasis standar seperti SOAP dan HTTP-menghilangkan kebutuhan pendengar yang tingkat menengah, seperti IIS, untuk mengakses antarmuka layanan Web yang terkena oleh SQL Server 2005.

f. SQL Server Management Studio

SQL Server 2005 termasuk SQL Server Management Studio, sebuah suite terintegrasi baru alat manajemen dengan fungsi untuk mengembangkan, menyebarkan, dan atasi masalah database SQL Server, serta perangkat tambahan untuk fungsi sebelumnya.

g. Dedicated Administrator Connection

SQL Server 2005 menyediakan koneksi administrator khusus yang dapat digunakan untuk mengakses *server* yang menjalankan bahkan jika *server* terkunci atau tidak tersedia. Kemampuan ini memungkinkan administrator untuk

memecahkan masalah pada *server* dengan menjalankan fungsi diagnostik atau pernyataan Transact-SQL.

h. Snapshot Isolation

Snapshot Isolasi (SI) tingkat disediakan pada tingkat *database*. Dengan SI, pengguna dapat mengakses baris komitmen terakhir menggunakan pandangan *transitionally* konsisten *database*. Kemampuan ini memberikan skalabilitas yang lebih besar.

i. Data Partitioning

Data partisi ditingkatkan dengan tabel partisi asli dan indeks yang memungkinkan pengelolaan efisien tabel besar dan indeks.

j. Replication Enhancements

Untuk *database* terdistribusi, SQL Server 2005 menyediakan skema komprehensif perubahan (DDL) replikasi, kemampuan pengawasan generasi berikutnya, dibangun pada replikasi dari Oracle ke SQL Server, menggabungkan replikasi lebih dari https, dan signifikan menggabungkan skalabilitas replikasi dan peningkatan kinerja. Selain itu, *peer-to-peer* fitur replikasi transaksional meningkatkan dukungan untuk skala data dengan menggunakan replikasi.

Fitur untuk Pengembang Database

a. Hosted Common Language Runtime

Dengan SQL Server 2005 pengembang dapat membuat objek *database* menggunakan bahasa asing seperti Microsoft Visual C # NET dan. Microsoft

Visual Basic. Pengembang juga dapat membuat dua jenis benda-*user-defined* baru dan agregat.

b. Native XML Support

Native XML data dapat disimpan, tanya, dan diindeks dalam database SQL Server-memungkinkan pengembang untuk membangun kelas baru sekitar tersambung aplikasi layanan *Web* dan di setiap *platform* atau perangkat.

c. ADO.NET version 2.0

Dari dukungan baru untuk SQL Jenis untuk Hasil *Multiple Set* Aktif (MARS), ADO.NET di SQL Server 2005 berkembang dataset akses dan manipulasi untuk mencapai skalabilitas dan fleksibilitas yang lebih besar.

d. *Security Enhancements* Keamanan Perangkat Tambahan

Model keamanan di SQL Server 2005 pengguna yang terpisah dari benda, menyediakan akses-butiran halus, dan memungkinkan kontrol yang lebih besar terhadap akses data. Selain itu, semua tabel sistem diimplementasikan sebagai pandangan, memberikan kontrol lebih atas objek sistem *database*.

e. Transact-SQL Enhancements

SQL Server 2005 menyediakan kemampuan bahasa baru untuk mengembangkan aplikasi database scalable. Peningkatan ini termasuk penanganan eror, kemampuan *query* rekursif, PIVOT operator relasional, BERLAKU, *ROW_NUMBER* dan fungsi baris lainnya peringkat, dan banyak lagi.

Fitur Sql Management Studio

Management Studio menggantikan baik SQL Server Enterprise Manager dan SQL Server *Query Analyzer*, dan menawarkan perbaikan atas alat-alat tua. SQL Server *Management Studio* menyediakan fitur berikut :

a. SQL Server Management Studio

SQL Server Management Studio dapat digunakan untuk membuat dan mengelola proyek *database*, yang berisi semua koneksi yang terkait, *query*, dan benda-benda lain yang terkait dengan aplikasi tersebut. Beberapa proyek dapat dikombinasikan menjadi solusi. sehingga lebih mudah untuk mengatur aplikasi yang kompleks

b. Integrated source control

Anda dapat menggunakan sistem kontrol sumber seperti Microsoft Visual *SourceSafe* langsung dari lingkungan SQL Server Management Studio.

c. Object Explorer

Obyek Explorer adalah alat grafis untuk menemukan dan mengelola *server*, *database*, dan objek *database*.

d. Wizards and Designers

SQL Server Management Studio menggabungkan *wizards* grafis dan desainer untuk membuat objek *database* dan *query* bangunan.

f. Reliable Messaging for Asynchronous Applications

Layanan *Broker* merupakan infrastruktur olah pesan kuat yang menyediakan pengiriman transaksional yang dapat diandalkan pesan penting antara *server*-dengan kinerja tinggi *scalable*-yang diharapkan dengan antrian *asynchronous*.

g. Visual Studio Integration

Integrasi ketat dengan Microsoft Visual Studio dan NET Framework. Pembangunan arus dan *debugging* aplikasi berbasis data. Pengembang dapat membangun objek *database*, seperti prosedur yang tersimpan, menggunakan bahasa apapun. NET dan mulus bisa debug di. NET dan *Transact*-SQL (TSQL) bahasa.

h. Web Services

Dengan SQL Server 2005 pengembang dapat mengembangkan layanan Web dalam database tier, membuat SQL Server protokol transfer *hypertext* (HTTP) pendengar dan menyediakan sebuah tipe baru kemampuan akses data untuk layanan-centric aplikasi Web.

i. Embedded Reports

Gunakan kontrol pelaporan sisi klien untuk laporan real-time embeded ke aplikasi pada saat desain.

j. Full-Text Search Enhancements

SQL Server 2005 mendukung kaya, aplikasi penuh-teks pencarian. Katalogisasi kemampuan memberikan fleksibilitas yang lebih besar atas apa yang di katalog. *Query* kinerja dan skalabilitas telah meningkat secara dramatis, dan

2.5. UML Sebagai Tools Pemodelan Sistem

Dalam pengembangan perangkat lunak dengan menggunakan UP, maka tidak lepas dari penggunaan notasi-notasi yang biasa disebut sebagai UML (*Unified Modeling Language*). Meskipun UP mensyaratkan penggunaan UML, (namun UML sendiri dapat digunakan pada berbagai metodologi yang lain bahkan dapat digunakan pada bidang selain sistem informasi. UML adalah bahasa pemodelan standar atau kumpulan teknik-teknik pemodelan untuk men-spesifikasi, memvisualisasi, mengkonstruksi dan mendokumentasi hasil kerja dalam pengembangan perangkat lunak (*Fowler, 2004*). UML lahir dari penggabungan banyak bahasa pemodelan grafis berorientasi obyek yang berkembang pesat pada akhir tahun 1980an dan awal 1990an.

Secara sederhana UML digunakan untuk menggambar sketsa sistem. Pengembang menggunakan UML untuk menyampaikan beberapa aspek dari sebuah perangkat lunak melalui notasi grafis. UML mendefinisikan notasi dan semantik. Notasi merupakan sekumpulan bentuk khusus yang memiliki makna tertentu untuk menggambarkan berbagai diagram piranti lunak dan semantik mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Ada beberapa jenis diagram yang disediakan dalam UML, antara lain adalah:

- *Use-case diagram*. Diagram ini berguna untuk menggambarkan interaksi antara pengguna dengan sebuah perangkat lunak
- *Activity diagram* . Diagram ini berguna untuk menggambarkan prosedur-prosedur perilaku perangkat lunak.

- *Class diagram.* Diagram ini berguna untuk menggambarkan class, fitur, dan hubungan-hubungan yang terjadi. Pada diagram ini pendekatan berorientasi obyek memegang peranan yang sangat penting.
- *Sequence diagram.* Diagram ini berguna untuk menggambarkan interaksi antar obyek dengan penekanan pada urutan proses atau kejadian.
- *State machine diagram.* Diagram ini digunakan untuk menggambarkan bagaimana suatu kejadian mengubah obyek selama masa hidup obyek tersebut.
- *Component diagram.* Diagram ini berguna untuk menggambarkan struktur dan koneksi komponen.
- *Deployment diagram.* Diagram ini berguna untuk menggambarkan detail bagaimana komponen di-deploy dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras).

2.5.1. Use Case Diagram

Use case class digunakan untuk memodelkan dan menyatakan unit fungsi/layanan yang disediakan oleh sistem (atau bagian sistem : subsistem atau class) ke pemakai. *Use case* dapat dilingkupi dengan batasan sistem yang diberi label nama sistem. *Use case* adalah sesuatu yang menyediakan hasil yang dapat diukur ke pemakai atau sistem eksternal.

Karakteristik :

- *Use cases* adalah interaksi atau dialog antara sistem dan *actor*, termasuk pertukaran pesan dan tindakan yang dilakukan oleh sistem.

- *Use cases* diprakarsai oleh actor dan mungkin melibatkan peran *actor* lain. *Use cases* harus menyediakan nilai minimal kepada satu *actor*.
- *Use cases* bisa memiliki perluasan yang mendefinisikan tindakan khusus dalam interaksi *atau use case* lain mungkin disisipkan.
- *Use case* class memiliki objek *use case* yang disebut skenario. Skenario menyatakan urutan pesan dan tindakan tunggal.

Komponen Pembentuk Use Case Diagram :

1. Actor

Pada dasarnya *actor* bukanlah bagian dari *use case diagram*, namun untuk dapat terciptanya suatu *use case diagram* diperlukan beberapa *actor*. *Actor* tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat, sistem lain) yang berinteraksi dengan sistem. Sebuah *actor* mungkin hanya memberikan informasi inputan pada sistem, hanya menerima informasi dari sistem atau keduanya menerima, dan memberi informasi pada sistem. *Actor* hanya berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case*. *Actor* digambarkan dengan *stick man*. *Actor* dapat digambarkan secara secara umum atau spesifik, dimana untuk membedakannya kita dapat menggunakan *relationship*.

Gambar 2.9 Actor, (*Fowler, 2004*)

2. Use Case

Use case adalah gambaran fungsionalitas dari suatu sistem, sehingga *customer* atau pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun.

Catatan : *Use case diagram* adalah penggambaran sistem dari sudut pandang pengguna sistem tersebut (*user*), sehingga pembuatan *use case* lebih dititik beratkan pada fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian.

Cara menentukan Use Case dalam suatu sistem:

a. Pola perilaku perangkat lunak aplikasi.
b. Gambaran tugas dari sebuah *actor*.
c. Sistem atau "benda" yang memberikan sesuatu yang bernilai kepada *actor*.
d. Apa yang dikerjakan oleh suatu perangkat lunak (bukan bagaimana cara mengerjakannya).

Gambar 2.10 Use Case (*Fowler, 2004*)

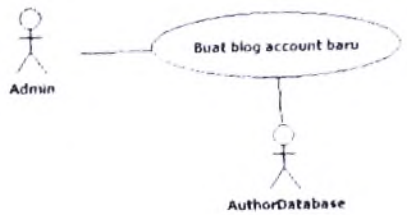
Relasi dalam Use Case

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

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

Tipe relasi/ *stereotype* 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 alarm.
3. *<<communicates>>*, mungkin ditambahkan untuk asosiasi yang menunjukkan asosiasinya adalah *communicates association*. Ini merupakan pilihan selama asosiasi hanya tipe *relationship* yang dibolehkan antara *actor* dan *use case*.



Gambar 2.11 *Use Case* Diagram (*Fowler, 2004*)

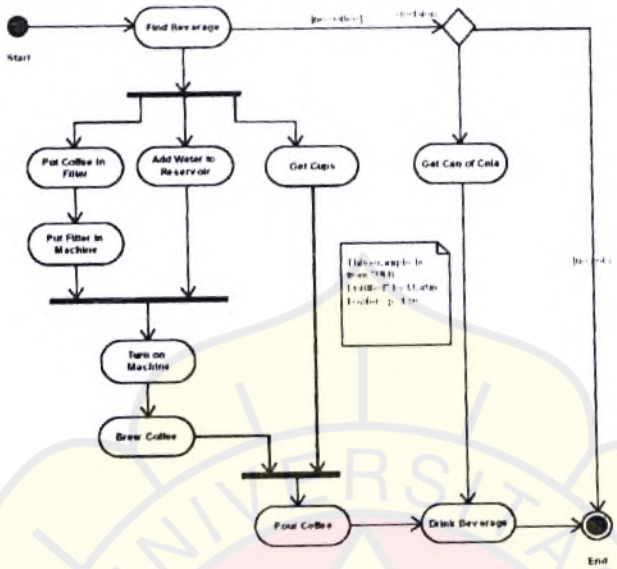
2.5.2. Activity Diagram

Activity diagrams menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity* diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus di mana sebagian besar *state* adalah *action* dan sebagian besar transisi dipicu oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara tepat, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state* , standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.



Gambar 2.12 contoh *activity diagram* tanpa *swimlan,* (*Fowler, 2004*)

2.5.3. Sequence Diagram

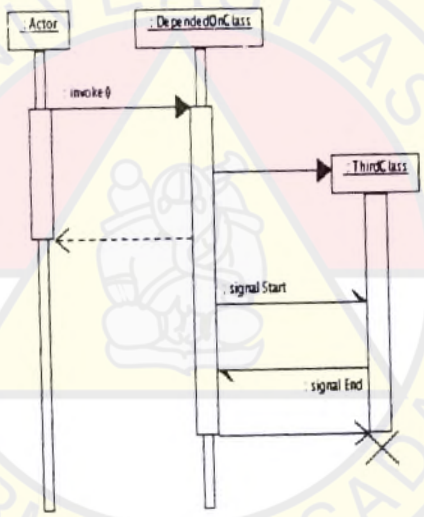
Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men- *trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

Masing-masing objek, termasuk *actor* memiliki *lifeline* vertikal. *Message*

digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain

berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class* . *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary, controller* dan *persistent entity* .

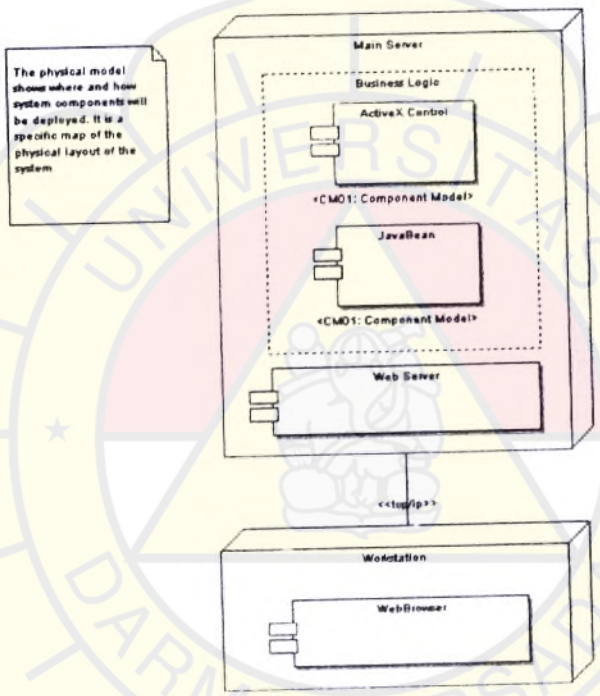


Gambar 2.13 Contoh *sequence diagram*, (*Fowler, 2004*)

2.5.4. Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di- *deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, *server* atau perangkat keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi *server*, dan hal-hal lain yang bersifat fisikal.

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



Gambar 2.14 Contoh *deployment diagram*, (*Fowler, 2004*)