

SEMESTER GANJIL 2017 / 2018

MATERI
PERKULIAHAN
REKAYASA PIRANTI
LUNAK

DAFTAR ISI

BAB 1 PENDAHULUAN	6
1.1 Perangkat Lunak.....	6
1.2 Rekayasa Perangkat Lunak.....	8
1.3 Proses Rekayasa Perangkat Lunak	10
1.4 Teknologi Informasi Sosial	12
BAB 2 ANALISIS DAN DESAIN SISTEM	15
2.1 Definisi Analisis Sistem.....	15
2.2 Teknik Pengumpulan Data	15
2.2.1 Teknik Wawancara.....	15
2.2.2 Teknik Observasi	16
2.2.3 Teknik Kuisisioner	17
2.3 Jenis Kebutuhan	17
2.4 Definisi Desain Sistem	18
BAB 3 SDLC.....	20
3.1 Pengertian SDLC.....	20
3.2 Model SDLC.....	21
3.2.1 Model WaterFall	22
3.2.2 Model Prototipe.....	24
3.2.3 Model Rapid Application Development (RAD).....	25
3.2.4 Model Iteratif.....	27
3.2.5 Model Spiral	28
BAB 4 PEMOGRAMAN TERSTRUKTUR	31
4.1 Pengertian Pemograman Terstruktur	31
4.2 DFD.....	32
4.3 Kamus Data	35
BAB 5 PEMODELAN UML	37
5.1 Kompleksitas Pengembangan Perangkat Lunak	37
5.2 Pemodelan	38
5.3 Pengenalan UML	39
5.4 Sejarah UML.....	39
5.5 Diagram UML	41
5.6 Class Diagram.....	42
5.7 Object Diagram	43
5.8 Component Diagram.....	44
5.9 Composite Structure Diagram.....	45

5.10 Package Diagram.....	46
5.11 Deployment Diagram	47
5.12 Use Case Diagram	48
5.13 Activity Diagram.....	52
5.14 State Machine Diagram.....	53
5.15 Sequence Diagram	54
5.16 Communication Diagram	56
5.17 Timing Diagram	57
5.18 Interaction Overview Diagram.....	58
BAB 6 ANALISIS DAN DESAIN BERORIENTASI OBJEK.....	61
6.1 Analisis Berorientasi Objek	61
6.2 Desain Berorientasi Objek.....	64
6.3 CASE Tools.....	65
6.4 RUP.....	67
6.4.1 Kelebihan RUP	68
6.4.2 Fase RUP.....	70
BAB 7 PEMOGRAMAN BERORIENTASI OBJEK	74
7.1 Pengertian Pemograman Berorientasi Objek	74
7.2 Konsep Dasar Berorientasi Objek	76
7.3 Perbandingan Pendekatan OO dan Terstruktur	80

DAFTAR GAMBAR

gambar 1.Tahapan Umum Rekayasa Perangkat Lunak.....	10
gambar 2.ilustrasi konvesi paralel	13
gambar 3. ilustrasi konversi langsung.....	13
gambar 4. ilustrasi konversi per fase	13
gambar 5.ilustrasi konversi pilot.....	13
gambar 6. Ilustrasi model waterfall	22
gambar 7.Ilustrasi model prototipe	24
gambar 8. Ilustrasi model RAD.....	26
gambar 9. Ilustrasi model iteratif	27
gambar 10. Ilustrasi model spiral.....	29
gambar 11. Ilustrasi pemgoraman terstruktur	31
gambar 12.Contoh DFD yang dikembangkan Chris Gane & Trish Sarson	32
gambar 13.Diagram UML	41
gambar 14.Diagram Deployment Sistem Client / Server	47
gambar 15.contoh State machine diagram	54
gambar 16.contoh timing diagram	57
gambar 17.contoh interaction occurence	58
gambar 18.contoh interaction element.....	59
gambar 19.diagram Booch.....	62
gambar 20. Contoh diagram OMT	63
gambar 21.contoh diagram kelas Coad Yourdon.....	64
gambar 22.Proses iteratif RUP.....	67
gambar 23.Alur hidup RUP.....	70
gambar 24.ilustrasi kelas	77
gambar 25.Ilustrasi kelas dan objek.....	78

BAB 1 PENDAHULUAN

1.1 Perangkat Lunak

Perangkat lunak (*software*) adalah program komputer yang terasosiasi dengan dokumentasi perangkat lunak seperti dokumentasi kebutuhan, model desain, dan cara penggunaan (*user manual*). Sebuah program komputer tanpa terasosiasi dengan dokumentasinya maka belum dapat disebut perangkat lunak (*software*). Sebuah perangkat lunak juga sering disebut dengan sistem perangkat lunak. Sistem berarti kumpulan komponen yang saling terkait dan mempunyai satu tujuan yang ingin dicapai.

Sistem perangkat lunak berarti sebuah sistem yang memiliki komponen berupa perangkat lunak yang memiliki hubungan satu sama lain untuk memenuhi kebutuhan pelanggan (*customer*). Pelanggan (*customer*) adalah orang atau organisasi yang memesan atau membeli perangkat lunak (*software*) dari pengembang perangkat lunak atau bisa dianggap bahwa pelanggan (*customer*) adalah orang atau organisasi yang dengan sukarela mengeluarkan uang untuk memesan atau membeli perangkat lunak. User atau pemakai perangkat lunak adalah orang yang memiliki kepentingan untuk memakai atau menggunakan perangkat lunak untuk memudahkan pekerjaannya.

Karakter perangkat lunak adalah sebagai berikut:

- Perangkat lunak dibangun dengan rekayasa (*software engineering*) bukan diproduksi secara manufaktur atau pabrikan.
- Perangkat lunak tidak pernah usang (*"wear out"*) karena kecacatan dalam perangkat lunak dapat diperbaiki.
- Barang produksi pabrikan biasanya komponen barunya akan terus diproduksi, sedangkan perangkat lunak biasanya terus diperbaiki seiring bertambahnya kebutuhan.

Aplikasi dari perangkat lunak adalah sebagai berikut:

- Perangkat lunak sistem (*system software*)
Adalah kumpulan program dalam hal ini program yang satu ditulis untuk memenuhi kebutuhan program lainnya.
- Perangkat lunak waktu nyata (*real-time software*)
Merupakan perangkat lunak yang memonitor, menganalisis, mengontrol sesuatu secara waktu nyata (*real-time*). Reaksi yang dibutuhkan pada perangkat lunak harus langsung menghasilkan respon yang diinginkan.
- Perangkat lunak bisnis (*business software*)

Merupakan perangkat lunak pengelola informasi bisnis (seperti akuntansi, penjualan, pembayaran, penyimpanan)

- Perangkat lunak untuk keperluan rekayasa dan keilmuan (*engineering and scientific software*)
Merupakan perangkat lunak yang mengimplementasikan algoritma yang terkait dengan keilmuan ataupun perangkat lunak yang membantu keilmuan, misalkan perangkat lunak di bidang astronomi, di bidang matematika dan lain sebagainya.
- Perangkat lunak tambahan untuk membantu mengerjakan suatu fungsi dari perangkat lunak yang lainnya (*embedded software*)
Misalnya perangkat lunak untuk mencetak dokumen ditambahkan agar perangkat lunak yang memerlukan dapat mencetak laporan, maka perangkat lunak untuk mencetak dokumen ini disebut embedded software.
- Perangkat lunak komputer personal (*personal computer software*)
Merupakan perangkat lunak untuk PC misalnya perangkat lunak pemroses teks, pemroses grafik dan lain sebagainya.
- Perangkat lunak berbasis web (*web based software*)
Merupakan perangkat lunak yang dapat diakses dengan menggunakan browser.
- Perangkat lunak berinteligensi buatan (*artificial intelligence software*)
Merupakan perangkat lunak yang menggunakan algoritma tertentu untuk mengelola data sehingga seakan-akan memiliki inteligensi seiring bertambahnya data yang diproses.

Produk perangkat lunak yang dibuat oleh pengembang (*developer*) perangkat lunak terdiri dari dua jenis:

- Produk generik
Produk perangkat lunak yang dibuat oleh pengembang perangkat lunak untuk dijual atau dipopulerkan (*open source*) tanpa ada yang memesan terlebih dahulu, perangkat lunak yang termasuk dalam produk generik misalnya perangkat lunak sistem operasi, perangkat lunak pendukung perkantoran untuk membuat dokumen, slide presentasi, atau perhitungan dalam bentuk papersheet dan lain sebagainya.
- Produk pemesanan
Produk perangkat lunak yang dibuat karena ada pelanggan yang melakukan pemesanan, misalnya sebuah instansi memerlukan perangkat lunak untuk memenuhi proses bisnis yang terjadi di instansinya, maka instansi itu akan bekerja sama dengan pengembang untuk membuat perangkat lunak yang diinginkan.

1.2 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak (*software engineering*) merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi yang dipercaya dan bekerja secara efisien menggunakan mesin. Perangkat lunak banyak dibuat dan pada akhirnya sering tidak digunakan karena tidak memenuhi kebutuhan pelanggan atau bahkan karena masalah non-teknis seperti keengganan pembeli perangkat lunak (*user*) untuk mengubah secara kerja dari manual ke otomatis atau ketidakmampuan user menggunakan komputer. Oleh karena itu rekayasa perangkat lunak dibutuhkan agar perangkat lunak yang dibuat tidak hanya menjadi perangkat lunak yang tidak terpakai.

Rekayasa perangkat lunak lebih fokus pada praktik pengembangan perangkat lunak dan mengirimkan perangkat lunak yang bermanfaat kepada pelanggan (*customer*). Adapun ilmu komputer lebih fokus pada teori dan konsep dasar perangkat komputer. Rekayasa perangkat lunak lebih fokus pada bagaimana membuat perangkat lunak yang memenuhi kriteria berikut:

- Dapat terus dipelihara setelah perangkat lunak selesai dibuat seiring berkembangnya teknologi dan lingkungan (*maintainability*)
- Dapat diandalkan dengan proses bisnis yang dijalankan dan perubahan yang terjadi (*dependability dan robust*)
- Efisien dari segi sumber daya dan penggunaan
- Kemampuan untuk dipakai sesuai dengan kebutuhan (*usability*)

Dari kriteria di atas maka perangkat lunak yang baik adalah perangkat lunak yang dapat memenuhi kebutuhan pelanggan (*customer*) atau user (pemakai perangkat lunak) atau berorientasi pada pelanggan atau pemakai perangkat lunak, bukan berorientasi pada pembuat atau pengembang perangkat lunak.

- ❖ Perangkat lunak yang baik adalah perangkat lunak yang fokus pada penggunaan atau pelanggan.

Pekerjaan yang terkait dengan rekayasa perangkat dapat dikategorikan menjadi tiga buah kategorikan menjadi tiga buah kategori umum tanpa melihat area dari aplikasi, ukuran proyek perangkat lunak, atau kompleksitas perangkat lunak yang akan dibuat. Setiap fase dialamatkan pada satu atau lebih pertanyaan yang diajukan sebelumnya.

Fase pendefinisian fokus pada "*what*" yang artinya harus mencari tahu atau mengidentifikasi informasi apa yang harus diproses, seperti apa fungsi dan performansi yang diinginkan, seperti apa perilaku sistem yang diinginkan, apa kriteria validasi yang dibutuhkan untuk mendefinisikan sistem.

Fase pengembangan yang fokus dengan "*how*" yang artinya selama tahap pengembangan perangkat lunak seorang perekraya perangkat lunak (*software engineering*) berusaha untuk mendefinisikan bagaimana data distrukturkan dan bagaimana fungsi-fungsi yang dibutuhkan diimplementasikan didalam arsitektur perangkat lunak, bagaimana detail prosedural

diimplementasikan, bagaimana karakter antarmuka tampilan, bagaimana desain ditranslasikan ke bahasa pemrograman, dan bagaimana pengujian akan dijalankan.

Fase pendukung (*support phase*) fokus pada perubahan yang terasosiasi pada perbaikan kesalahan (*error*), adaptasi yang dibutuhkan pada lingkungan perangkat lunak yang terlibat, dan perbaikan yang terjadi akibat perubahan kebutuhan pelanggan (*customer*). Fase pendukung terdiri dari empat tipe perubahan antar lain:

- Koreksi (*correction*)
Walaupun dengan jaminan kualitas yang terbaik, akan selalu ada kecacatan atau keinginan pelanggan (*customer*) yang tidak tertangani oleh perangkat lunak. Pemeliharaan dengan melakukan perbaikan terhadap kecacatan perangkat lunak.
- Adaptasi (*adaptation*)
Pada saat tertentu lingkungan asli (seperti CPU, sistem operasi, aturan bisnis, karakteristik produk luar) dimana perangkat lunak dikembangkan akan mengalami perubahan. Pemeliharaan adaptasi merupakan tahap untuk memodifikasi perangkat lunak guna mengakomodasi perubahan lingkungan luar dimana perangkat lunak dijalankan.
- Perbaikan (*enhancement*)
Sejalan dengan digunakannya perangkat lunak, maka pelanggan (*customer*) atau pemakainya (*user*) akan mengenali fungsi tambahan yang dapat mendatangkan manfaat. Pemeliharaan perfektif atau penyempurnaan melakukan eksistensi atau penambahan pada kebutuhan fungsional sebelumnya.
- Pencegahan (*prevention*)
Keadaan perangkat lunak komputer sangat dimungkinkan untuk perubahan. Oleh karena itu, pemeliharaan pencegahan (*preventive*) atau sering disebut juga dengan rekayasa ulang sistem (*software reengineering*) harus dikondisikan untuk mampu melayani kebutuhan pemakainya (*user*). Untuk menanggulangi hal ini maka perangkat lunak harus dirancang dan dikondisikan untuk mengakomodasi perubahan kebutuhan yang diinginkan oleh pemakai (*user*). Di lain sisi biasanya setelah perangkat lunak dikirimkan ke user maka masih dibutuhkan asistensi dan help desk dari pengembang perangkat lunak.

Tantangan yang dihadapi dari proses rekayasa perangkat lunak adalah sebagai berikut:

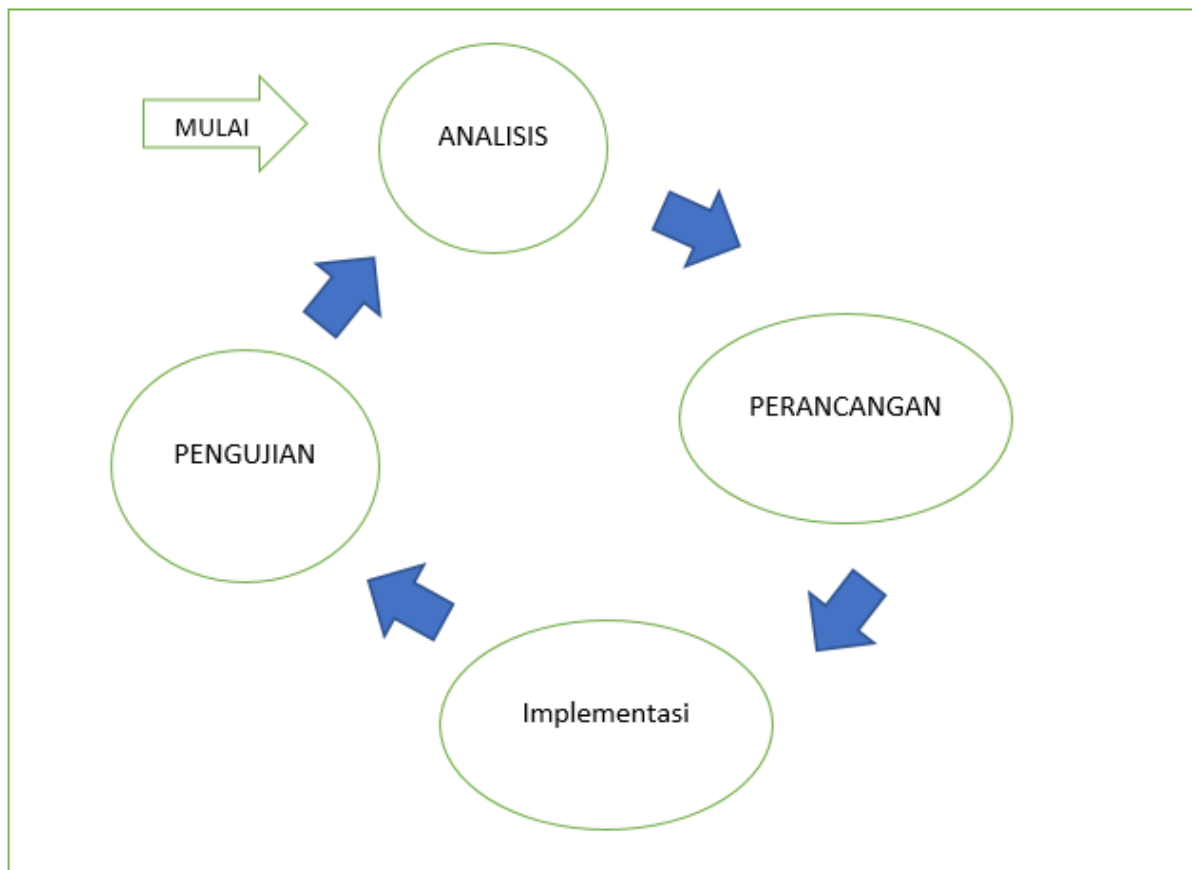
- Tantangan warisan dimana perangkat lunak dikembangkan selama bertahun-tahun oleh orang-orang yang berbeda, hal ini dapat menyebabkan ketidakpahaman atau perubahan tujuan pembuatan perangkat lunak;

- Tantangan heterogenitas dimana perangkat lunak harus dapat beradaptasi dengan teknologi yang terus berkembang dengan semakin luasnya lingkungan distribusi perangkat lunak;
- Tantangan pengiriman bahwa perangkat lunak dengan skala besar dan kompleks sekalipun dapat sampai ke tangan pelanggan (*customer*) atau user dengan cepat dan kualitas tetap terjaga.

Ada suatu hal tambahan dalam rekayasa perangkat lunak yang berbasis web pada tahap desainnya yaitu bagaimana menentukan aliran halaman web.

1.3 Proses Rekayasa Perangkat Lunak

Proses rekayasa perangkat lunak dilakukan selama pembangunan perangkat lunak. Proses-proses yang dilakukan dalam rekayasa perangkat lunak secara garis besar adalah sebagai berikut:



gambar 1. Tahapan Umum Rekayasa Perangkat Lunak.

Proses proses pada gambar diatas dapat dilakukan berulang kali sampai perangkat lunak memenuhi kebutuhan pelanggan atau user. Untuk membangun perangkat lunak yang benar-benar baik maka diperlukan tahapan-tahapan rekayasa perangkat lunak. Perangkat lunak yang dibangun tanpa melalui tahap rekayasa perangkat lunak yang baik maka akan seperti membuat baju tanpa pola dan rencana model baju yang baik. Jika baju dibuat tanpa pola,

maka yang terjadi adalah baju hasil tambahl sulam. Seperti halnya pembuatan perangkat lunak tanpa melalui proses rekayasa perangkat lunak, maka hanya akan menghasilkan perangkat lunak “tambal sulam”, begitu kurang makan akan ditambal. Tentu saja hasil perangkat lunak “tambal sulam” tidak akan bagus. Contoh lain misalnya sebuah bangunan tanpa rencana yang matang, maka yang terjadi bangunan adalah “tambal sulam” yang akhirnya dirobuhkan atau bahkan roboh karena pembuatannya tanpa perencanaan yang matang.

Perkembangan teknologi informasi saat ini sangat mempermudah dan mempercepat proses rekayasa perangkat lunak dengan adanya bermacam-macam *tools desain, tools pengembangan* seperti IDE (*Integrated Development Environment*), *application, framework*, dan lain lain. Hal tersebut sering disalahartikan oleh pengembang perangkat lunak pemula dan pelanggan/pengguna perangkat lunak bahwa pengembang perangkat lunak bisa diselesaikan dalam satu malam.

Anggapan tersebut sangat tidak tepat. Rekayasa perangkat lunak terdiri dari beberapa kegiatan yang harus dilakukan. Jika tahapan-tahapan tersebut tidak dilalui dengan baik, hampir bisa dipastikan perangkat lunak yang dihasilkan tidak akan mempunyai kualitas yang baik. Jadi, tidak ada perangkat lunak yang tidak dihasilkan dengan baik hanya dalam satu malam.

- ❖ Rekayasa perangkat lunak bukan seperti sulap yang bisa diselesaikan dalam sekejap, ada beberapa proses yang harus dilalui agar menghasilkan perangkat lunak yang berkualitas.

Proses perangkat lunak (*software process*) adalah sekumpulan aktifitas yang memiliki tujuan untuk mengembangkan atau mengubah perangkat lunak. Secara umum proses perangkat lunak terdiri dari :

- Pengumpulan Spesifikasi (*Specification*)
Mengetahui apa saja yang harus dapat dikerjakan sistem perangkat lunak dan batasan pengembangan perangkat lunak.
- Pengembangan (*Development*)
Pengembangan perangkat lunak untuk menghasilkan sistem perangkat lunak.
- Validasi (*Validation*)
Memeriksa apakah perangkat lunak sudah memenuhi keinginan pelanggan (*customer*)
- Evolusi (*Evolution*)
Mengubah perangkat lunak untuk memenuhi perubahan kebutuhan pelanggan (*customer*)

1.4 Teknologi Informasi Sosial

Teknologi informasi sosial adalah *social information technology (social IT)* adalah faktor atau aspek social yang berkaitan dengan implementasi suatu teknologi informasi. Sebuah produk perangkat lunak tidak hanya berkutat di masalah teknologi dan teknis. Sering sebuah produk perangkat lunak hanya dibuat berdasarkan sudut pandang pengembang perangkat lunak yang memiliki kecenderungan berpola teknis.

Sebuah perangkat lunak dianggap berkualitas jika memenuhi kebutuhan pelanggan (*customer*) dan sesuai keinginan pelanggan (*customer*). Sering kendala dari pengembangan perangkat lunak bukan berada pada masalah teknis (teknologi perangkat lunak dan perangkat keras) tapi pada kondisi lingkungan pelanggan, misalnya ternyata yang akan menjadi *user* atau pemakai perangkat lunak belum bisa mengoperasikan komputer, atau keenganan *user* untuk mengubah kebiasaannya bekerja menggunakan perangkat lunak yang baru atau dari manual menjadi menggunakan perangkat lunak, lingkungan pelanggan apakah dapat dijalankan perangkat lunak yang akan dikembangkan, karena sebaik apapun perangkat lunak jika tidak mampu digunakan di lingkungan pelanggan maka perangkat lunak itu hanya akan menjadi 'artifak' yang disimpan di gudang atau bahkan dibuang.

Hal-hal yang harus dilakukan sebelum mengembangkan perangkat lunak di lingkungan tertentu maka harus dicari tahu:

- Pengetahuan lingkungan tentang teknologi informasi dari komputer
- "*social knowledge*" atau "*local knowledge*" (pengetahuan mengenai budaya lokal) di lingkungan yang akan dikembangkan perangkat lunak, apakah memungkinkan untuk dikembangkan perangkat lunak.
- Pengetahuan tentang apa saja yang bisa dibatasi dan yang tidak, sehingga saat pengembangan perangkat lunak dapat mendefinisikan aturan main dari perangkat lunak.

Setelah perangkat lunak dikembangkan tetap masih diperlukan adanya sosialisasi perangkat lunak dengan mengadakan pelatihan secara bertahap, karena mengubah kebiasaan sebuah lingkungan kesesuatu yang baru tidaklah gampang dan yang akan menggunakan perangkat lunak memiliki karakter yang berbeda-beda. Memang tidak harus menuruti semua karakter, tapi setidaknya dapat dicari titik tengah yang dapat diterima semua pihak. Komunikasi yang baik antara pengembang dan pelanggan atau user sangat dibutuhkan agar terjalin kerja sama yang baik dan saling menguntungkan.

Melakukan konversi dari cara kerja yang lama ke cara kerja yang baru menggunakan perangkat lunak yang dikembangkan perlu dilakukan secara bertahap, karena perubahan yang ekstrim akan menghabiskan lebih banyak sumber daya, dana, dan waktu. Beberapa cara konversi adalah sebagai berikut:

- Konversi paralel



gambar 2. ilustrasi konvesi paralel

Konversi paralel dilakukan dengan melakukan beberapa waktu transisi dimana ada waktu kedua sistem (sistem lama dan sistem baru) berjalan bersama untuk keperluan transisi sampai sistem baru dapat berjalan mandiri. Sumber daya yang dibutuhkan pada konversi paralel akan banyak terkuras pada waktu transisi.

- Konversi Langsung



gambar 3. ilustrasi konversi langsung

Konversi langsung dilakukan karena sistem lama secara ekstrim langsung diganti dengan sistem yang baru. Konversi ini akan mengalami waktu yang sangat sulit di awal berjalannya sistem baru.

- Konversi Per Fase



gambar 4. ilustrasi konversi per fase

Konversi per fase dilakukan dengan berpindah per fase dan dari sistem lama ke sistem baru misalkan pada awal konversi hanya pada pekerjaan memasukkan data-data saja, pada tahap berikutnya mulai menggunakan proses perhitungan, lalu fase berikutnya mulai menggunakan proses pelaporan sistem baru, dan seterusnya (lebih fokus pada per fungsi sistem).

- Konversi pilot atau single location



gambar 5. ilustrasi konversi pilot

Konversi pilot dilakukan dengan melakukan konversi perunit kerja atau per lokasi di dalam sebuah lingkungan kerja. Misalnya pada tahap awal unit kerja yang sistemnya berubah adalah bagian keuangan, berikutnya pada bagian sumber daya manusia, dan seterusnya.

Pengembang perangkat lunak yang berhasil dan dapat diterima dengan baik tidak hanya memperhatikan masalah teknis, tapi memperhatikan masalah nonteknis seperti permasalahan sosial dan mencermati isu-isu yang sedang dikembangkan di masyarakat

Soal Latihan

1. Sebutkan definisi atau pengertian dari istilah-istilah sebagai berikut:
 - a. Perangkat lunak (*Software*)
 - b. Sistem (*system*)
 - c. Aplikasi (*application*)
 - d. Rekayasa Piranti lunak (*Software engineering*)
 - e. *Software engineer*
 - f. *Software developer*
 - g. *Programmer*
 - h. Social IT
2. Apakah proses produksi perangkat lunak identik atau serupa dengan proses produksi pada pabrik / manufaktur pembuatan mobil? Jelaskan alasannya
3. Bidang rekayasa perangkat lunak apakah sebagian bagian dari seni atau bagian dari teknik? Jelaskan alasannya
4. Mengapa ada proses-proses atau tahapan-tahapan yang harus dilakukan dalam rekayasa perangkat lunak?
5. Mengapa rekayasa perangkat lunak sebaiknya fokus pada pelanggan atau pengguna?
6. Mengapa faktor sosial dari teknologi informasi sering sekali diabaikan oleh pengembang aplikasi?
7. Mengapa faktor sosial dari teknologi informasi perlu untuk diperhatikan?
8. Sebutkan kelebihan dan kekurangan masing-masing metode konversi sistem yang sudah disebutkan di atas

BAB 2 ANALISIS DAN DESAIN SISTEM

2.1 Definisi Analisis Sistem

Kegiatan analisis sistem adalah kegiatan untuk melihat sistem yang sudah berjalan, melihat bagian mana yang bagus dan tidak bagus, dan kemudian mendokumentasikan kebutuhan yang akan dipenuhi dalam sistem yang baru. Hal tersebut terlihat sederhana, namun sebenarnya tidak. Banyak hambatan yang akan ditemui dalam proses tersebut.

Pada banyak proyek sistem informasi, proses analisis dan desain sering kali berjalan bersamaan. Jadi selama kegiatan analisis, kegiatan desain juga dilakukan. Hal ini dilakukan karena pada banyak kasus, user sering kesulitan untuk mendefinisikan kebutuhan mereka. Jadi mereka akan melihat gambar rancangan sistem yang baru. Khususnya rancangan antarmuka.

Oleh karena itu, seringkali batasan mengenai bagian mana yang dianggap sebagai analisis dan bagian mana yang dianggap sebagai desain banyak terjadi perbedaan.

Misalnya untuk pemograman berorientasi objek ada yang mengatakan bahwa *use case*, *analysis class*, dan *sequence diagram* merupakan bagian dari analisis. Namun, ada juga pihak lain yang menyatakan bahwa *use case* dan *sequence diagram* merupakan bagian dari desain, dan *analysis class* tidak ada karena sudah ada *design class*.

Pada buku ini yang dibahas pada bagian analisis adalah bagaimana metode pengumpulan data dan bagaimana mendokumentasikannya. Sedangkan *use case*, *class diagram*, dan *sequence diagram* dianggap merupakan bagian dari desain sistem dan akan dibahas pada bab yang terkait dengan UML.

2.2 Teknik Pengumpulan Data

Hal pertama yang dilakukan dalam analisis sistem adalah melakukan pengumpulan data. Ada beberapa teknik pengumpulan data yang sering dilakukan yaitu sebagai berikut.

- Teknik Wawancara
- Teknik Observasi
- Teknik Kuisioner

2.2.1 Teknik Wawancara

Pengumpulan data dengan menggunakan wawancara mempunyai beberapa keuntungan sebagai berikut.

- Lebih mudah dalam menggali bagian sistem mana yang dianggap baik dan bagian mana yang dianggap kurang baik.

- Jika ada bagian tertentu yang menurut anda perlu untuk digali lebih dalam, anda dapat langsung menanyakan pada narasumber
- Dapat menggalai kebutuhan user secara lebih bebas
- User dapat mengungkapkan kebutuhannya secara lebih bebas

Selain mempunyai beberapa kelebihan tersebut, teknik wawancara juga mempunyai beberapa kelemahan. Berikut ini adalah beberapa kelemahan dari teknik wawancara.

- Wawancara akan sulit dilakukan jika narasumber kurang dapat mengungkapkan kebutuhannya
- Pertanyaan dapat menjadi tidak terarah, terlalu fokus pada hal-hal tertentu dan mengabaikan bagian lainnya

Berikut ini adalah beberapa panduan dalam melakukan kegiatan wawancara agar memperoleh data yang diharapkan.

- Buatlah jadwal wawancara dengan narasumber dan beritahukan maksud dan tujuan wawancara
- Buatlah panduan wawancara yang akan anda jadikan arahan agar pertanyaan dapat fokus kepada hal-hal yang dibutuhkan
- Gunakan pertanyaan yang jelas dan mudah dipahami.
- Cobalah untuk menggali mengenai kelebihan dan kekurangan sistem yang telah berjalan sebelumnya
- Anda boleh berimprovisasi dengan mencoba menggali bagian-bagian tertentu yang menurut anda penting
- Catat hasil wawancara tersebut.

2.2.2 Teknik Observasi

Pengumpulan data dengan menggunakan observasi mempunyai keuntungan yaitu:

- Analis dapat melihat langsung bagaimana sistem lama berjalan
- Mampu menghasilkan gambaran lebih baik jika dibandingkan dengan teknik lainnya

Sedangkan kelemahan dengan menggunakan teknik observasi adalah

- Membutuhkan waktu cukup lama karena jika observasi waktunya sangat terbatas maka gambaran sistem secara keseluruhan akan sulit untuk diperoleh
- Orang-orang yang sedang diamati biasanya perilakunya akan berbeda dengan perilaku sehari-hari (cenderung berusaha terlihat baik). Hal ini akan menyebabkan gambaran yang diperoleh selama observasi akan berbeda dengan perilaku sehari-hari
- Dapat mengganggu pekerjaan orang-orang pada bagian yang sedang diamati

Berikut ini adalah beberapa petunjuk untuk melakukan observasi

- Tentukan hal-hal apa saja yang akan diobservasi agar kegiatan observasi menghasilkan sesuai dengan yang diharapkan

- Mintalah ijin kepada orang yang berwenang pada bagian yang akan diobservasi
- Berusaha sesedikit mungkin agar tidak mengganggu pekerjaan orang lain
- Jika ada yang anda tidak mengerti, cobalah bertanya. Jangan membuat asumsi sendiri

2.2.3 Teknik Kuisisioner

Pengumpulan data dengan menggunakan kuisisioner mempunyai keuntungan yaitu:

- Hasilnya lebih objektif, karena kuisisioner dapat dilakukan kepada banyak orang sekaligus
- Waktunya lebih singkat

Sedangkan kelemahan pengumpulan data dengan menggunakan kuisisioner adalah sebagai berikut

- Responden cenderung malas untuk mengisi kuisisioner
- Sulit untuk membuat pertanyaan yang singkat, jelas, dan mudah dipahami

Berikut ini adalah beberapa cara yang dapat dilaksanakan untuk membuat teknik kuisisioner menghasilkan data yang baik

- Hindari pertanyaan isian, lebih baik pilihan ganda, karena responden biasanya malas untuk menulis banyak, dan jika responden menuliskan sesuatu sering kali susah untuk dipahami. Dan juga dengan pertanyaan pilihan ganda, akan memudahkan Anda untuk melakukan rekapitulasi data hasil kuisisioner.
- Buatlah pertanyaan yang tidak terlalu banyak
- Buatlah pertanyaan yang singkat, pada, dan jelas

2.3 Jenis Kebutuhan

Kebutuhan (requirement) yang dikumpulkan dengan menggunakan wawancara, observasi, kuisisioner, atau gabungan dari ketiga hal tersebut dapat dikelompokkan menjadi beberapa kategori sebagai berikut (tidak semua kebutuhan ini harus ada).

- *Functional requirement*
Kebutuhan yang terkait dengan fungsi produk, misalInnya sistem informasi harus mampu mencetak laporan, sistem informasi harus mampu menampilkan grafik, dan lain-lain.
- *Development requirement*
Kebutuhan yang terkait tools untuk pengembangan sistem informasi baik perangkat keras maupun perangkat lunak misalnya sistem informasi dikembangkan dengan menggunakan alat bantu Eclipse untuk pengembangan dan StarUML untuk pemodelan.
- *Deployment requirement*

Kebutuhan terkait dengan lingkungan di mana sistem informasi akan digunakan baik perangkat lunak maupun perangkat keras. Contoh kebutuhan ini misalnya sistem informasi harus mampu berjalan pada server dengan spesifikasi perangkat keras memory 4GB DDR3, processor Intel Xeon Quad Core, dan spesifikasi sistem operasi Ubuntu Server 9.

- *Performance requirement*
Kebutuhan yang terkait dengan ukuran kualitas maupun kuantitas, khususnya terkait dengan kecepatan, skalabilitas dan kapasitas. Misalnya sistem informasi tersebut harus mampu diakses oleh minimal 1000 orang pada waktu yang bersamaan.
- *Documentation requirement*
Kebutuhan ini terkait dengan dokumen apa saja yang akan disertakan pada produk akhir. Dokumen yang biasanya dihasilkan pada tahap akhir pengembangan sistem informasi antara lain dokumen teknis (mulai dari dokumen perencanaan, proyek, analisis, desain, sampai pengujian), user manual, dan dokumen pelatihan.
- *Support requirement*
Kebutuhan yang terkait dukungan yang diberikan setelah sistem informasi digunakan. Dukungan teknis tersebut misalnya adanya pelatihan bagi calon pengguna.
- *Miscellaneous requirement*
Kebutuhan ini adalah kebutuhan-kebutuhan tambahan lainnya yang belum tercakup pada beberapa kategori kebutuhan yang telah terdefinisi di atas.

2.4 Definisi Desain Sistem

Desain atau perancangan dalam pembangunan perangkat lunak merupakan upaya untuk mengonstruksi sebuah sistem yang memberikan kepuasan (mungkin informal) akan spesifikasi kebutuhan fungsional, memenuhi target, memenuhi kebutuhan secara implisit atau eksplisit dari segi performansi maupun penggunaan sumber daya, kepuasan batasan pada proses desain dari segi biaya, waktu, dan perangkat. Kualitas perangkat lunak biasanya dinilai dari segi kepuasan penggunaan perangkat lunak terhadap perangkat lunak yang digunakan.

Analisis dan desain sistem akan dijelaskan lebih lengkap pada bab-bab selanjutnya, mulai dari analisis dan desain basis data, pemrograman terstruktur, maupun pemrograman berorientasi objek.

Soal Latihan

1. Apakah yang dimaksud dengan analisis sistem?
2. Kegiatan apa saja yang dilakukan pada saat analisis sistem?
3. Sebutkan dan jelaskan macam-macam teknik pengumpulan data
4. Sebutkan jenis-jenis kebutuhan pengembangan sistem informasi
5. Apakah yang dimaksud dengan dokumen SRS?
6. Apa saja isi dokumen SRS?
7. Buatlah sebuah dokumen SRS untuk pengembangan sistem informasi apotek
8. Apakah yang dimaksud dengan desain sistem?
9. Hal apa saja yang dilakukan pada tahap desain sistem?
10. Sebutkan dan jelaskan karakteristik apa saja yang terdapat pada sistem berorientasi objek
11. Mengapa berkembang metodologi berorientasi objek?
12. Apa yang dimaksud dengan kelas dan objek? Gambarkan keterhubungan antara kelas dan objek
13. Apa yang dimaksud dengan pendekatan terstruktur?
14. Apa perbedaan pendekatan terstruktur dengan pendekatan berorientasi objek?
15. Sebutkan minimal 2 metodologi lain (selain pendekatan terstruktur dan pendekatan berorientasi objek). Sebutkan karakteristiknya dan kemudahan carilah kelebihan dan kekurangannya jika dibandingkan dengan pendekatan terstruktur dan pendekatan berorientasi objek?

BAB 3 SDLC

3.1 Pengertian SDLC

Pada awal pengembangan perangkat lunak, para pembuat program (*programmer*) langsung melakukan pengodean perangkat lunak tanpa menggunakan prosedur atau tahapan pengembangan perangkat lunak. Dan ditemuilah kendala-kendala seiring dengan perkembangan skala sistem-sistem perangkat yang semakin besar.

SDLC dimulai dari tahun 1960-an, untuk mengembangkan sistem skala usaha besar secara fungsional untuk para konglomerat pada jaman itu. Sistem-sistem yang dibangun mengelola informasi kegiatan dan rutinitas dari perusahaan-perusahaan yang berpotensi memiliki data yang besar dalam perkembangannya.

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

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

- Inisiasi (*initiation*)
Tahap ini biasanya ditandai dengan pembuatan proposal proyek perangkat lunak
- Pengembangan konsep sistem (*system concept development*) mendefinisikan lingkungan konsep termasuk dokumen lingkup sistem, analisis manfaat biaya, manajemen rencana, dan pembelajaran kemudahan sistem.
- Perencanaan (*planning*)
Mengembangkan rencana manajemen proyek dan dokumentasi perencanaan lainnya. Menyediakan dasar untuk mendapatkan sumber daya (*resources*) yang dibutuhkan untuk mempermudah solusi.
- Analisis kebutuhan (*requirements analysis*)
Menganalisis kebutuhan pemakai sistem perangkat lunak (*user*) dan mengembangkan kebutuhan user. Membuat dokumen kebutuhan fungsional.
- Desain (*desain*)

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

- Pengembangan (*Development*)
Mengonversikan desain ke sistem informasi yang lengkap termasuk bagaimana memperoleh dan melakukan instalasi lingkungan sistem yang dibutuhkan; membuat basis data dan mempersiapkan prosedur kasus pengujian; mempersiapkan berkas atau file pengujian, pengodean, pengompilasian, memperbaiki dan membersihkan program; peninjauan pengujian.
- Integrasi dan pengujian (*integration and test*)
Mendemonstrasikan sistem perangkat lunak bahwa telah memenuhi kebutuhan yang dispesifikasikan pada dokumen kebutuhan fungsional. Dengan diarahkan oleh staf penjamin kualitas (*quality assurance*) dan user. Menghasilkan laporan analisis pengujian.
- Implementasi (*implementation*)
Termasuk pada persiapan implementasi, implementasi perangkat lunak pada lingkungan produksi (lingkungan pada user) dan menjalankan resolusi dari permasalahan yang teridentifikasi dari fase integrasi dan pengujian.
- Operasi dan pemeliharaan (*operations and maintenance*)
Mendeskripsikan pekerjaan untuk mengoperasikan dan memelihara sistem informasi pada lingkungan produksi (lingkungan pada user), termasuk implementasi akhir dan masuk pada proses peninjauan.
- Disposisi (*disposition*)
Mendeskripsikan aktivitas akhir dari pengembangan sistem dan membangun data yang sebenarnya sesuai dengan aktivitas user.

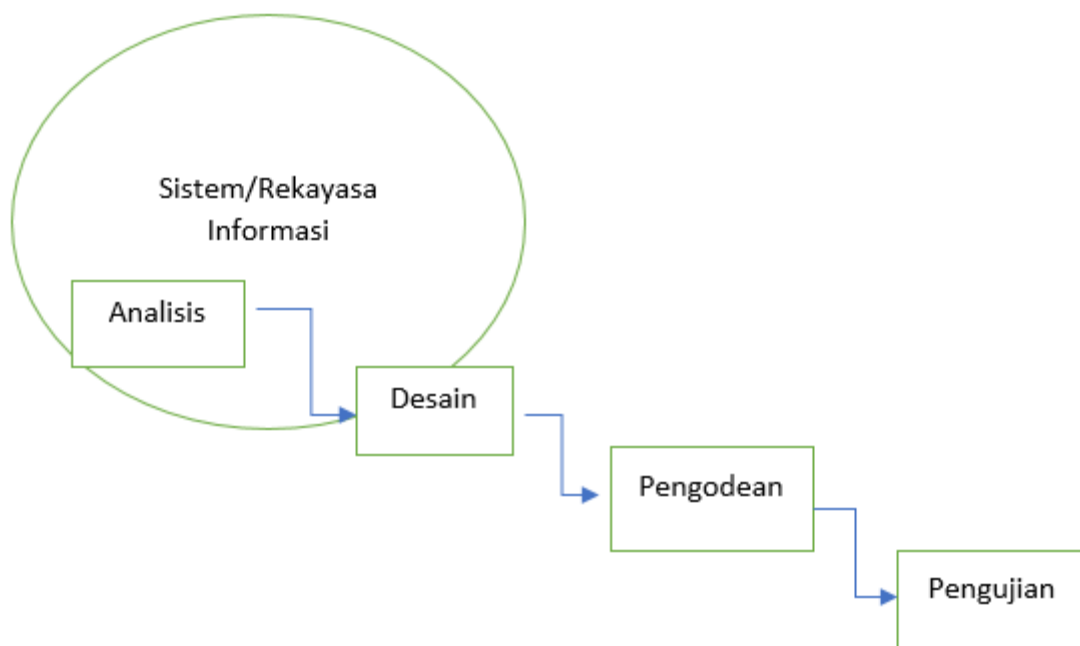
Analisis dan Desain sering dikelompokkan sebagai proses sistem/rekayasa informasi karena pada tahapan inilah informasi mengenai kebutuhan perangkat lunak banyak dikumpulkan dan diintegrasikan. Ada beberapa model SDLC yang dapat digunakan. Semuanya memiliki kelemahan dan kelebihan pada setiap model SDLC. Hal terpenting adalah mengenali tipe pelanggan (*customer*) dan memilih menggunakan model SDLC yang sesuai dengan karakter pelanggan (*customer*) dan sesuai dengan karakter pengembang.

3.2 Model SDLC

SDLC memiliki beberapa model dalam penerapan tahapan prosesnya. Beberapa model dasar akan dibahas pada subbab-subab berikutnya. Selain model-model dasar yang akan dibahas, masih banyak model-model yang muncul dengan memodifikasi model-model SDLC dasar.

3.2.1 Model *WaterFall*

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



gambar 6. Ilustrasi model waterfall

- Analisis kebutuhan perangkat lunak
Proses pengumpulan kebutuhan dilakukan secara intensif untuk menspesifikasikan kebutuhan perangkat lunak agar dapat dipahami perangkat lunak seperti apa yang dibutuhkan oleh user. Spesifikasi kebutuhan perangkat lunak pada tahap ini perlu untuk didokumentasikan.
- Desain
Desain perangkat lunak adalah proses multi langkah yang fokus pada desain pembuatan program perangkat lunak termasuk struktur data, arsitektur perangkat lunak. Representasi antarmuka, dan prosedur pengodean. Tahap ini mentransiasi kebutuhan perangkat lunak dari tahap analisis kebutuhan ke representasi desain agar dapat diimplementasikan menjadi program pada tahap selanjutnya. Desain perangkat lunak yang dihasilkan pada tahap ini juga perlu didokumentasikan.

- Pembuatan kode program
Desain harus ditranslasikan ke dalam program perangkat lunak. Hasil dari tahap ini adalah program komputer sesuai dengan desain yang telah dibuat pada tahap desain.
- Pengujian
Pengujian fokus pada perangkat lunak secara dari segi logik dan fungsional dan memastikan bahwa semua bagian sudah diuji. Hal ini dilakukan untuk meminimalisir kesalahan (*error*) dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan.
- Pendukung (*support*) atau pemeliharaan (*maintenance*)
Tidak menutup kemungkinan sebuah perangkat lunak mengalami perubahan ketika sudah dikirimkan ke user. Perubahan bisa terjadi karena adanya kesalahan yang muncul dan tidak terdeteksi saat pengujian atau perangkat lunak harus beradaptasi dengan lingkungan baru. Tahap pendukung atau pemeliharaan dapat mengulangi proses pengembangan mulai dari analisis spesifikasi untuk perubahan perangkat lunak yang sudah ada, tetapi tidak untuk membuat perangkat lunak baru.

Dari kenyataan yang terjadi sangat jarang model air terjun dapat dilakukan sesuai alurnya karena sebab berikut:

- Perubahan spesifikasi perangkat lunak terjadi di tengah alur pengembangan.
- Sangat sulit bagi pelanggan untuk mendefinisikan semua spesifikasi di awal alur pengembangan. Pelanggan sering kali butuh contoh (*prototype*) untuk menjabarkan spesifikasi kebutuhan sistem lebih lanjut.
- Pelanggan tidak mungkin bersabar mengakomodasi perubahan yang diperlukan diakhir alur pengembangan

Dengan berbagai kelemahan yang dimiliki model air terjun tapi model ini telah menjadi dasar dari model-model yang lain dalam melakukan perbaikan model pengembangan perangkat lunak.

Model air terjun sangat cocok digunakan kebutuhan pelanggan sudah sangat dipahami dan kemungkinan terjadinya perubahan kebutuhan selama pengembangan perangkat lunak kecil. Hal positif dari model air terjun adalah struktur tahap pengembangan sistem jelas, dokumentasi dihasilkan di setiap tahap pengembangan, dan sebuah tahap dijalankan setelah tahap sebelumnya selesai dijalankan (tidak ada tumpang tindih pelaksanaan tahap).

- ❖ Model waterfall adalah model SDLC yang paling sederhana. Model ini hanya cocok untuk pengembangan perangkat lunak dengan spesifikasi yang tidak berubah-ubah

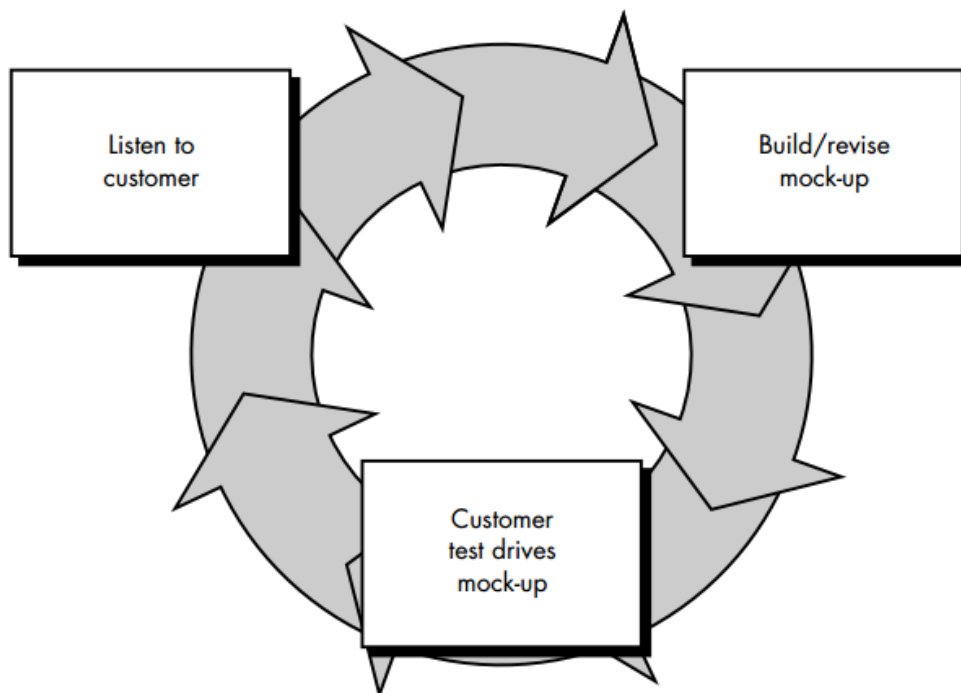
3.2.2 Model Prototipe

Sering pelanggan (*customer*) membayangkan kumpulan kebutuhan yang diinginkan tapi tidak terspesifikasikan secara detail dari segi masukan (*input*), proses, maupun keluaran (*output*). Di sisi lain seorang pengembang perangkat lunak harus menspesifikasikan sebuah kebutuhan secara detail dari segi teknis dimana pelanggan sering kurang mengerti mengenai hal teknis ini.

Model prototipe dapat digunakan untuk menyambungkan ketidakpahaman pelanggan mengenai hal teknis dan memperjelas spesifikasi kebutuhan yang diinginkan pelanggan kepada pengembang perangkat lunak.

Model prototipe (*prototyping model*) dimulai dari mengumpulkan kebutuhan pelanggan terhadap perangkat lunak yang akan dibuat. Lalu dibuatlah program prototipe agar pelanggan lebih terbayang dengan apa yang sebenarnya diinginkan. Program prototipe biasanya merupakan program yang belum jadi. Program ini biasanya menyediakan tampilan dengan simulasi alur perangkat lunak sehingga tampak seperti perangkat lunak atau yang sudah jadi. Program prototipe ini dievaluasi oleh pelanggan atau user sampai ditemukan spesifikasi yang sesuai dengan keinginan pelanggan atau user.

Berikut ini adalah gambar dari model prototipe:



gambar 7. Ilustrasi model prototipe

Mock-up adalah sesuatu yang digunakan sebagai model desain yang digunakan untuk mengajar. Mendemostrasi, evaluasi desain, promosi dan keperluan lain. Sebuah mock-up disebut sebagai prototipe perangkat lunak jika menyediakan atau mampu mendemostrasikan sebagian besar fungsi sistem perangkat lunak dan memungkinkan pengujian desain sistem perangkat lunak. Iterasi terjadi pada pembuatan prototipe sampai sesuai dengan keinginan pelanggan (*customer*) atau user.

Seiring dengan mengembangkan prototipe maka sistem perangkat lunak yang sebenarnya dikembangkan juga sehingga sesuai dengan kebutuhan pelanggan (*customer*) atau user.

Model prototipe juga memiliki kelemahan sebagai berikut:

- Pelanggan dapat sering mengubah-ubah atau menambah tambah spesifikasi kebutuhan karena menganggap aplikasi sudah dengan cepat dikembangkan, karena adanya iterasi ini dapat menyebabkan pengembang banyak mengalah dengan pelanggan karena perubahan atau penambahan spesifikasi kebutuhan perangkat lunak.
- Pengembang lebih sering mengambil kompromi dengan pelanggan untuk mendapatkan prototipe dengan waktu yang cepat sehingga pengembang lebih sering melakukan segala cara (tanpa idealis) guna menghasilkan prototipe untuk didemonstrasikan. Hal ini dapat menyebabkan kualitas perangkat lunak yang kurang baik atau bahkan menyebabkan iteratif tanpa akhir.

Permasalahan dapat terjadi pada model prototipe, hal ini dapat diatasi dengan melakukan perjanjian antara pengembang perangkat lunak dengan pelanggan (*customer*) atau user agar model prototipe hanya digunakan untuk mendefinisikan spesifikasi kebutuhan perangkat lunak, tapi tidak untuk seluruh proses pengembangan seluruh sistem perangkat lunak.

Model prototipe cocok digunakan untuk menjabarkan kebutuhan pelanggan secara lebih detail karena pelanggan sering kali kesulitan menyampaikan kebutuhannya secara detail tanpa melihat gambaran yang jelas. Untuk mengantisipasi agar proyek dapat berjalan sesuai dengan target waktu dan biaya di awal, maka sebaiknya spesifikasi kebutuhan sistem harus sudah disepakati oleh pengembang dengan pelanggan secara tertulis. Dokumen tersebut akan menjadi patokan agar spesifikasi kebutuhan sistem masih dalam ruang lingkup proyek.

Model prototipe kurang cocok untuk aplikasi dengan skala besar karena membuat prototipe untuk aplikasi skala besar akan sangat memakan waktu dan tenaga.

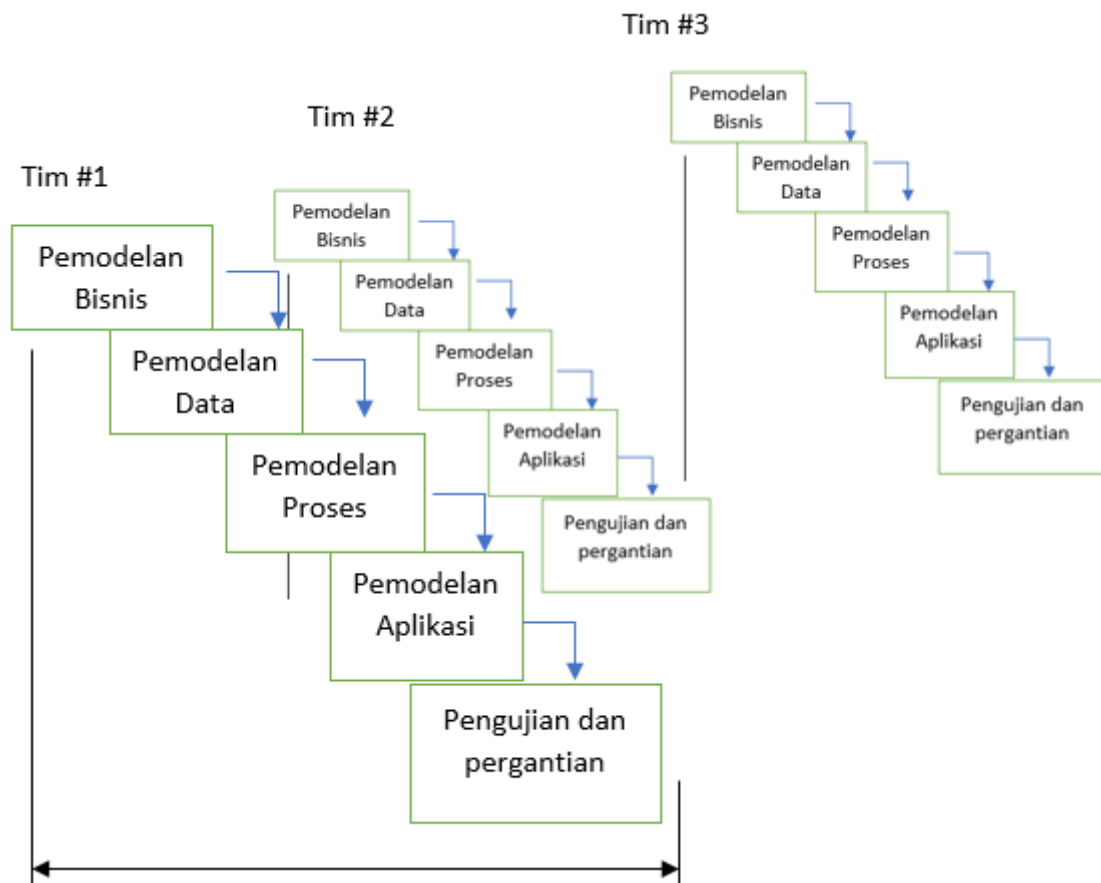
- ❖ Model Prototipe cocok digunakan untuk menggali spesifikasi kebutuhan pelanggan secara lebih detail tetapi beresiko tinggi terhadap membengkaknya biaya dan waktu proyek.

3.2.3 Model Rapid Application Development (RAD)

Rapid Application Development (RAD) adalah model proses pengembangan perangkat lunak yang bersifat inkremental terutama untuk waktu pengerjaan yang pendek. Model RAD adalah

adaptasi dari model air terjun versi kecepatan tinggi dengan menggunakan model air terjun untuk pengembangan setiap komponen perangkat lunak.

Jika kebutuhan perangkat lunak dipahami dengan baik dan lingkungan perangkat lunak dibatasi dengan baik sehingga tim dapat menyelesaikan pembuatan perangkat lunak dengan waktu yang pendek. Model RAD membagi tim pengembang menjadi beberapa tim untuk mengerjakan beberapa komponen masing-masing tim pengerjaan dapat dilakukan secara paralel. Berikut adalah gambar model RAD.

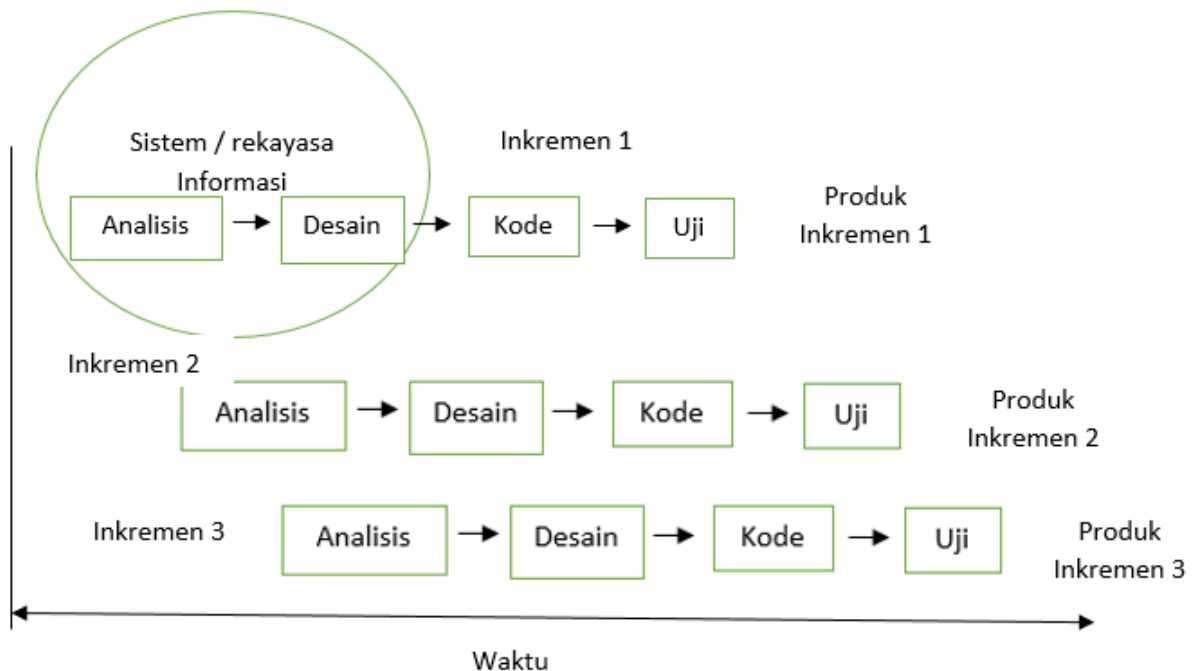


gambar 8. Ilustrasi model RAD

- Pemodelan bisnis
Pemodelan yang dilakukan untuk memodelkan fungsi bisnis untuk mengetahui informasi apa saja yang terkait bisnis, informasi apa yang harus dibuat, siapa yang harus membuat informasi itu, bagaimana alur informasi itu, proses apa saja yang terkait informasi itu.

3.2.4 Model Iteratif

Model iteratif (iterative model) mengkombinasikan proses-proses pada model air terjun dan iteratif pada model prototipe. Model inkremental akan menghasilkan versi-versi perangkat lunak yang sudah mengalami penambahan fungsi untuk setiap pertambahan (inkremental/incremental). Berikut adalah gambar dari model inkremental:



gambar 9. Ilustrasi model iteratif

Model inkremental dibuat untuk mengatasi kelemahan dari model air terjun yang tidak mengakomodasi iterasi, dan mengatasi kelemahan pada metode prototipe yang memiliki proses terlalu pendek dan setiap iteratif prosesnya tidak selalu menghasilkan produk (bisa jadi hanya prototipe). Model inkremental menghasilkan produk / aplikasi untuk setiap tahapan inkremen.

Model inkremental sangat cocok digunakan jika staf yang dimiliki memiliki pergantian (*turnover*) yang tinggi sehingga staf tidak dapat terus ikut dalam pengembangan perangkat lunak. Mekanisme tahapan inkremental perlu direncanakan terlebih dahulu agar hasil produk dan pengerjaan setiap tahapan inkremen menjadi lebih baik.

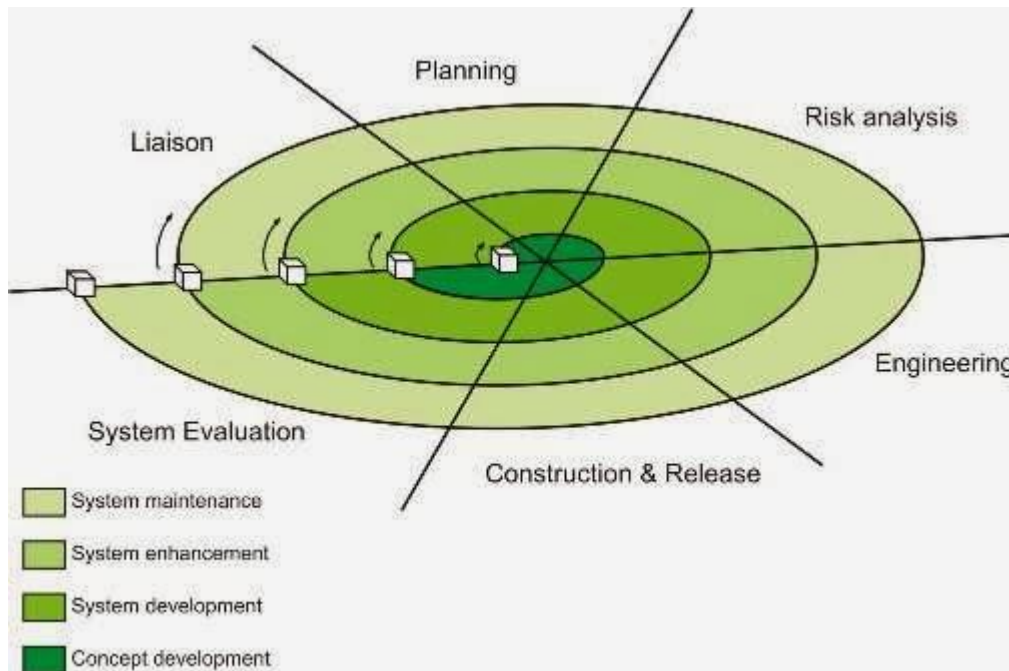
- ❖ Model Iteratif merupakan gabungan dari model waterfall dan model prototipe. Model ini cocok digunakan pengembang dengan turnover staff yang tinggi.

3.2.5 Model Spiral

Model spiral (*spiral model*) memasangkan iteratif pada model prototipe dengan kontrol dan aspek sistematis yang diambil dari model air terjun. Model Spiral menyediakan pengembangan dengan cara cepat dengan perangkat lunak yang memiliki versi yang terus bertambah fungsinya (*increment*).

Pada iterasi awal maka yang dihasilkan adalah prototipe sedangkan pada iterasi akhir yang dihasilkan adalah perangkat lunak yang sudah lengkap. Model spiral dibagi menjadi beberapa kerangka aktifitas atau disebut juga wilayah kerja (*task region*). Banyaknya wilayah kerja biasanya diantara tiga sampai enam wilayah sebagai berikut:

- Komunikasi dengan pelanggan (*customer communication*)
Aktifitas ini diperlukan untuk membantu komunikasi yang efektif antara pengembang (*developer*) dan pelanggan (*customer*).
- Perencanaan (*planning*)
Aktifitas ini diperlukan untuk mendefinisikan sumber daya, waktu, dan informasi yang terkait dengan proyek.
- Analisis risiko (*risk analysis*)
Aktifitas ini diperlukan untuk memperkirakan risiko dari segi teknis maupun manajemen.
- Rekayasa (*engineering*)
Aktifitas ini diperlukan untuk membangun satu atau lebih representasi dari aplikasi perangkat lunak (dapat juga berupa prototipe).
- Konstruksi dan peluncuran (*construction and release*)
Aktifitas ini dibutuhkan untuk mengkonstruksi, menguji, melakukan instalasi, dan menyediakan dukungan terhadap user (misalnya dari segi dokumentasi dan pelatihan).
- Evaluasi pelanggan (*customer evaluation*)
Aktifitas ini dibutuhkan untuk mendapatkan umpan balik berdasarkan evaluasi representasi perangkat lunak yang dihasilkan dari proses rekayasa dan diimplementasi pada tahap instalasi.



gambar 10. Ilustrasi model spiral

Setiap wilayah kerja terdiri dari kumpulan pekerjaan (*task set*) yang tergantung pada karakteristik proyek yang sedang dikerjakan. Semakin besar suatu proyek maka kumpulan pekerjaan di dalam setiap wilayah kerja juga semakin banyak.

Model spiral dilakukan searah dengan jarum jam dimulai dari sumbu proyek. Sumbu proyek dapat digunakan sebagai awal iterasi ataupun evaluasi dari iterasi yang sudah dilakukan. Pada gambar di atas setiap wilayah kerja dibedakan dengan warna wilayah, dimana setiap wilayah berputar dengan urutan kerja tertentu.

Pada model spiral, hasil akhir dan evaluasi dari sebuah wilayah kerja akan menjadi inisiasi dari wilayah kerja berikutnya. Model spiral cocok digunakan untuk mengembangkan sistem perangkat lunak berskala besar karena memiliki proses analisis risiko yang dapat sangat meminimalisir risiko yang mungkin terjadi. Model spiral memungkinkan pengembang untuk menggunakan prototipe pada setiap tahap untuk mengurangi risiko.

Dari beberapa model SDLC yang sudah disebutkan sebelumnya, model spiral merupakan model yang bisa memberikan jaminan kualitas yang paling baik untuk aplikasi berskala besar. Penerapan model spiral cocok digunakan untuk suatu proyek dengan target waktu dan biaya yang tidak terlalu ketat. Setiap perubahan spesifikasi pasti berisiko pada molornya waktu pengerjaan dan membengkaknya biaya proyek.

- ❖ Model spiral cocok digunakan untuk mengembangkan aplikasi dengan skala besar tetapi target waktu dan biaya tidak terlalu mengikat.

Soal Latihan

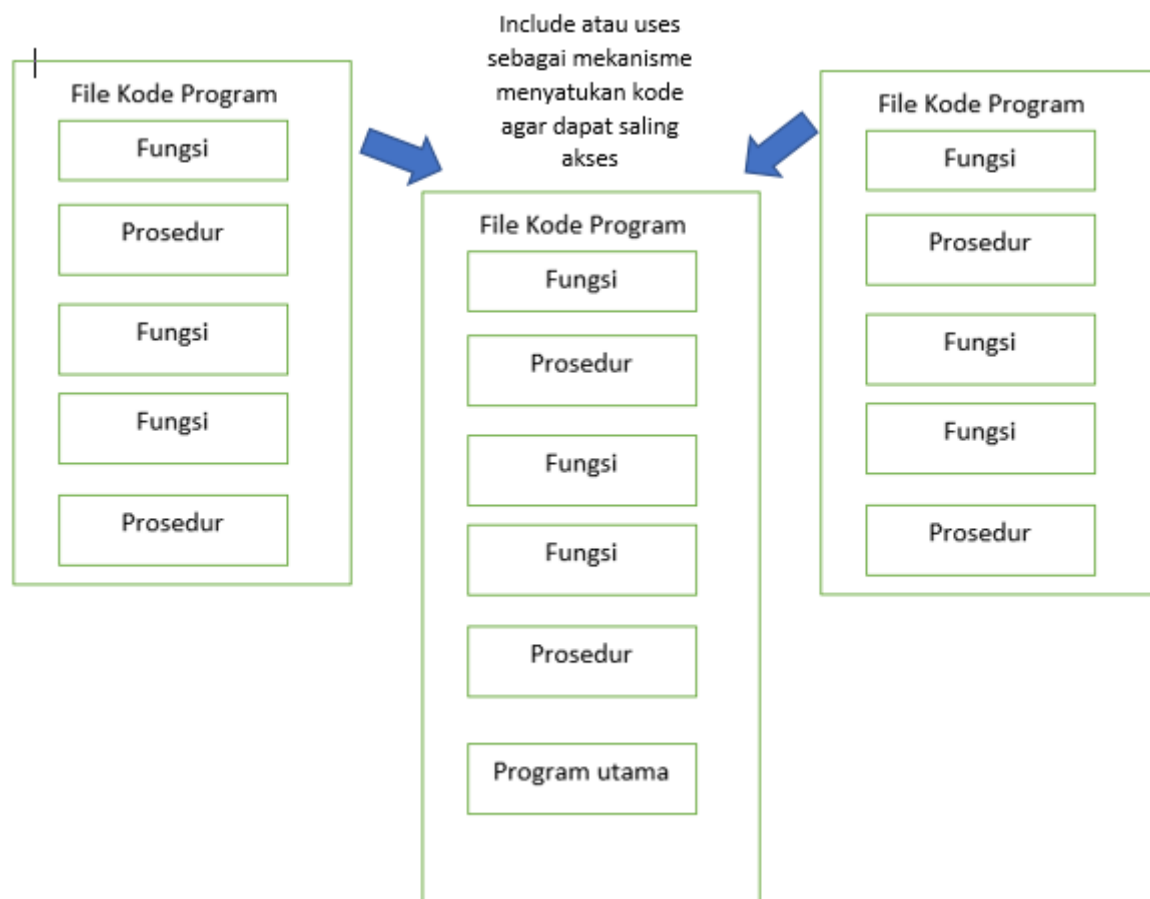
1. Jelaskan Pengertian SDLC dan sebutkan tahapan-tahapannya secara umum
2. Apa kelebihan dan kekurangan masing-masing model sebagai berikut:
 - a. Model waterfall
 - b. Model prototipe
 - c. Model RAD
 - d. Model iteratif
 - e. Model spiral
3. Apa risiko yang dihadapi jika pengembangan aplikasi (rekayasa perangkat lunak) tidak mengikuti tahapan-tahapan SDLC?
4. Sebutkan alasan munculnya SDLC?
5. Mengapa model waterfall dianggap sebagai model SDLC yang paling sederhana dan hanya cocok digunakan untuk aplikasi skala kecil?
6. Mengapa ketika menerapkan metode prototipe, pada tahap awal harus benar-benar diperjelas batasan-batasan / ruang lingkup / spesifikasi perangkat lunak secara umum?
7. Mengapa metode RAD bisa memberikan hasil yang lebih cepat dibandingkan metode waterfall?
8. Mengapa metode iteratif cocok digunakan untuk pengembang dengan turnover staf yang tinggi?

BAB 4 PEMOGRAMAN TERSTRUKTUR

4.1 Pengertian Pemograman Terstruktur

Pemograman terstruktur adalah konsep atau paradigma atau sudut pandang pemograman yang membagi-bagi program berdasarkan fungsi-fungsi atau prosedur-prosedur yang dibutuhkan program komputer. Modul-modul (pembagian program) biasanya dibuat dengan mengelompokkan fungsi-fungsi dan prosedur-prosedur yang diperlukan sebuah proses tertentu.

Fungsi-fungsi dan prosedur-prosedur ditulis secara sekuensial atau terurut dari atas ke bawah sesuai dengan kebergantungan antar fungsi atau prosedur (fungsi atau prosedur yang dapat dipakai oleh fungsi atau prosedur dibawahnya harus yang sudah ditulis atau dideklarasikan di atasnya). Berikut adalah contoh ilustrasi pemograman terstruktur:



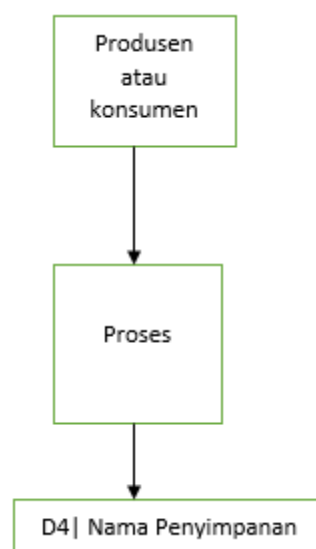
gambar 11. Ilustrasi pemograman terstruktur

Permodulan pada pemograman terstruktur dibagi berdasarkan fungsi-fungsi dan prosedur-prosedur. Oleh karena itu, pemodelan pada pemograman terstruktur lebih fokus bagaimana memodelkan data dan fungsi-fungsi atau prosedur-prosedur yang harus dibuat. Jenis paradigma pemograman yang digunakan dapat dideteksi dari bahasa pemograman apa yang

akan digunakan untuk membuat program, baru setelah itu ditentukan paradigma pemrograman apa yang akan digunakan.

4.2 DFD

Data flow Diagram (DFD) awalnya dikembangkan oleh Chris Gane dan Trish Sarson pada tahun 1979 yang termasuk dalam *Structured Systems Analysis and Design Methodology* (SSADM) yang ditulis oleh Cris Gane dan Trish Sarson. Sistem yang dikembangkan ini berbasis pada *dekomposisi fungsional* dari sebuah sistem. Berikut adalah contoh DFD yang dikembangkan oleh Chris Gane dan Trish Sarson:



gambar 12. Contoh DFD yang dikembangkan Chris Gane & Trish Sarson

Edward yourdon dan Tom DeMarco memperkenalkan metode yang lain pada tahun 1980-an di mana mengubah persegi dengan sudut lengkung (pada DFD Chris Gane dan Trish Sarson) dengan lingkaran untuk menotasikan. DFD Edward Yourdon dan Tom DeMarco populer digunakan sebagai model analisis sistem perangkat lunak untuk sistem perangkat lunak yang akan diimplementasikan dengan pemrograman terstruktur.

Informasi yang ada di dalam perangkat lunak dimodifikasikan dengan beberapa transformasi yang dibutuhkan. *Data Flow Diagram* (DFD) atau dalam bahasa Indonesia menjadi Diagram Alir Data (DAD) adalah representasi grafik yang menggambarkan aliran informasi dan transformasi informasi yang diaplikasikan sebagai data yang mengalir dari masuk (*input*) dan keluaran (*output*).

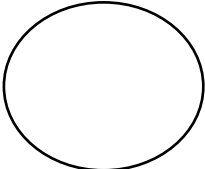
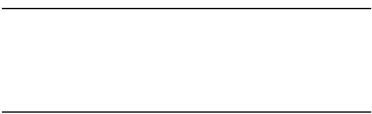

DFD dapat digunakan untuk merepresentasikan sebuah sistem atau perangkat lunak pada beberapa level abstraksi. DFD dapat dibagi menjadi beberapa level yang detail untuk merepresentasikan aliran informasi atau fungsi yang lebih detail. DFD menyediakan

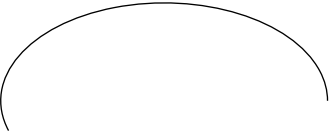
mekanisme untuk pemodelan fungsional ataupun pemodelan aliran informasi. Oleh karena itu, DFD lebih sesuai digunakan untuk memodelkan fungsi-fungsi perangkat lunak yang akan diimplementasikan menggunakan pemrograman terstruktur karena pemrograman terstruktur membagi-bagi bagiannya dengan fungsi-fungsi dan prosedur-prosedur.

DFD tidak sesuai untuk memodelkan sistem perangkat lunak yang akan dibangun menggunakan pemrograman berorientasi objek paradigma pemrograman terstruktur dan pemrograman berorientasi objek merupakan hal yang berbeda. Jangan mencampuradukkan pemrograman terstruktur dan pemrograman berorientasi objek.

- ❖ DFD tidak sesuai untuk memodelkan sistem yang menggunakan pemrograman berorientasi objek.

Notasi-notasi pada DFD (Edward Yourdon dan Tom DeMarco) adalah sebagai berikut:

Notasi	Keterangan
	<p>Proses atau fungsi prosedur; pada pemodelan perangkat lunak yang akan diimplementasikan dengan pemrograman terstruktur, maka pemodelan notasi inilah yang harusnya menjadi fungsi atau prosedur di dalam kode program.</p> <p>Catatan: Nama yang diberikan pada sebuah proses biasanya berupa kata kerja.</p>
	<p>File atau basisdata atau penyimpanan (<i>storage</i>); pada pemodelan perangkat lunak yang akan diimplementasikan dengan pemrograman terstruktur, maka pemodelan notasi inilah yang harusnya dibuat menjadi tabel-tabel basis data yang dibutuhkan, tabel-tabel ini juga harus sesuai dengan perancangan tabel-tabel pada basis data (<i>Entity Relationship Diagram (ERD)</i>, <i>Conceptual Data Model (CDM)</i>, <i>Physical Data Model (PDM)</i>)</p> <p>Catatan: Nama yang diberikan pada sebuah penyimpanan biasanya kata benda</p>
	<p>Entitas luar (<i>external entity</i>) atau masukan (<i>input</i>) atau keluaran (<i>output</i>) atau orang yang memakai/berinteraksi dengan perangkat lunak yang dimodelkan atau sistem lain yang terkait dengan aliran data dari sistem yang dimodelkan</p>

	<p>Catatan: Nama yang digunakan pada masukan (<i>input</i>) atau keluaran (<i>output</i>) biasanya berupa kata benda</p>
	<p>Aliran data; merupakan data yang dikirim antara proses, dari penyimpanan ke proses, atau dari proses ke masukan (<i>input</i>) atau keluaran (<i>output</i>)</p> <p>Catatan: Nama yang digunakan pada aliran data biasanya berupa kata benda, dapat diawali dengan kata data misalnya "data siswa: atau tanpa kata data misalnya "siswa"</p>

Berikut ini adalah tahapan-tahapan perancangan dengan menggunakan DFD:

1. Membuat DFD level 0 atau sering disebut juga *Context Diagram*
DFD level 0 menggambarkan sistem yang akan dibuat sebagai suatu entitas tunggal yang berinteraksi dengan orang maupun sistem lain. DFD level 0 digunakan untuk menggambarkan interaksi antara sistem yang akan dikembangkan dengan entitas luar.
2. Membuat DFD level 1
DFD level 1 digunakan untuk menggambarkan modul-modul yang ada dalam sistem yang akan dikembangkan. DFD level 1 merupakan hasil *breakdown* DFD level 0 yang sebelumnya sudah dibuat.
3. Membuat DFD level 2
Modul-modul pada DFD level 1 dapat di-breakdown menjadi DFD level 2. Modul mana saja yang harus di-breakdown lebih detail tergantung pada tingkat kedetailan modul tersebut. Apabila modul tersebut sudah cukup detail dan rinci maka modul tersebut sudah tidak perlu untuk di-breakdown lagi. Untuk sebuah sistem, jumlah DFD level 2 sama dengan jumlah modul pada DFD level 1 yang di-breakdown.
4. Membuat DFD level 3 dan seterusnya
DFD level 3,4,5 dan seterusnya merupakan *breakdown* dari modul pada DFD level di atasnya. *Breakdown* pada level 3,4,5 dan seterusnya aturannya sama persis dengan DFD level 1 atau level 2.

Pada satu diagram DFD sebaiknya jumlah modul tidak boleh lebih dari 20 buah. Jika lebih dari 20 buah modul, diagram akan terlihat rumit dan susah untuk dibaca sehingga menyebabkan sistem yang dikembangkan juga menjadi rumit.

4.3 Kamus Data

Kamus data (data dictionary) dipergunakan untuk memperjelas aliran data yang digambarkan pada DFD. Kamus data adalah kumpulan daftar elemen data yang mengalir pada sistem perangkat lunak sehingga masukan (*input*) dan keluaran (*output*) dapat dipahami secara umum (memiliki standar cara penulisan). Kamus data dalam implementasi program dapat menjadi parameter masukan atau keluaran dari sebuah fungsi atau prosedur. Kamus data biasanya berisi:

- Nama-nama dari data
- Digunakan pada-merupakan proses-proses yang terkait data
- Deskripsi-merupakan deskripsi data
- Informasi tambahan-seperti tipe data, nilai data, batas nilai data, dan komponen yang membentuk data.

Kamus data memiliki beberapa simbol untuk menjelaskan informasi tambahan sebagai berikut:

Simbol	Keterangan
=	Disusun atau terdiri dari
+	Dan
[]	Baik...atau...
{ }n	n kali diulang / bernilai banyak
()	Data opsional
* *	Batas komentar

Kamus data pada DFD nanti harus dapat dipetakan dengan hasil perancangan basis data yang dilakukan sebelumnya. Jika ada kamus data yang tidak dapat dipetakan pada tabel hasil perancangan basis data berarti hasil perancangan basis data dengan perancangan dengan DFD masih belum sesuai, sehingga harus ada yang diperbaiki baik perancangan basis datanya, perancangan DFD-nya, atau keduanya.

Soal Latihan

1. Apa yang dimaksud dengan pemograman terstruktur?
2. Apa kelebihan dan kekurangan paradigm pemrograman terstruktur?
3. Apa yang dimaksud dengan DFD (Data Flow Diagram)?
4. Apa yang dimaksud dengan Context Diagram?
5. Bagaimana tahapan melakukan perancangan sistem menggunakan DFD?
6. Apa yang dimaksud dengan Kamus Data?
7. Bagaimana keterhubungan antara kamus data dengan perancangan basis data?
8. Bisakan DFD digunakan untuk menggambarkan sistem berorientasi objek? Jelaskan

BAB 5 PEMODELAN UML

5.1 Kompleksitas Pengembangan Perangkat Lunak

Mengelola pengembangan perangkat lunak bukanlah hal yang mudah. Secara logika sama dengan mengelola banyak kepala yang memiliki tingkat pemahaman dan pemikiran yang berbeda untuk membuat sebuah benda. Semakin banyak kepala yang harus disatukan maka semakin sulit mengelolanya.

Kompeksitas sebuah perangkat lunak dapat dilihat dari hal-hal berikut:

- Kompleksitas domain atau permasalahan perangkat lunak
Pendefinisian fungsi-fungsi pada perangkat dan pendefinisian penanganan kasus-kasus yang mungkin di dalam sebuah fungsi bukanlah hal yang mudah. Sering permasalahan yang belum didefinisikan pada spesifikasi muncul begitu sebuah perangkat lunak sudah masuk ke tahap implementasi atau pengkodean. Hal seperti ini dapat menyebabkan proses kembali ke tahap analisis atau sering langsung diputuskan pada saat implementasi tanpa memperbaiki dokumen analisis perangkat lunak, secara konsep hal ini dapat menyebabkan ketidakkonsistenan antara dokumen dan perangkat lunak. Belum lagi permasalahan perbedaan interpretasi pemahaman spesifikasi oleh orang yang mengembangkan aplikasi yang terkadang tidak ditanyakan, tapi membuat asumsi sendiri.
- Kesulitan mengelola proses pengembangan perangkat lunak
Pengembangan perangkat lunak biasa dilakukan secara tim. Oleh karena itu diperlakukan koordinasi yang cukup tinggi diantara anggota tim. Jika koordinasi kurang maka kesalahpahaman interpretasi akan banyak terjadi sehingga bisa jadi perangkat lunak yang dibangun tidak pernah selesai.
- Kemungkinan fleksibilitas perubahan perangkat lunak
Sering kasus pada dunia nyata bahwa tim yang mengerjakan bukanlah orang yang memiliki kepentingan terhadap aplikasi. Sehingga tercipta hubungan klien dan developer dimana klien akan melakukan permintaan aplikasi dengan fungsi-fungsi yang dia butuhkan developer berkewajiban membuat aplikasi yang diminta oleh klien. Semakin terbatasnya pengetahuan klien mengenai teknologi informasi dan semakin banyaknya klien yang perlu dimintai pertimbangan mengenai kebutuhan aplikasi maka akan semakin tinggi kemungkinan perubahan spesifikasi di tengah proses pengembangan jika tidak ada perjanjian spesifikasi yang mengikat. Hal inilah yang sering menyebabkan pengembangan aplikasi mengalami kemoloran waktu.
- Permasalahan karakteristik bagian-bagian perangkat lunak secara diskrit
Dalam pembangunan perangkat lunak sering dilakukan dengan beberapa orang atau tim. Maka perangkat lunak akan dibagi-bagi menjadi bagian-bagian yang harus

dikerjakan orang-orang di dalam tim. Jika permasalahan bagian-bagian perangkat lunak secara diskrit tidak terdefinisi dengan benar atau dipahami berbeda oleh setiap orang yang ada di dalam tim, maka kemungkinan saat bagian-bagian ini digabungkan akan menjadi banyak kesalahan atau error. Maka dari itu koordinasi dan komunikasi yang baik di dalam sebuah tim sangat dibutuhkan.

Karena berbagai masalah dan risiko yang mungkin timbul di dalam pengembangan perangkat lunak maka perlu adanya perencanaan dan pemodelan perangkat lunak.

5.2 Pemodelan

Pemodelan adalah gambaran dari realita yang simpel dan dituangkan dalam bentuk pemetaan dengan aturan tertentu. Pemodelan dapat menggunakan bentuk yang sama dengan realitas misalnya jika seorang arsitek ingin memodelkan sebuah gedung yang akan dibangun maka dia akan memodelkannya dengan membuat sebuah maket (tiruan) arsitek gedung yang akan dibangun di mana maket itu akan dibuat semirip mungkin dengan desain gedung yang akan dibangun agar arsitektur gedung yang diinginkan dapat terlihat. Seperti yang kita ketahui bahwa manusia akan lebih memahami suatu hal dengan menggunakan visual agar sekelompok manusia yang berkepentingan dapat mengerti bagaimanakah ide yang akan dikerjakan. Pemodelan juga banyak digunakan untuk merencanakan suatu hal agar kegagalan dan risiko yang mungkin terjadi dapat diminimalisasi.

Pemodelan perangkat lunak memiliki beberapa abstraksi, misalnya sebagai berikut:

- Petunjuk yang terfokus pada proses yang dimiliki oleh sistem
- Spesifikasi struktur secara abstrak dari sebuah sistem (belum detail)
- Spesifikasi lengkap dari sebuah sistem yang sudah final
- Spesifikasi umum atau khusus sistem
- Bagian penuh atau parsial dari sebuah sistem

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

Peran perangkat pemodelan:

- Komunikasi
Perangkat pemodelan dapat digunakan sebagai alat komunikasi antara pemakai dengan analisis sistem maupun developer dalam pengembangan sistem.
- Eksperimentasi
Pengembangan sistem yang bersifat “trial and error”.
- Prediksi
Model meramalkan bagaimana suatu sistem akan bekerja.
- ❖ Salah satu perangkat pemodelan adalah Unified Modeling Language (UML)

5.3 Pengenalan UML

Pada perkembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang di berbagai negara dapat mengerti pemodelan perangkat lunak. Seperti yang kita ketahui bahwa menyatukan banyak kepala untuk menceritakan sebuah ide dengan tujuan untuk memahami hal yang sama tidaklah mudah, oleh karena itu diperlukan sebuah bahasa pemodelan perangkat lunak yang dapat dimengerti orang banyak.

Banyak orang yang telah membuat bahasa pemodelan pembangunan perangkat lunak sesuai dengan teknologi pemrograman yang berkembang pada saat itu, misalnya yang sempat berkembang dan digunakan oleh banyak pihak adalah *Data Flow Diagram* (DFD) untuk memodelkan perangkat lunak yang menggunakan pemrograman prosedural atau struktural, kemudian juga ada *State Transition Diagram* (STD) yang digunakan untuk memodelkan sistem real time (waktu nyata).

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

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

5.4 Sejarah UML

Bahasa pemrograman berorientasi objek yang pertama dikembangkan dikenal dengan nama Simula-67 yang dikembangkan pada tahun 1967. Bahasa pemrograman ini kurang berkembang dan dikembangkan lebih lanjut, namun dengan kemunculannya telah memberikan sumbangan yang besar pada developer pengembang bahasa pemrograman berorientasi objek selanjutnya.

Perkembangan aktif dari pemrograman berorientasi objek mulai menggeliat ketika berkembangnya bahasa pemrograman Smalltalk pada awal 1980-an yang kemudian diikuti dengan perkembangan bahasa pemrograman berorientasi objek yang lainnya seperti C objek, C++, Eiffel, dan CLOS. Secara aktual, penggunaan bahasa pemrograman berorientasi objek pada saat itu masih terbatas, namun telah banyak menarik perhatian di saat itu. Sekitar lima tahun setelah Smalltalk berkembang, maka berkembang pula metode pengembangan berorientasi objek. Metode yang pertama diperkenalkan oleh Sally Shlaer dan Stephen Mellor (Shlaer-Mellor, 1988) dan Peter Coad dan Edward Yourdon (Coad-Yourdon, 1991), diikuti oleh Grady Booch (Booch, 1991), James R. Rumbaugh, Michael R. Blaha, William Lorensen, Frederick Eddy, William Premerlani (Rumbaugh-Blaha-Premerlani-Eddy-Lorensen, 1991), dan masih banyak lagi.

Buku terkenal yang juga berkembang selanjutnya adalah karangan Ivar Jacobson (Jacobson, 1992) yang menerangkan perbedaan pendekatan yang fokus pada use case dan proses pengembangan. Sekitar lima tahun kemudian muncul buku yang membahas mengenai metodologi berorientasi objek yang diikuti dengan buku-buku yang lainnya. Di dalamnya juga membahas mengenai konsep, definisi, notasi, terminologi, dan proses mengenai metodologi berorientasi objek.

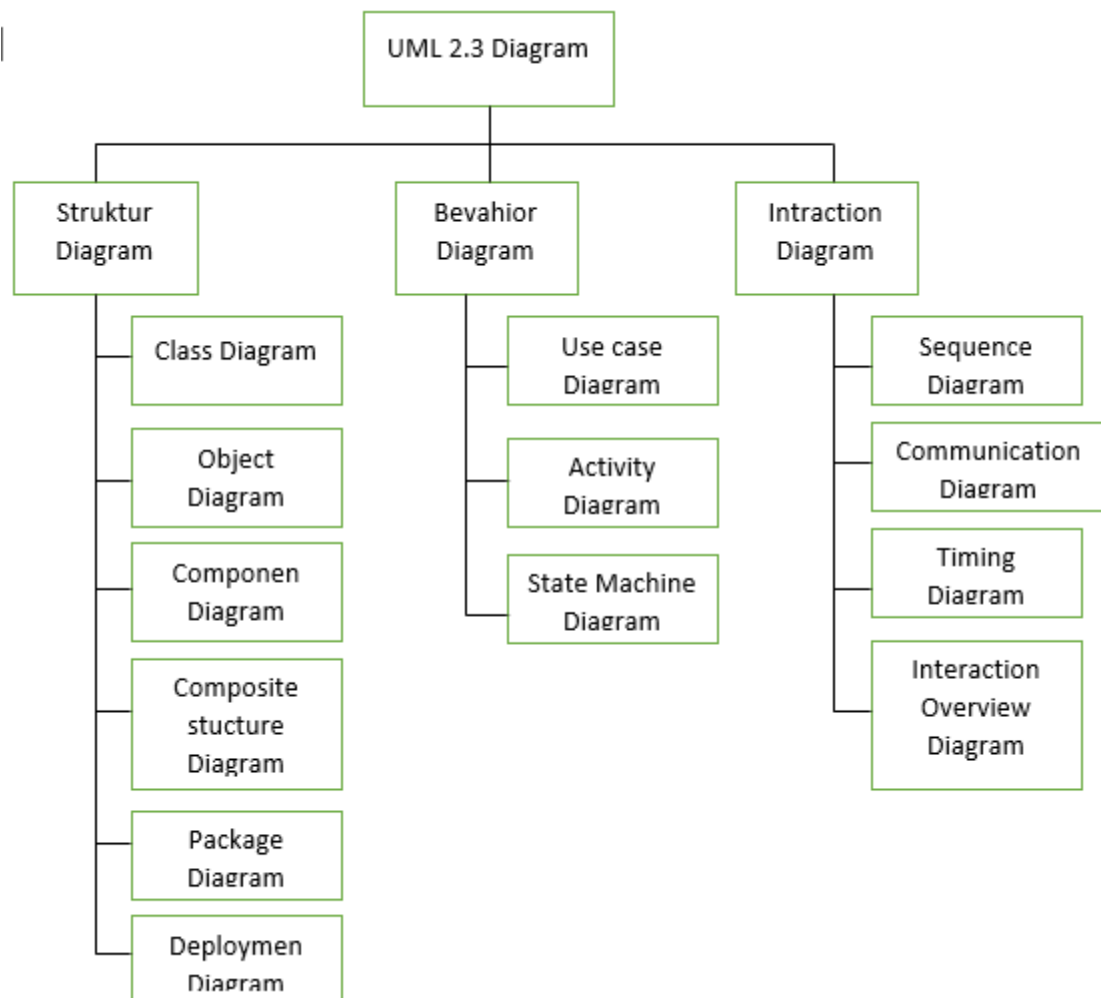
Karena banyaknya metodologi-metodologi yang berkembang pesat saat itu, maka muncullah ide untuk membuat sebuah bahasa yang dapat dimengerti semua orang. Usaha penyatuan ini banyak mengambil dari metodologi-metodologi yang berkembang saat itu. Maka dibuat bahasa yang merupakan gabungan dari beberapa konsep seperti konsep *Object Modelling Technique* (OMT) dari Rumbaugh, Collaborators (CRC) dari Rebecca Wirfs-Brock (1990), konsep pemikiran Ivar Jacobson, dan beberapa konsep lainnya dimana James R. Rumbaugh, Grady Booch, dan Ivar Jacobson bergabung dalam sebuah perusahaan yang bernama Rational *Software Corporation* menghasilkan bahasa yang disebut dengan *Unified Modelling Language* (UML).

Pada 1996, *Object Management Group* (OMG) mengajukan proposal agar adanya standarisasi pemodelan berorientasi objek dan pada bulan september 1997 UML diakomodasi oleh OMG sehingga sampai saat ini UML telah memberikan kontribusinya yang cukup besar di dalam metodologi berorientasi objek dan hal-hal yang terkait di dalamnya.

Secara fisik, UML adalah sekumpulan spesifikasi yang dikeluarkan oleh OMG, UML terbaru adalah UML 2.3 yang terdiri dari 4 macam spesifikasi, yaitu *Diagram Interchange Specification*, *UML Infrastructure*, *UML Superstructure*, dan *Object Constraint Language* (OCL). Seluruh spesifikasi tersebut dapat diakses di website omg.org

5.5 Diagram UML

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



gambar 13. Diagram UML

Berikut ini penjelasan singkat dari pembagian kategori tersebut.

- Structure diagrams yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur dari sistem yang dimodelkan
 - Behavior diagrams yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.
 - Interaction diagrams yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem.
- ❖ Perbedaan simbol UML dapat terjadi karena perbedaan perangkat lunak / CASE tools yang digunakan. CASE tools untuk membuat diagram-diagram UML misalnya:

- StarUML (opensource)
- Argo UML
- PoseidonCE
- Rational Rose
- Visual Paradigm (*shareware*)

5.6 Class Diagram

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

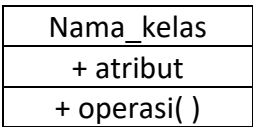


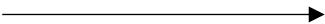
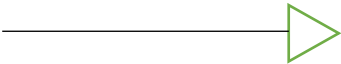


- Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
- Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Diagram kelas dibuat agar pembuat program atau programmer membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak sinkron. Banyak berbagai kasus, perancangan kelas yang dibuat tidak sesuai dengan kelas-kelas yang dibuat pada perangkat lunak, sehingga tidaklah ada gunanya lagi sebuah perancangan karena apa yang dirancang dan hasil jadinya tidak sesuai.

Kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem sehingga pembuat perangkat lunak atau programmer dapat membuat kelas-kelas di dalam program perangkat lunak sesuai dengan perancangan diagram kelas. Susunan struktur kelas yang baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut:

- Kelas main
Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan
- Kelas yang menangani tampilan sistem (*view*)
Kelas yang mendefinisikan dan mengatur tampilan pemakai
- Kelas yang diambil dari pendefinisian usecase (*controller*)
Kelas yang menangani fungsi-fungsi yang harus ada diambil dari pendefinisian use case, kelas ini biasanya disebut dengan kelas proses yang menangani proses bisnis pada perangkat lunak
- Kelas yang diambil dari pendefinisian data (*model*)
Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data. Semua tabel yang dibuat di basis data dapat dijadikan kelas, namun untuk tabel dari hasil relasi atau atribut multivalue pada ERD dapat dijadikan kelas tersendiri dapat juga tidak asalkan pengaksesannya dapat dipertanggungjawabkan atau tetap ada di dalam perancangan kelas.

Berikut adalah simbol-simbol yang ada pada diagram kelas

Simbol	Deskripsi
<p>Kelas</p> 	Kelas pada struktur sistem
<p>Antar muka / interface</p>  <p>Nama_interface</p>	Sama dengan konsep interface dalam pemrograman berorientasi objek
<p>Asosiasi / association</p> 	Relasi antarkelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
<p>Asosiasi berarah / directed</p> 	Relasi antarkelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplicity
<p>Generalisasi</p> 	Relasi antarkelas dengan makna generalisasi-spesialisasi (umum khusus)
<p>Ketergantungan / dependency</p> 	Relasi antarkelas dengan makna
<p>Agregasi / aggregation</p> 	Relasi antarkelas dengan makna semua-bagian (whole-part)

- ❖ Arah panah relasi pada diagram kelas mengarah pada diagram kelas yang lebih besar kontrolnya atau yang dipakai.

5.7 Object Diagram

Diagram object menggambarkan struktur sistem dari segi penamaan objek dan jalannya objek dalam sistem. Pada diagram objek harus dipastikan semua kelas yang sudah didefinisikan pada diagram kelas harus dipakai objeknya, karena jika tidak, pendefinisian kelas itu tidak dapat dipertanggungjawabkan. Diagram objek juga berfungsi untuk mendefinisikan contoh nilai atau isi dari atribut tiap kelas.

Untuk apa mendefinisikan sebuah kelas sedangkan pada jalannya sistem, objek tidak pernah dipakai. Hubungan link pada diagram objek merupakan hubungan memakai dan dipakai dimana dua buah objek akan dihubungkan oleh link jika ada objek yang dipakai oleh objek lainnya.

Berikut adalah simbol-simbol yang ada pada diagram objek:

Simbol	Deskripsi
<p style="text-align: center;">Objek</p> <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;"> <p style="text-align: center;">Nama_objek : nama_kelas</p> <hr style="border: 0; border-top: 1px solid black; margin: 2px 0;"/> <p style="text-align: center;">Atribut = nilai</p> </div>	Objek dari kelas yang berjalan saat sistem dijalankan
<p style="text-align: center;">Link</p> <hr style="border: 0; border-top: 1px solid black; margin: 10px auto; width: 50%;"/>	Relasi antar objek

Penulisan nilai sama dengan penulisan pada kamus data DFD. Misalkan jika sebuah atribut dapat berisi lebih dari satu string maka akan ditulis dengan lambang {"nilai1","nilai2",....., "nilain"}, atau misalkan ingin menuliskan bahwa nilai sebuah atribut string dapat diisi nilai1 atau nilai2 maka akan ditulis ["nilai1" | "nilai2"]

5.8 Component Diagram

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

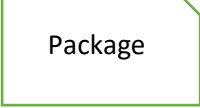
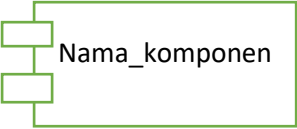



- Source code program perangkat lunak
- Komponen executable yang dilepas ke user
- Basis data secara fisik
- Sistem yang harus beradaptasi dengan sistem lain
- Framework sistem, framework pada perangkat lunak merupakan kerangka kerja yang dibuat untuk memudahkan pengembangan dan pemeliharaan aplikasi,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

Komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut:

- Komponen user interface yang menangani tampilan
- Komponen bussiness processing yang menangani fungsi-fungsi process bisnis
- Komponen data yang menangani manipulasi data.
- Komponen security yang menangani keamanan sistem.

Komponen lebih terfokus pada penggolongan secara umum fungsi-fungsi yang diperlukan.

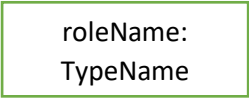
Berikut adalah simbol-simbol yang ada pada diagram komponen:

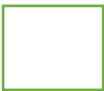

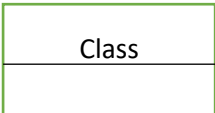
Simbol	Deskripsi
Package 	Package merupakan sebuah bungkus dari satu atau lebih komponen
	Komponen sistem
Kebergantungan dependency 	Kebergantungan antar komponen, arah panah mengarah pada komponen yang dipakai
Antarmuka / interface  nama_interface	Sama dengan konsep interface pada pemrograman berorientasi objek, yaitu sebagai antarmuka komponen agar tidak mengakses langsung komponen
Link 	Relasi antar komponen

5.9 Composite Structure Diagram

Composite structure diagram baru mulai ada pada UML versi 2.0 pada versi 1.x diagram ini belum muncul. Diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (runtime) dari instance yang saling terhubung. Dapat menggambarkan struktur di dalam kelas atau kolaborasi. Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait untuk menjalankan fungsi mesin tersebut, menggambarkan aliran data router pada jaringan komputer, dan lain-lain.

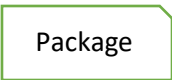
Berikut adalah simbol-simbol yang ada pada diagram composite strukture:

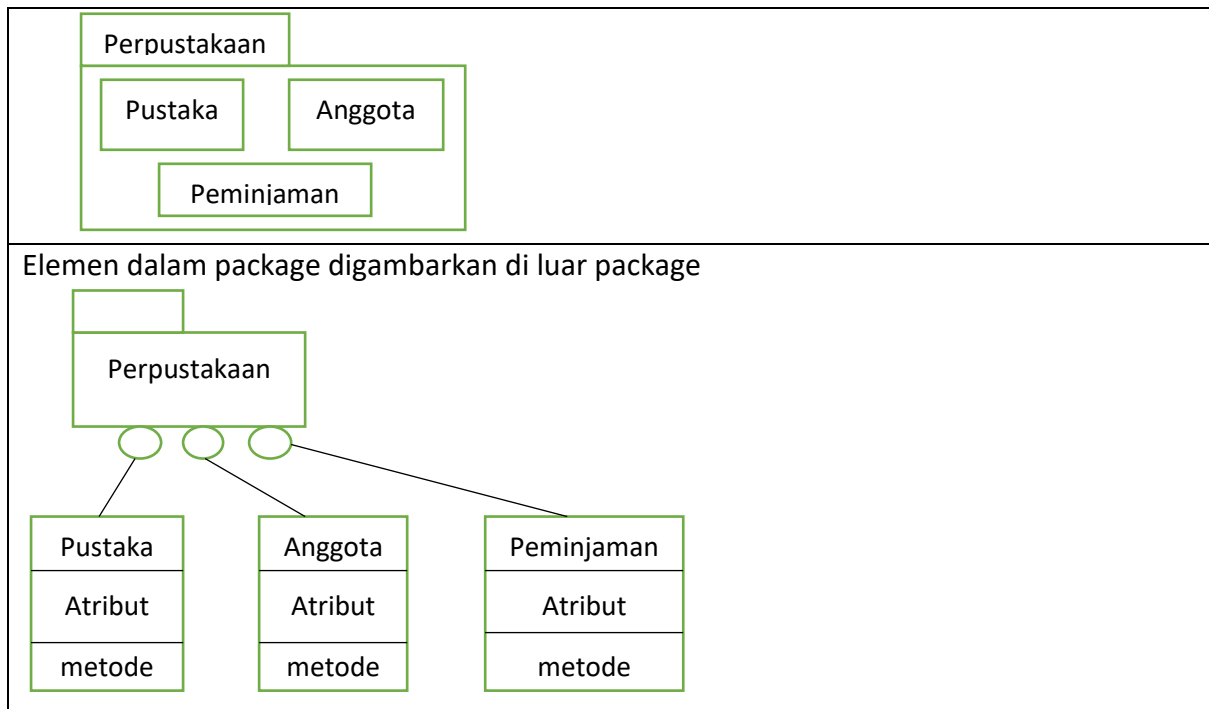
Simbol	Deskripsi
Property 	Property adalah satu set dari suatu instance. roleName : peran / nama / identitas dari property (opsional)

	<p>TypeName : tipe kelas dari property (harus ada)</p>
<p>Connector</p> <p>[multiplicity1] [multiplicity2]</p> <hr/>	<p>Connector adalah cara komunikasi dari 2 buah instance</p> <p>Connector adalah cara komunikasi dari 2 buah instance</p> <p>connName : nama connector (opsional)</p> <p>ConnType : tipe connector (opsional)</p>
<p>Port</p>  <p>PortName:EntityName[n]</p> <p>Pemakaiannya adalah sebagai berikut:</p> 	<p>Port adalah cara yang digunakan dalam diagram composite structure tanpa menampilkan detail internal dari suatu system</p> <p>Port digambarkan dalam bentuk kotak kecil yang menempel atau di dalam suatu property</p> <p>Port digambarkan menempel property jika fungsi tersebut dapat diakses public.</p> <p>Sedangkan port digambarkan di dalam suatu property jika fungsi tersebut bersifat protected</p>
<p>Class</p> 	<p>Kelas; jika yang akan dijabarkan strukturnya adalah sebuah kelas.</p>

5.10 Package Diagram

Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML. Hampir semua diagram dalam UML dapat dikelompokkan menggunakan package diagram. Berikut ini simbol-simbol yang digunakan :

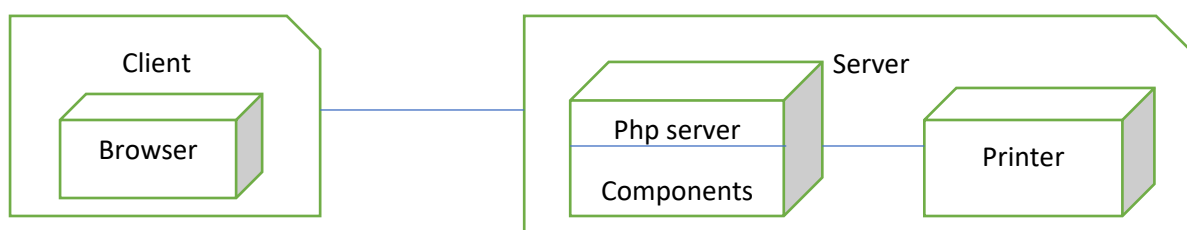
Simbol	Deskripsi
<p>Package</p> 	<p>Package merupakan sebuah bungkus dari satu atau lebih kelas atau elemen diagram UML lainnya.</p>
Elemen dalam package digambarkan di dalam package	



5.11 Deployment Diagram

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

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

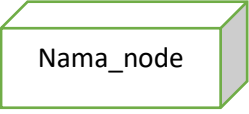




gambar 14. Diagram Deployment Sistem Client / Server

- Sistem terdistribusi murni
- Rekayasa ulang aplikasi

Berikut adalah simbol-simbol yang ada pada diagram deployment:

Simbol	Deskripsi
<div style="border: 1px solid black; padding: 5px; width: 100px; margin: 0 auto;">Package</div>	Package merupakan sebuah bungkus dari satu atau lebih node

Node 	Biasanya mengacu pada perangkat keras (<i>hardware</i>), perangkat lunak yang tidak dibuat sendiri (<i>software</i>), jika di dalam node disertakan. Komponen untuk mengkonsistenkan rancangan maka komponen yang diikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen
Kebergantungan / dependency 	Kebergantungan antar node, arah panah mengarah pada node yang dipakai
Link 	Relasi antar node

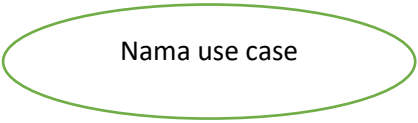
5.12 Use Case Diagram



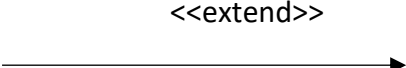
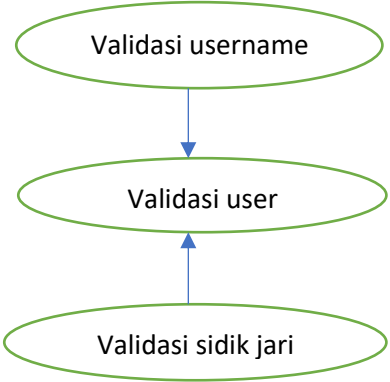

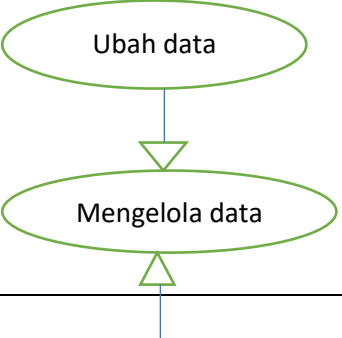
Use case atau diagram use case merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use Case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang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

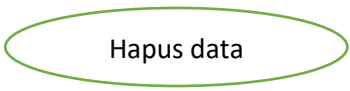
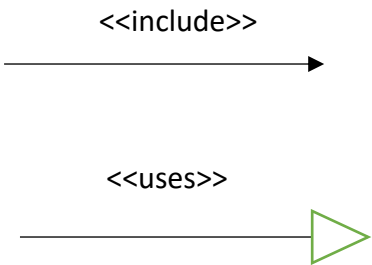
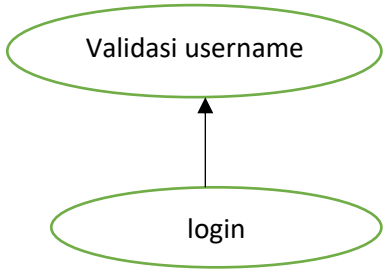
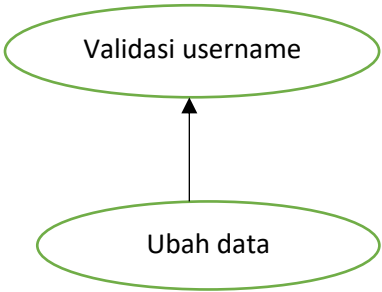
Syarat penamaan pada use case adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada use case yaitu pendefinisian apa yang disebut aktor dan use case.

- Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
- Use case merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

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

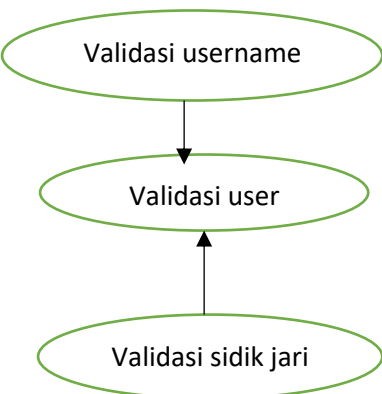
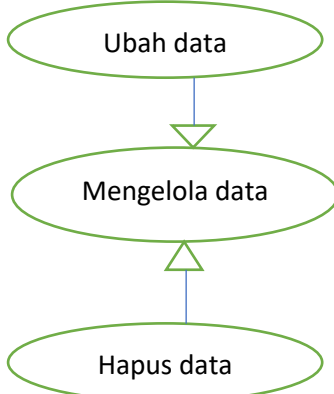

Simbol	Deskripsi
Use case 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor; biasanya dinyatakan dengan menggunakan kata kerja di awal di awal frase mana use case

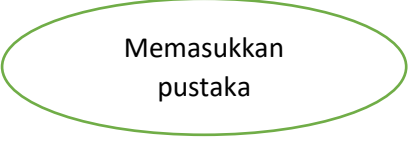
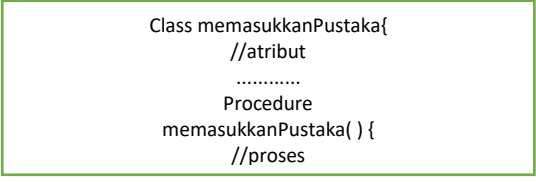
<p>Aktor / actor</p>  <p>Nama aktor</p>	<p>Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri. Jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor.</p>
<p>Asosiasi / association</p> 	<p>Komunikasi antara aktor dan use case yang berpartisipasi pada use case atau use case memiliki interaksi dengan aktor</p>
<p>Ekstensi / extend</p> 	<p>Relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan dapat berdiri sendiri walau tanpa use case tambahan itu; mirip dengan prinsip inheritance pada pemrograman berorientasi objek; biasanya use case tambahan memiliki nama depan yang sama dengan use case yang ditambahkan</p>  <p>Arah panah mengarah pada use case yang ditambahkan; biasanya use case yang menjadi extend-nya merupakan jenis yang sama dengan use case yang menjadi induknya.</p>
<p>Generalisasi / generalization</p> 	<p>Hubungan generalisasi dan spesialisasi (umum – khusus) antara dua buah use case dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya, misalnya:</p> 

	 <p>Arah panah mengarah pada use case yang menjadi generalisasi (umum).</p>
Menggunakan / include / uses	<p>Relasi use case tambahan ke sebuah use case di mana use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsinya atau sebagai syarat.</p>
	<p>Dijalankan use case ini.</p> <p>Ada dua sudut pandang yang cukup besar mengenai include di use case:</p> <ul style="list-style-type: none"> • Include berarti use case yang ditambahkan akan selalu dipanggil saat use case tambahan dijalankan, misal pada kasus berikut:  <ul style="list-style-type: none"> • Include berarti use case yang tambahan akan selalu melakukan pengecekan apakah use case yang ditambahkan telah dijalankan sebelum use case tambahan dijalankan, misal pada kasus berikut:  <p>Kedua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi yang dibutuhkan.</p>

- ❖ Arah panah relasi pada use case mengarah pada use case yang lebih besar kontrolnya atau yang dipakai.

Use case nantinya akan menjadi kelas proses pada diagram kelas sehingga perlu dipertimbangkan penamaan yang dilakukan apakah sudah layak menjadi kelas atau belum sesuai dengan aturan pendefinisian kelas yang baik. Berikut adalah aturan perubahan use case yang layak menjadi kelas proses sehingga layak sebagai dijadikan use case:

Hubungan	Keterangan
<p>Ekstensi / extend</p> 	<p>Pada hubungan ekstensi maka dapat hanya diambil use case induknya yang dijadikan kelas dengan metode berupa use case ekstensinya</p> <pre> Class ValidasiUse { //atribut Procedure ValidasiUsername () { //proses } Procedure ValidasiSidikJari () { //proses } } </pre>
<p>Generalisasi / generalization</p> 	<p>Pada hubungan generalisasi maka dapat hanya diambil use case umumnya yang dijadikan kelas dengan metode berupa use case khususnya</p> <pre> Class MengelolaData{ //atribut Procedural ubahData () { //process } Procedure hapusData () { //proses } } </pre>
<p>Use case yang berdiri sendiri</p> 	<p>Metode yang mungkin bisa ada di dalam kelas proses login adalah sebagai berikut:</p> <pre> Class Login { //atribut Procedure login () { //proses } } </pre>

Use case yang kurang tepat sebagai sebuah use case yang berdiri sendiri 	Kurang tepat karena kelasnya akan menjadi: 

Setiap use case dilengkapi dengan skenario, Skenario use case adalah alur jalannya proses use case dari sisi aktor dan sistem. Berikut adalah format tabel skenario use case:

Aksi Aktor	Reaksi Sistem
Skenario Normal	
Skenario Alternatif	

Skenario *use case* dibuat per use case terkecil, misalkan untuk generalisasi maka skenario yang dibuat adalah *use case* yang lebih khusus. Skenario normal adalah skenario bila sistem berjalan normal tanpa terjadi kesalahan atau *error*. Sedangkan skenario alternatif adalah skenario bila sistem tidak berjalan normal, atau mengalami *error*. Skenario normal dan skenario alternatif dapat lebih dari satu. Alur dari skenario inilah yang nantinya menjadi dasar pembuatan diagram sekuen.



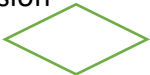


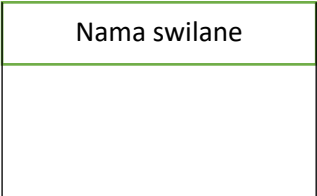

5.13 Activity Diagram

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

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

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

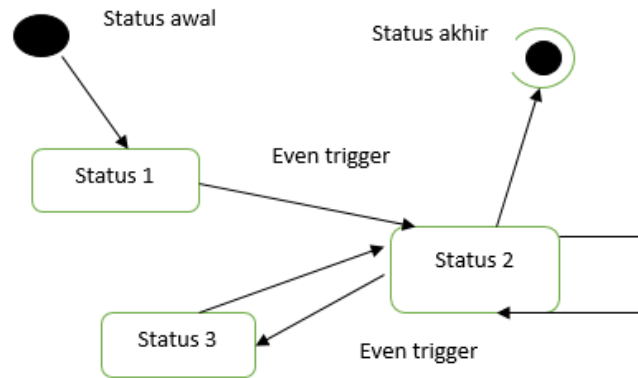
Berikut adalah simbol-simbol yang ada pada diagram aktivitas:

Simbol	Deskripsi
Status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal
Aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja
Percabangan / decision 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu
Penggabungan / join 	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu
Status akhir 	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir
Swimlane  Atau 	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi

5.14 State Machine Diagram





State machine diagram atau state chart diagram atau dalam bahasa Indonesia disebut diagram mesin status atau sering juga disebut diagram status digunakan untuk menggambarkan perubahan status dan transisi status dari sebuah mesin atau sistem atau objek. Diagram status digunakan untuk interaksi di dalam sebuah objek. Perubahan tersebut digambarkan dalam suatu graf berarah. State machine diagram merupakan pengembangan dari diagram Finite state automata dengan penambahan beberapa fitur dan konsep baru. *Diagram Finite State automata (FSA)* ini biasanya diajarkan dalam mata kuliah automata.

State machine diagram cocok digunakan untuk menggambarkan alur interaksi pengguna dengan sistem. Berikut ini adalah contoh gambar diagram mesin status.



gambar 15.contoh State machine diagram

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


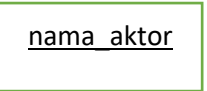

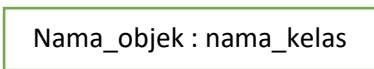


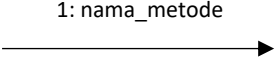
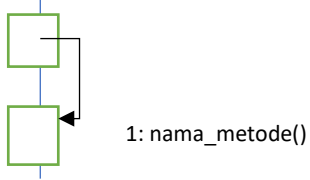
Simbol	Deskripsi
Start / Status awal (Inisiate State) 	Star atau initial state adalah state atau keadaan awal pada saat sistem mulai hidup
End / Status akhir (Final State) 	End atau final state adalah state keadaan akhir dari daur hidup suatu sistem
Event 	Event adalah kegiatan yang menyebabkan berubahnya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

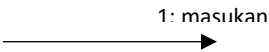

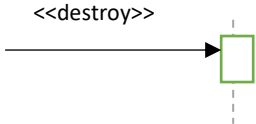
5.15 Sequence Diagram

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

Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian use case yang memiliki proses sendiri atau yang penting semua use case yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak use case yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.

Berikut adalah simbol-simbol yang ada pada diagram sekuen:

Simbol	Deskripsi
<p>Aktor</p>  <p>nama aktor</p> <p>atau</p>  <p>nama_aktor</p> <p>tanpa waktu aktif</p>	<p>Orang , Proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang, biasanya dinyatakan menggunakan kata benda di awal frase nama aktor</p>
<p>Garis hidup / lifetime</p> 	<p>Menyatakan kehidupan suatu objek</p>
<p>Objek</p>  <p>Nama_objek : nama_kelas</p>	<p>Menyatakan objek yang berinteraksi pesan</p>
<p>Waktu aktif</p>	<p>Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah sebuah tahapan yang dilakukan di dalamnya, misalnya</p>  <p>Maka CekStatusLogin() dan open() dilakukan di dalam metode login()</p> <p>Aktor tidak memiliki waktu aktif</p>
<p>Pesan tipe Create</p> 	<p>Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat</p>
<p>Pesan tipe call</p>  <p>1: nama_metode</p>	<p>Menyatakan suatu objek memanggil operasi / metode yang ada pada objek lain atau dirinya sendiri,</p>  <p>1: nama_metode()</p>

	Arah panah mengarah pada objek yang memiliki operasi/metode, karena ini memanggil operasi/metode maka operasi / metode yang dipanggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi.
Pesan tipe send 	Menyatakan bahwa suatu objek mengirimkan data/masukan/informasi ke objek lainnya, arah panah mengarah pada objek yang dikirim.
Pesan tipe return 	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian
Pesan tipe destroy 	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada create maka ada destroy

Penomoran pesan berdasarkan urutan interaksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu.

Semua metode di dalam kelas harus ada di dalam diagram kolaborasi atau sekuen. Jika ada berarti perancangan metode di dalam kelas itu kurang baik. Hal ini dikarenakan ada metode yang tidak dapat dipertanggungjawabkan kegunaannya.

5.16 Communication Diagram

Communication diagram atau diagram komunikasi pada UML versi 2.X adalah penyederhanaan dari diagram kolaborasi (Collaboration diagram) pada UML versi 1.X. Collaboration diagram sudah tidak muncul lagi pada UML versi 2.X. Diagram komunikasi sebenarnya adalah diagram kolaborasi tetapi dibuat untuk setiap sekuen.

Diagram komunikasi menggambarkan interaksi antar objek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari Diagram kelas, Diagram sekuen, dan Diagram Use Case untuk mendeskripsikan gabungan antara struktur statis dan tingkah laku dinamis dari suatu sistem.

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

Berikut adalah simbol-simbol yang ada pada diagram kolaborasi:

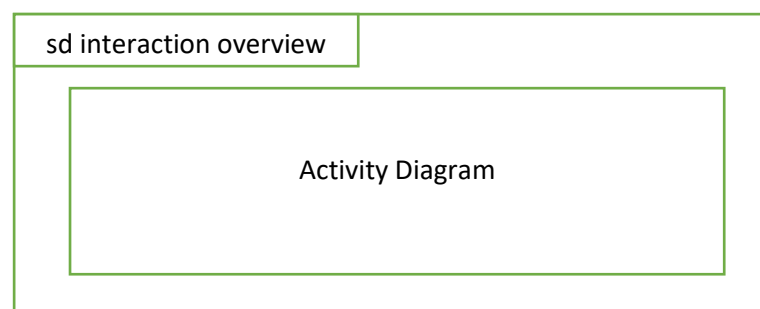
5.18 Interaction Overview Diagram

Interaction overview diagram mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. *Interaction Overview Diagram* adalah bentuk aktivitas diagram yang setiap titik merepresentasikan diagram interaksi. Interaksi diagram dapat meliputi diagram sekuen, diagram komunikasi, *interaction overview diagram*, dan *timing diagram*.

Hampir semua notasi pada *interaction overview* digram sama dengan notasi pada diagram aktivitas. Sebagai contoh *initiasl*, *final*, *decision*, *merge*, *fork*, dan *join nodes* sama pada diagram aktivitas. Tambahan pada *interaction overview diagram* adalah *interaction occurance* dan *interaction element*.

1. *Interaction Occurrence*

Interaction occurence atau kejadian interaksi adalah referensi untuk diagram interaksi yang ada. Sebuah *interaction occurence* ditunjukkan sebagai frame referensi (frame dengan tulisan “ref” di pojok kiri atas). Nama diagram yang sedang direferensikan ditunjukkan pada tengah frame. Berikut adalah gambar contoh dari *interaction occurrence*:

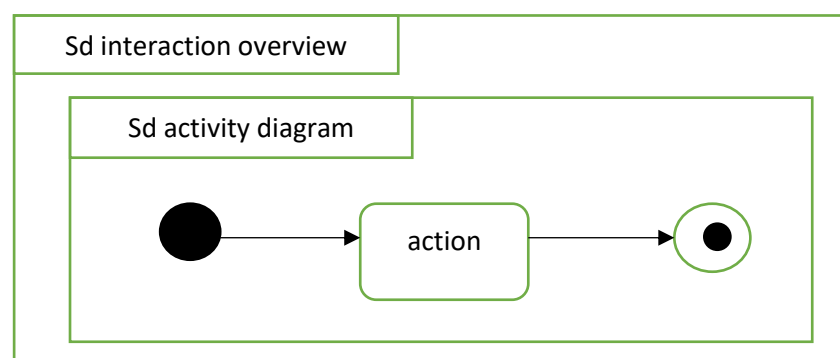


gambar 17.contoh interaction occurence

sd didalam gambar adalah kependekan dari *sequence diagram*.

2. *Interaction Element*

Interaction Element atau elemen interaksi mirip *interaction occurence*. Perbedaanya adalah di dalam *interaction elemen* menampilkan isi diagram yang direferensikan secara langsung, sedangkan *interaction occurrence* hanya menampilkan nama diagram yang direferensikan. Berikut adalah gambar contoh dari *interaction element*:



gambar 18.contoh interaction element

Berikut ini adalah contoh *interaction overview* diagram yang menampilkan kontrol-kontrol pada diagram aktivitas (*fork, join, merge*, dan lain-lain) dengan abstraksi subproses digambarkan menggunakan *interaction occurrence*.

Soal Latihan

1. Mengapa dibutuhkan pemodelan dalam melakukan rekayasa perangkat lunak?
2. Apa tujuan munculnya UML? Jelaskan?
3. Apa pengertian use case? Kapan penggunaan use case?
4. Sebutkan perbedaan class diagram dan object diagram?
5. Sebutkan perbedaan state machine diagram dengan FSA (Finite State Diagram)
6. Apa fungsi timing diagram?
7. Apa perbedaan UML versi 1.X dengan UML 2.X?
8. Mengapa collaboration diagram dihilangkan pada UML versi 2.X?
9. Sebutkan perbedaan antara collaboration diagram dengan communication diagram?
10. Apakah diagram UML bisa dimanfaatkan untuk analisis dan desain sistem yang tidak berorientasi objek? Jelaskan?

BAB 6 ANALISIS DAN DESAIN BERORIENTASI OBJEK

6.1 Analisis Berorientasi Objek

Analisis berorientasi objek atau *Object Oriented Analysis* (OOA) adalah tahapan untuk menganalisis spesifikasi atau kebutuhan akan sistem yang akan dibangun dengan konsep berorientasi objek, apakah benar kebutuhan yang ada dapat diimplementasikan menjadi sebuah sistem berorientasi objek

Analisis ini sebaiknya dilakukan oleh orang-orang yang benar-benar memahami implementasi sistem yang berbasis atau berorientasi objek. Karena tanpa pemahaman itu maka sistem yang dihasilkan bisa jadi tidak realistis jika diimplementasikan dengan berbasis objek.

OOA biasanya menggunakan kartu CRC (*Component, Responsibility, Collaborator*) untuk membangun kelas-kelas yang akan digunakan atau menggunakan UML (*Unified Modeling Language*) pada bagian diagram *use case*, diagram kelas, dan diagram objek.

Bahasa pemodelan berorientasi objek pada awal perkembangannya (Generasi awal) diantaranya adalah:

1. CRC (*Component, Responsibility, Collaborator*)

CRC diperkenalkan oleh Kent Beck dan Ward Cunningham pada tahun 1989 sebagai bagian dari *Object-Oriented Programming, System, Languages and Applications* (OOPSLA). CRC dibuat untuk bakal yang akan menjadi kelas yang dianalisis. Berikut adalah contoh sebuah kartu CRC:

Class Name : Person	
Superclasses:	
Subclasses:	
Responsibilities	Collaborators
Gender Age Calculate age at given Year Cook capalility	Cuisine, Receipe

- Nama kelas (*Class Name*)
Merupakan nama yang diberikan pada sebuah kelas
- Kelas Orang Tua (*Superclasses*)
Merupakan kelas anak (dalam hubungan pewarisan) atau sub kelas dari kelas yang dibuat CRC-nya
- Kelas anak (*Subclasses*)

Merupakan kelas anak (dalam hubungan pewarisan) atau sub kelas dari kelas yang dibuat CRC-nya.

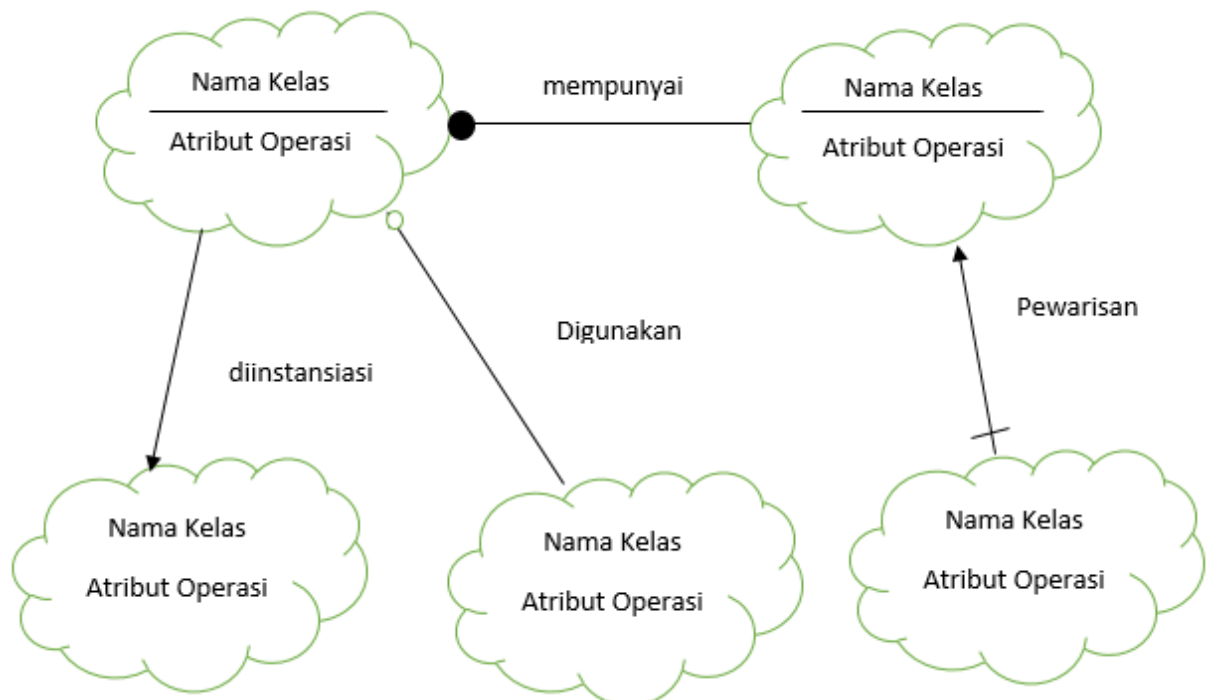
- **Tanggung Jawab (*Responsibilities*)**
Merupakan isi atribut dan operasi yang harus ada dalam kelas yang dibuat CRC-nya
- **Kolaborator/Kelas yang terkait (*Collaborators*)**
Merupakan kelas yang terkait (untuk bekerjasama) dengan kelas yang sedang dibuat CRC-nya tetapi bukan merupakan kelas orang tua atau kelas anak. Terkait dalam hubungan memakai dan dipakai di dalam kelas.

CRC dibuat per kelas. Keterkaitan antara kelas dapat dilihat pada kolom kelas super, subkelas dan kolaborator/kelas terkait. Dalam sebuah analisis pembuatan kelas, dapat terdiri dari banyak kartu CRC.

CRC awalnya dibuat di kertas, lalu ditempelkan di papan atau di meja di ruang diskusi pengembang / *developer* perangkat lunak.

2. Metode Booch

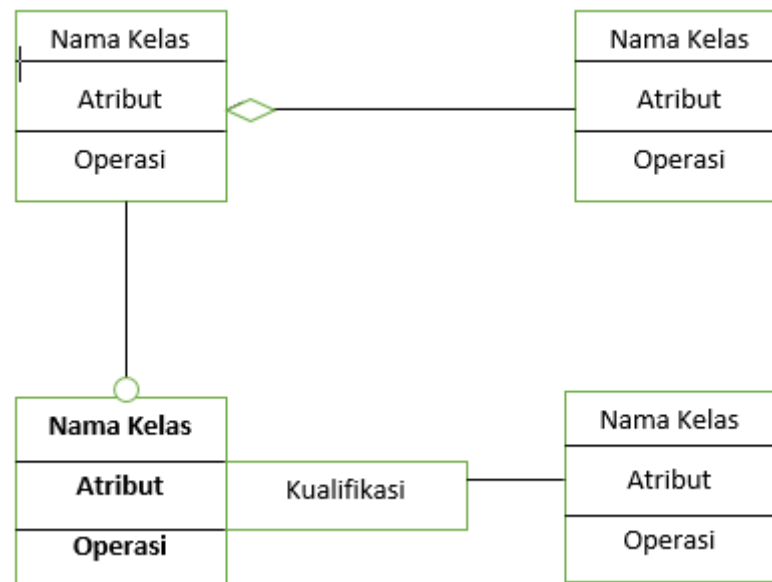
Metode Booch dikembangkan oleh Grady Booch pada tahun 1991. Metode Booch terdiri dari enam diagram antara lain diagram kelas, objek, transisi status, interaksi, modul, dan proses. Berikut adalah contoh diagram kelas dari metode Booch:



gambar 19. diagram Booch

3. OMT (*Object Modelling Technique*)

Dikembangkan oleh James Rumbaugh, Blaha, Premerlani, Eddy dan Iorensen sebagai metode untuk mengembangkan sistem berorientasi objek, dan untuk mendukung pemrograman berorientasi objek pada tahun 1991. Berikut adalah salah satu contoh diagram OMT:



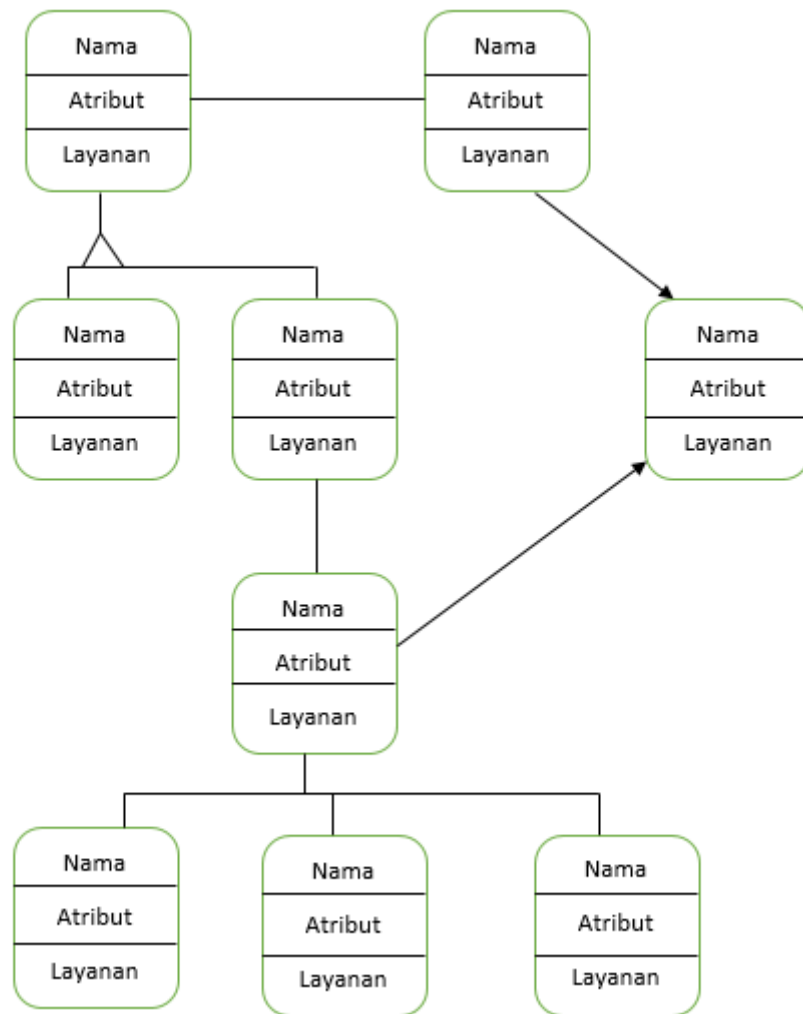
gambar 20. Contoh diagram OMT

4. Metode Coad Yourdon

Metode ini diperkenalkan pada tahun 1991, metode ini menyediakan sebuah diagram kelas. Diagram kelas pada metode ini dibuat dengan langkah-langkah sebagai berikut:

- Mendefinisikan kelas dan objek
- Mengidentifikasi struktur kelas dan objek
- Mendefinisikan subjek nama kelas
- Mendefinisikan atribut
- Mendefinisikan operasi / layanan (*services*)

Berikut adalah contoh diagram kelas Coad Yourdon:



gambar 21.contoh diagram kelas Coad Yourdon

5. OOSE (*Object Oriented Software Engineering*)

OOSE dikembangkan oleh Ivar Jacobson pada tahun 1992. OOSE adalah metodologi desain berorientasi objek yang mulai melibatkan use case (penggunaan kasus).

- ❖ Saat ini pemodelan yang paling sering digunakan pada pemograman berorientasi objek adalah menggunakan UML.

6.2 Desain Berorientasi Objek

Desain berorientasi objek atau *Object Oriented Design* (OOD) adalah tahapan perantara untuk memetakan spesifikasi atau kebutuhan sistem yang akan dibangun dengan konsep berorientasi objek ke desain pemodelan agar lebih mudah diimplementasikan dengan pemograman berorientasi objek.

Pemodelan berorientasi objek biasanya dituangkan dalam dokumentasi perangkat lunak dengan menggunakan perangkat pemodelan berorientasi objek, diantaranya adalah UML

(Unified Modeling Language). Kendala dan permasalahan pembangunan sistem berorientasi objek biasanya dapat dikenali dalam tahap ini.

OOA dan OOD dalam proses yang berulang-ulang sering memiliki batasan yang samar, sehingga kedua tahapan ini sering juga disebut OOAD (object oriented analysis and design) atau dalam bahasa Indonesia berarti Analisis dan Desain Berorientasi Objek.

- ❖ OOA dan OOD sering kali memiliki batasan yang samar, sehingga biasanya disebutkan langsung menjadi OOAD (*Object Oriented Analysis and Design*)

6.3 CASE Tools

CASE (*Computer-Aided Software Engineering*) tools atau perangkat pembantu berbasis komputer untuk rekayasa perangkat lunak adalah aplikasi atau perangkat lunak yang membantu pembuatan sebuah sistem perangkat lunak, istilah CASE muncul karena kebutuhan para pelaku rekayasa perangkat lunak akan perangkat yang memudahkan pekerjaan mereka.

CASE tools mulai digunakan oleh pelaku rekayasa perangkat lunak sejak awal tahun 1980-an dalam hal ini mereka fokus pada tahap awal pengembangan perangkat lunak seperti tahap pengumpulan kebutuhan, desain dan analisis di dalam alur hidup (*life-cycle*) pengembangan perangkat lunak. Namun, saat ini CASE tools tidak hanya berfokus pada tahap awal pengembangan perangkat lunak, tapi sudah ke semua bagian pengembangan dan perbaikan perangkat lunak.

Perangkat lunak yang termasuk CASE tools dapat merupakan perangkat lunak dalam tahap analisis, desain, dokumentasi, maupun implementasi ke dalam kode program dari pengujian program. CASE tools termasuk editor desain perangkat lunak, editor kode program, kompiler, debugger, perangkat pembangunan sistem perangkat lunak dan lain sebagainya. CASE juga dapat mengacu pada metode yang mendedikasikan pada disiplin rekayasa sistem informasi yang bersifat otomatis (tidak manual). CASE pada dasarnya digunakan untuk membangun kualitas perangkat lunak yang memiliki performansi efektif.

CASE tools dapat dikelompokkan menjadi empat dimensi yang berbeda sebagai berikut.

1. Pendukung alur hidup perangkat lunak (*life Cycle support*) dibagi menjadi dua buah kelompok, yaitu
 - Upper CASE tools
Upper CASE tools mendukung perencanaan strategis dan pembangunan perangkat lunak, misalnya aplikasi untuk membuat diagram ER (*Entity Relationship*), DFD (*Data Flow Diagram*), dan diagram perencanaan / desain perangkat lunak yang lain. Termasuk juga perangkat lunak untuk membuat diagram UML, seperti:
 - StarUML
 - Argo UML
 - PoseidonCE

- Rational Rose
- Visual Paradigm
- Lower CASE tools
Lower CASE tools fokus pada bagian akhir pembangunan perangkat lunak seperti implementasi perangkat lunak, perbaikan kesalahan perangkat lunak, pengujian, konstruksi dan integrasi perangkat lunak, pemeliharaan perangkat lunak, perekayasaan perangkat lunak kembali (*reengineering*), atau *reverse engineering* (perbaikan rekayasa perangkat lunak diawali dari perangkat lunak yang sudah jadi).
- 2. Dimensi integrasi (*integration dimension*)
Dimensi ini fokus pada integrasi komponen perangkat lunak, seperti misalnya framework (kerangka kerja), integrasi lingkungan pengembangan perangkat lunak dan lain sebagainya.
- 3. Dimensi konstruksi (*construction dimension*)
Dimensi ini fokus pada proses konstruksi perangkat lunak, misalkan untuk rekayasa ulang (*reengineering*), *reverse engineering*, atau konstruksi awal perangkat lunak.
- 4. Dimensi CASE berbasis keilmuan (*knowledge-based CASE*)
Dimensi ini fokus pada perangkat untuk membantu perangkat lunak memiliki basis keilmuan (biasanya berupa perangkat lunak pemodelan simulasi proses bisnis yang diinginkan misalkan untuk membangun sistem pendukung keputusan (*Decision Support System*) atau sistem Pakar (*Expert System*).

Banyak CASE tools yang dapat digunakan membuat perlunya ada standarisasi CASE tools yang baik itu sendiri untuk menghindari hal-hal berikut:

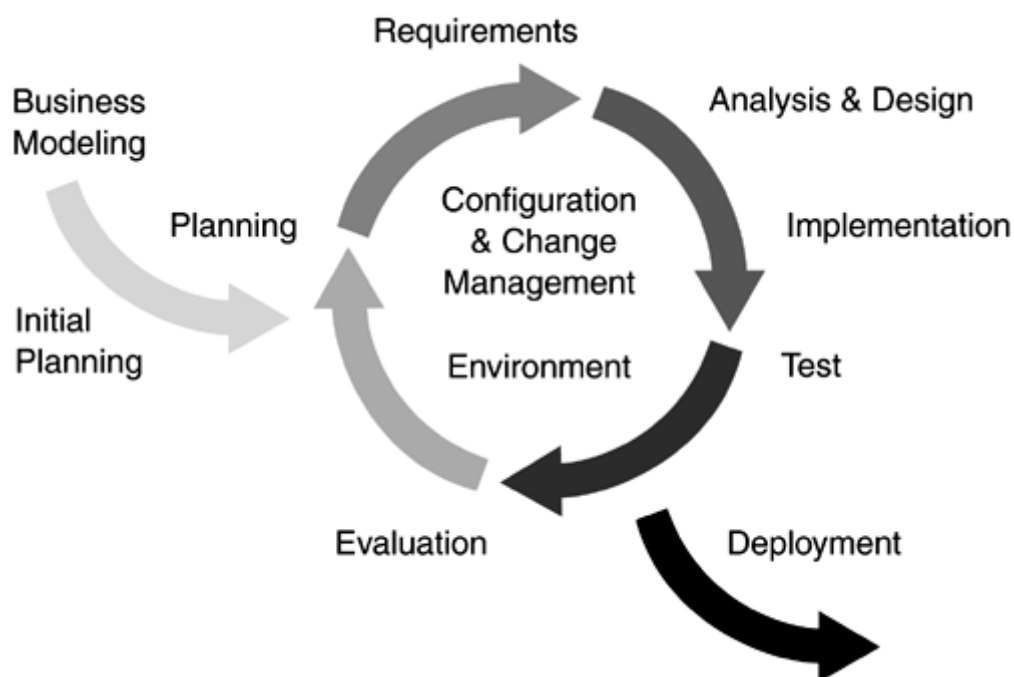
- Standarisasi yang tidak memadai yang dapat menyebabkan integrasi komponen antar yang menggunakan CASE tools satu dan lainnya tidak dapat dilakukan karena perbedaan standar.
- Harapan yang tidak realistis karena tidak menggunakan CASE tools sehingga batasan-batasan yang seharusnya tidak dilakukan dapat dibatasi jika menggunakan CASE tools.
- Waktu implementasi yang lambat.
- Kontrol penyimpanan yang lemah, misalkan kontrol pada penyimpanan dokumentasi, kode program versi terbaru dan lain sebagainya yang terkait dengan perangkat lunak.
- ❖ CASE tools adalah alat bantu untuk pengembangan perangkat lunak agar hasilnya lebih baik dan waktu pengerjaannya lebih singkat.

6.4 RUP

Unified Process atau dikenal juga dengan proses iteratif dan incrementasi merupakan sebuah proses pengembangan perangkat lunak yang dilakukan secara iteratif (berulang) dan inkremental (bertahap dengan progres menaik). Iteratif dapat dilakukan di dalam setiap tahap, atau iteratif tahap pada proses pengembangan perangkat lunak untuk menghasilkan perbaikan fungsi yang inkremental (bertambah menaik) di mana setiap iterasi akan memperbaiki iterasi berikutnya. Salah satu *Unified Process* yang terkenal adalah RUP (*Rational Unified Process*).

RUP adalah pendekatan pengembangan perangkat lunak yang dilakukan berulang-ulang (*iterative*), fokus pada arsitektur (*architecture-centric*), lebih diarahkan berdasarkan penggunaan kasus (*use case driven*). RUP merupakan proses rekayasa perangkat lunak dengan pendefinisian yang baik (*well defined*) dan perstrukturannya yang baik (*well structured*). RUP menyediakan pendefinisian struktur yang baik untuk alur hidup proyek perangkat lunak. RUP adalah sebuah produk proses perangkat lunak yang dikembangkan oleh Rational Software yang diakuisisi oleh IBM di bulan Februari 2003.

Proses pengulangan / iteratif pada RUP secara global dapat dilihat pada gambar berikut:



gambar 22. Proses iteratif RUP

- ❖ RUP (*Rational Unified Process*) adalah tahapan pengembangan sistem secara iteratif khusus untuk pemrograman berorientasi objek.

6.4.1 Kelebihan RUP

Pendekatan iteratif/pengulangan dari RUP dapat mengakomodir beberapa kelemahan pengembangan perangkat lunak tanpa menggunakan konsep pengulangan, misalnya pada pengembangan perangkat lunak waterfall, berikut adalah hal-hal yang dapat diatasi oleh RUP dibandingkan waterfall.

1. RUP mengakomodasi perubahan kebutuhan perangkat lunak.
Kebutuhan untuk mengubah dan menambah fitur karena perubahan teknologi atau ketidakeinginan pelanggan (customer) merupakan salah satu kendala yang sering dialami pengembang perangkat lunak yang berimbas pada keterlambatannya waktu penyelesaian perangkat lunak. Keterlambatan ini dapat menyebabkan ketidakpuasan di sisi pelanggan (customer) dan pengembang menjadi frustrasi. Pengembangan secara iteratif fokus tim dalam membangun dan mendemonstrasikan perangkat lunak untuk beberapa minggu berikutnya sehingga memaksa untuk fokus pada kebutuhan yang lebih penting.
2. Integrasi bukanlah sebuah proses besar dan cepat ("big bang") diakhir proyek.
Menempatkan integrasi di bagian akhir proyek biasanya memakan waktu proyek yang cukup banyak, bisa mencapai 40 persen dari waktu proyek. Untuk mencegah hal ini maka pendekatan secara iteratif (pengulangan) dapat memecah proyek menjadi bagian iterasi kecil yang diakhiri dengan integrasi kecil yang nantinya digabungkan menjadi integrasi besar.
3. Risiko biasanya ditemukan atau dialamatkan selama pada proses integrasi awal
Pendekatan integrasi pada RUP mengurangi resiko pada iterasi awal dimana saat semua komponen diuji.
4. Manajemen berarti membuat perubahan taktik pada produk.
Taktik produk misalnya pengembangan dengan waktu singkat akan menghasilkan produk dengan fungsi yang terbatas akan dapat cepat digunakan oleh user sehingga memperkenalkan produk lebih cepat ke masyarakat dibandingkan produk kompetitor lain yang masih sedang dikembangkan.
5. Mendukung fasilitas penggunaan kembali.
Lebih mudah untuk mengidentifikasi bagian umum yang sering digunakan dalam aplikasi jika diimplementasikan secara iterasi daripada mengidentifikasi pada saat perencanaan saja. Peninjauan kembali pada iterasi awal dapat membuat arsitek perangkat lunak untuk menandai peluang penggunaan kembali (*reuse*) dan kemudian mengembangkan kode umum yang lebih baik atau mapan pada iterasi berikutnya.
6. Kecacatan dapat ditemukan dan diperbaiki pada beberapa iterasi menghasilkan arsitektur yang baik dan aplikasi berkualitas tinggi.

Kecacatan dideteksi lebih baik mulai pada iterasi awal daripada pada tahap pengujian. Sehingga kejadian seperti bottleneck dapat ditemukan tidak pada saat aplikasi sudah dikirimkan ke pelanggan (*customer*).

7. Lebih baik menggunakan “anggota proyek” dibandingkan susunan secara seri pada tim proyek.

Pada pendekatan waterfall anggota tim bekerja secara seri, seperti seorang analis bekerja untuk menganalisis kebutuhan sistem lalu memberikan hasil analisis ke desainer untuk melakukan desain sistem, kemudian desainer memberikan desain ke programmer dan programmer mengirimkan aplikasi ke pelanggan (*customer*). Hal ini dapat menyebabkan kesalahpahaman akan hasil sebelumnya misalkan antara analis kebutuhan dan programmer. Selain itu juga dapat menyebabkan kesalahpahaman akan siapa yang harus bertanggung jawab jika terjadi kecacatan atau kesalahan. Hal ini dapat terjadi karena pada pendekatan waterfall seorang analis hanya bekerja pada waktu analisis kebutuhan saja, desainer hanya pada waktu desain saja, dan programmer hanya pada saat implementasi dan pengujian program. Dengan menggunakan mekanisme “anggota proyek” maka setiap iterasi akan terjadi kerjasama antar anggota proyek untuk saling memperbaiki yang menjadi tanggung jawabnya.

8. Anggota tim belajar selama proyek berjalan.

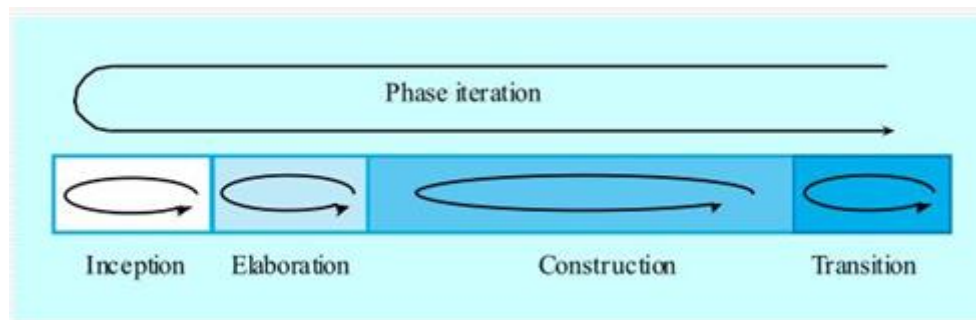
Anggota proyek memiliki peluang belajar dari kesalahan selama siklus pengembangan perangkat lunak dan memperbaiki kesalahan pada iterasi berikutnya. Kesalahan yang terjadi pada iterasi sebelumnya dapat diperbaiki pada iterasi selanjutnya.

9. Pengembangan perangkat lunak dapat diperbaiki seiring proses pengembangan perangkat lunak.

Setiap akhir proses iterasi tidak hanya ada peninjauan mengenai target produk tetapi juga peninjauan pada kesalahan yang terjadi pada iterasi sebelumnya agar dapat diperbaiki pada iterasi berikutnya.

6.4.2 Fase RUP

RUP memiliki empat buah tahap atau fase yang dapat dilakukan pula secara iteratif. Berikut ini adalah gambar alur hidup RUP.



gambar 23. Alur hidup RUP

Berikut ini penjelasan untuk setiap fase pada RUP.

1. Inception (permulaan)

Tahap ini lebih pada memodelkan proses bisnis yang dibutuhkan (*business modeling*) dan mendefinisikan kebutuhan akan sistem yang akan dibuat (*requirements*). Berikut adalah tahap yang dibutuhkan pada tahap ini:

- Memahami ruang lingkup dari proyek (termasuk pada biaya, Waktu, Kebutuhan, Resiko dan lain sebagainya)
- Membangun kasus bisnis yang dibutuhkan.

Hasil yang diharapkan dari tahap ini adalah memenuhi *LifeCycle Objective Milestone* (batas/tonggak objectif dari siklus) dengan kriteria berikut :

- Umpan balik dari pendefinisian ruang lingkup, perkiraan biaya, dan perkiraan jadwal
- Kebutuhan dimengerti dengan pasti (dapat dibuktikan) dan sejalan dengan kasus primer yang dibutuhkan.
- Kredibilitas dari perkiraan biaya, perkiraan jadwal, penentuan skala prioritas, resiko dan proses pengembangan.
- Ruang lingkup purwarupa (*prototype*) yang akan dikembangkan
- Membangun garis dasar dengan membandingkan perencanaan aktual dengan perencanaan yang direncanakan.

Jika pada akhir tahap ini targe yang diinginkan tidak dicapai maka dapat dibatalkan atau diulang kembali setelah dirancang ulang agar kriteria yang diinginkan dapat dicapai. Batas/tonggak objektif digunakan untuk mendeteksi apakah sebuah kebutuhan akan sistem dapat diimplementasikan atau tidak.

2. Elaboration (perluasan / perencanaan)

Tahap ini lebih difokuskan pada perencanaan arsitektur sistem. Tahap ini juga dapat mendeteksi apakah arsitektur sistem yang diinginkan dapat dibuat atau tidak.

Mendeteksi resiko yang mungkin terjadi dari arsitektur yang dibuat. Tahap ini lebih pada analisis dan desain sistem serta implementasi sistem yang fokus pada prototype.

Hasil yang diharapkan dari tahap ini adalah memenuhi *Lifecycle Architecture Milestone* (batas arsitektur dari siklus) dengan kriteria berikut:

- Model kasus yang digunakan (use case) dimana kasus dan aktor yang terlibat telah didefinisikan dan sebagian besar kasus harus dikembangkan. Model use case harus 80 persen lengkap dibuat.
- Deskripsi dari arsitektur perangkat lunak dari proses pengembangan sistem perangkat lunak telah dibuat.
- Rancangan arsitektur yang dapat diimplementasikan dan mengimplementasikan use case.
- Kasus bisnis atau proses bisnis dan daftar risiko yang sudah mengalami perbaikan (revisi) telah dibuat.
- Rencana pengembangan untuk seluruh proyek telah dibuat
- Prototype yang dapat didemonstrasikan untuk mengurangi setiap resiko teknis yang didefinisikan.

Jika pada akhir tahap ini target yang diinginkan tidak dicapai maka dapat dibatalkan atau diulang kembali. Batas / tonggak arsitektur digunakan untuk mendeteksi apakah sebuah kebutuhan akan sistem dapat diimplementasikan atau tidak melalui pembuatan arsitektur.

3. *Construction* (Konstruksi)

Tahap ini fokus pada pengembangan komponen dan fitur-fitur sistem. Tahap ini lebih pada implementasi dan pengujian sistem yang fokus pada implementasi perangkat lunak pada kode program. Tahap ini menghasilkan produk perangkat lunak dimana menjadi syarat dari Initial Operational Capability Milestone atau batas / tonggak kemampuan operasional awal.

4. *Transition* (transisi)

Tahap ini lebih pada developement atau instalasi sistem agar dapat dimengerti oleh user. Tahap ini menghasilkan produk perangkat lunak dimana menjadi syarat dari Initial *Operational Capability Milestone* atau batas / tonggak kemampuan operasional awal. Aktivitas pada tahap ini termasuk pada pelatihan user, pemeliharaan dan pengujian sistem apakah sudah memenuhi harapan user.

Produk perangkat lunak juga disesuaikan dengan kebutuhan yang didefinisikan pada tahap inception. Jika semua kriteria objektif terpenuhi maka dianggap sudah memenuhi Product Release Milestone (batas/ tonggak peluncuran produk) dan pengembangan perangkat lunak selesai dilakukan.

Akhir dari keempat fase ini adalah produk perangkat lunak yang sudah lengkap. Keempat fase pada RUP dijalankan secara berurutan dan iteratif dimana setiap iterasi dapat digunakan untuk memperbaiki iterasi berikutnya.

Latihan Soal

1. Apa perbedaan antara analisis berorientasi objek dengan desain berorientasi objek?
2. Jelaskan bagaimana sejarah perkembangan metode analisis dan desain berorientasi objek?
3. Bandingkan metode OMT dan OOSE, kemudian sebutkan masing-masing kelebihan dan kekurangannya?
4. Bilamana sebaiknya CASE tools digunakan?
5. Jelaskan bagaimana sejarah perkembangan CASE tools?
6. Sebutkan dan jelaskan minimal 2 contoh metode analisis dan desain berorientasi objek selain yang sudah dijelaskan?
7. Sebutkan contoh-contoh CASE tools dan berikan contohnya
8. Apa pengertian RUP?
9. Bagaimana penggunaan RUP?
10. Sebutkan metode pengembangan perangkat lunak selain RUP, sebutkan masing-masing kelebihan dan kekurangannya
11. Kapan sebaiknya kita menerapkan RUP?
12. Apa keterkaitan antara RUP dengan metode OMT dan OOSE? Jelaskan

BAB 7 PEMOGRAMAN BERORIENTASI OBJEK

7.1 Pengertian Pemograman Berorientasi Objek

Metode berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek dan pengujian berorientasi objek.

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransnformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur, jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan saat ini sangat beragam (aplikasi bisnis, *real-time*, utility, dan sebagainya dengan platform yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan metode berorientasi objek adalah sebagai berikut:

1. Meningkatkan produktivitas
Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (reusable)
2. Kecepatan perkembangan
Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.
3. Kemudahan pemeliharaan
Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.
4. Adanya konsistensi
Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.
5. Meningkatkan kualitas perangkat lunak
Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

Saat ini sudah banyak bahasa pemrograman berorientasi objek. Banyak orang berpikir bahwa pemrograman berorientasi objek identik dengan bahasa Java. Memang bahasa Java merupakan bahasa yang paling konsisten dalam mengimplementasikan paradigma pemrograman berorientasi objek. Namun sebenarnya bahasa pemrograman yang mendukung pemrograman berorientasi objek tidak hanya bahasa Java.

- ❖ Java merupakan bahasa pemrograman yang paling konsisten dalam mengimplementasikan paradigma pemrograman berorientasi objek.

Berikut beberapa contoh bahasa pemrograman yang mendukung pemrograman berorientasi objek:

- Bahasa Pemrograman Smalltalk
Smalltalk merupakan salah satu bahasa pemrograman yang dikembangkan untuk mendukung pemrograman berorientasi objek mulai tahun 1978. Smalltalk memiliki berbagai versi, versi banyak dikenal adalah Smalltalk-80 yang dibuat pada tahun 1980.
- Bahasa Pemrograman Eiffel
Eiffel merupakan bahasa pemrograman yang dikembangkan untuk mendukung pemrograman berorientasi objek mulai tahun 1985 oleh bertrand Meyer dan compiler Eiffel selesai dibuat tahun 1987. Eiffel memiliki sintaks mirip dengan sintaks pada bahasa pemrograman Pascal. Eiffel merupakan bahasa pemrograman objek murni karena semua kode programnya dibungkus di dalam kelas.
- Bahasa pemrograman C++
Bahasa pemrograman C++ merupakan pengembangan lebih lanjut dari bahasa pemrograman C untuk mendukung pemrograman berorientasi objek.
- Bahasa Pemrograman (web) PHP
PHP dibuat pertama kali oleh seorang perekayasa perangkat lunak (software engineering) yang bernama Rasmus Lerdoff. Rasmus Lerdoff membuat halawam web PHP pertamanya pada tahun 1994. PHP4 dengan versi-versi akhir menuju PHP5 sudah mendukung pemrograman berorientasi objek PHP merupakan bahasa pemrograman yang digunakan untuk pemrograman web.
- Bahasa pemrograman Java
Java dikembangkan oleh perusahaan Sun Microsystem. Java menurut definisi dari Sun Microsystem adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer standalone ataupun pada lingkungan jaringan, Java 2 adalah generasi kedua dari Java Platform.

Java berdiri di atas sebuah mesin interpreter yang diberi nama Java Virtual Machine (JVM). JVM inilah yang akan membaca bytecode dalam file .class dari suatu program sebagai representasi langsung program yang berisi bahasa mesin. Oleh karena itu bahasa Java disebut sebagai bahasa pemrograman yang portable karena dapat dijalankan pada berbagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM.

Java merupakan bahasa pemrograman objek murni karena semua kode programnya dibungkus dalam kelas. Saat ini Sun Microsystems sudah diakuisisi Oracle Corporation sehingga pengembangan Java diteruskan oleh Oracle Corporation.

7.2 Konsep Dasar Berorientasi Objek

Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya). Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.

Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dari abstraksi objek, kelas, hubungan antarkelas sampai abstraksi sistem. Saat mengabstraksikan dan memodelkan objek, data dan proses-proses yang dimiliki oleh objek akan dienkapsulasi (dibungkus) menjadi satu kesatuan.

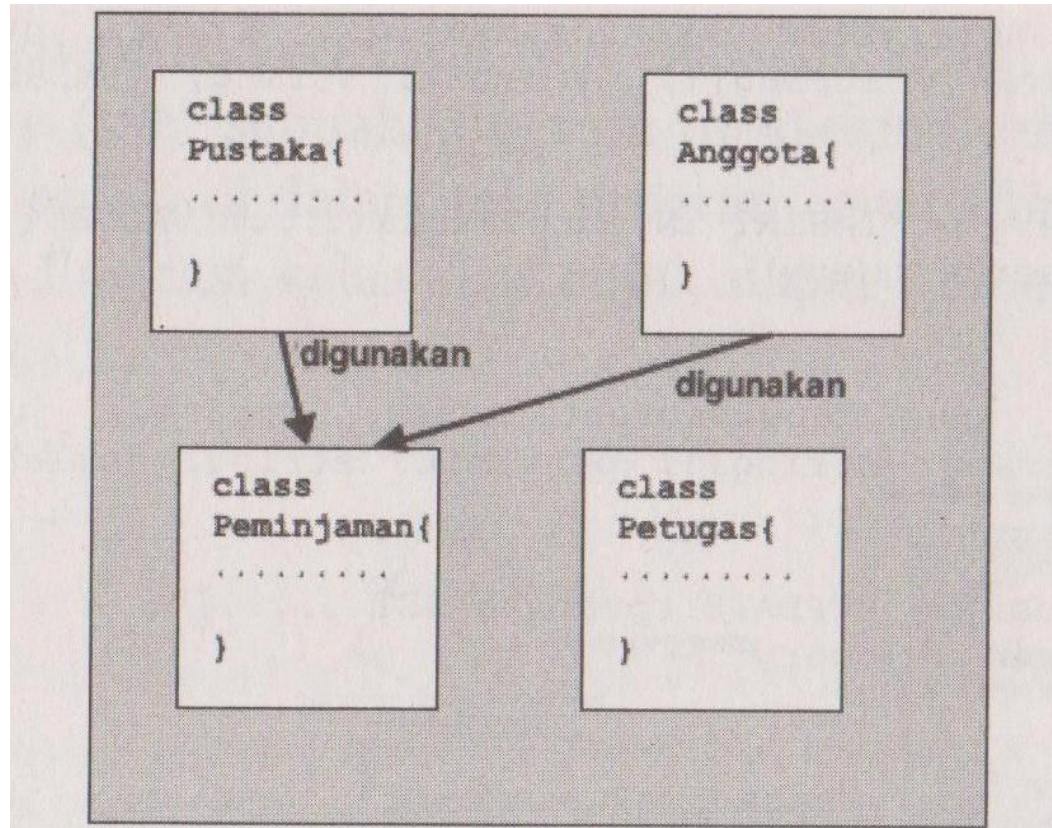
Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman, dan pengujian perangkat lunak. Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu.

Sistem berorientasi objek merupakan sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (enkapsulasi) menjadi kelompok data dan fungsi. Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dan komponen lainnya, dan dapat berinteraksi satu sama lain.

Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek:

- **Kelas (class)**
Kelas adalah kumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statik dan himpunan objek yang sama yang mungkin lahir atau diciptakan dari kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (operasi/metode), hubungan (relationship) dan arti. Suatu kelas dapat diturunkan dari kelas lain, dimana atribut dan kelas semula dapat diwariskan ke kelas yang baru.

Secara teknis kelas adalah sebuah struktur tertentu dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek. Ilustrasi dari sebuah kelas dapat dilihat pada gambar berikut



gambar 24.ilustrasi kelas

Kelas secara fisik adalah berkas atau file yang berisi kode program, di mana kode program merupakan semua hal yang terkait dengan nama kelas, Berikut adalah bentuk fisik dari sebuah kelas dalam pseudocode:

```

class Pustaka{

//atribut
id : string
judul : string
pengarang : string[]
penerbit : string
jumlah : int
tahun : int

//metode
procedure setld(id: string){
this.id = id
}
function getld(){
return id
}

procedure queryCariPustaka (kata_kunci: string, kategori:
string) {

//proses
query = "SELECT * FROM INTO tpustaka WHERE ..."
execute (query)
}
  
```

```

procedure queryMenghapusPustaka(id: string, judul: string,
pengarang: string[], penerbit: string, jumlah: int, tahun:
int) {

//proses
query = "DELETE FROM tpustaka WHERE ...."
execute (query)
}
procedure melihatPustaka() {

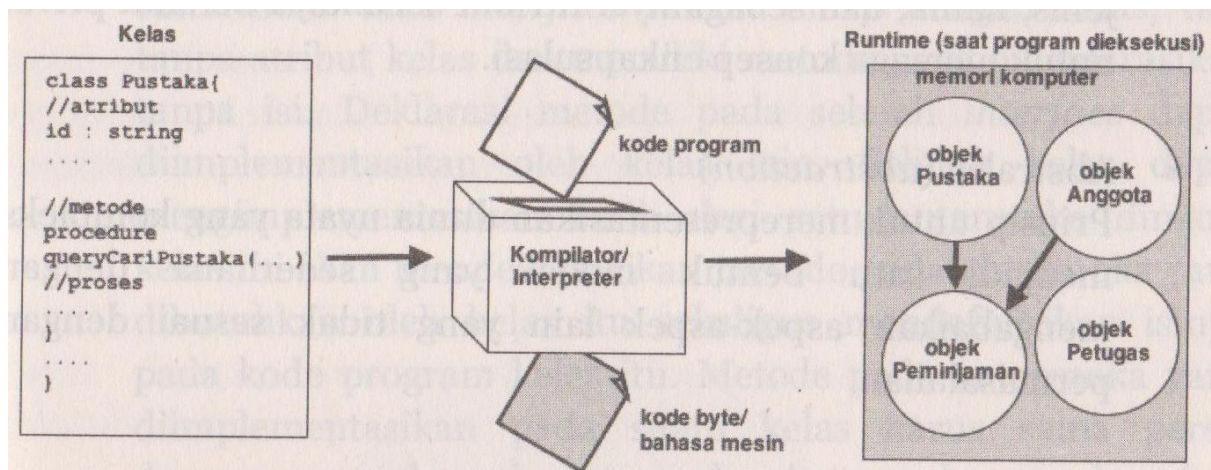
//proses
query = "SELECT * FROM tpustaka ....." execute (query)
}
}

```

- **Objek (object)**

Objek adalah abstraksi dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, struktur, status, atau hal-hal ini yang bersifat abstraksi. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan.

Secara teknis, sebuah kelas saat program dieksekusi maka akan dibuat sebuah objek. Objek dilihat dari segi teknik adalah elemen pada saat runtime yang akan diciptakan, dimanipulasi, dan dihancurkan saat ekspresi sehingga sebuah objek hanya ada saat sebuah program dieksekusi, jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat runtime (saat sebuah program dieksekusi), yang kita punya adalah objek, di dalam teks program yang kita lihat hanyalah kelas. Ilustrasi kelas dan objek dapat dilihat pada gambar berikut.



gambar 25. Ilustrasi kelas dan objek

- **Metode (method)**

Operasi atau metode atau method pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk

memanipulasi objek itu sendiri. Operasi atau metode merupakan fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek. Metode atau operasi dapat berasal dari event, aktivitas atau aksi keadaan, fungsi, atau kelakuan dunia nyata. Contoh metode atau operasi misalnya Read, Write, Move, Copy, dan sebagainya. Kelas sebaiknya memiliki metode get dan set untuk setiap atribut agar konsep enkapsulasi tetap terjaga. Metode get digunakan untuk memberikan akses kelas lain dalam mengakses atribut, agar kelas lain tidak mengakses atribut secara langsung.

- **Atribut (attribute)**
Atribut dari sebuah kelas adalah variable global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya. Atribut sebaiknya bersifat privat untuk menjaga konsep enkapsulasi.
- **Abstraksi (abstraction)**
Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.
- **Enkapsulasi (encapsulasi)**
Pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerjanya.
- **Pewarisan (inheritance)**
Mekanisme yang memungkinkan satu objek mewariskan sebagian atau seluruh definisi dan objek lain sebagai bagian dan dirinya
- **Antarmuka (interface)**
Antarmuka atau interface sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah interface dapat diimplementasikan oleh kelas lain. Sebuah kelas dapat mengimplementasikan lebih dari satu antarmuka dimana kelas ini akan mendeklarasikan metode pada antarmuka yang diimplementasikan pada suatu kelas harus sama persis dengan yang ada pada antarmuka. Antarmuka atau interface biasanya digunakan agar kelas yang lain tidak mengakses langsung ke suatu kelas, mengakses antarmukanya.
- **Reusability**
Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

Misalkan dalam sebuah aplikasi peminjaman buku diperlukan kelas anggota, masa ketika membuat aplikasi penyewaan VCD. Kelas anggota ini bisa digunakan kembali dengan sedikit perubahan untuk aplikasi penyewaan VCD tanpa harus membuat dari awal kembali.

- **Generalisasi dan Spesialisasi**
Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalny kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil, motor, dan kereta.
- **Komunikasi Antar Objek**
Komunikasi antar-objek dilakukan lewat pesan (message) yang dikirim dari satu objek ke objek lainnya.
- **Polimorfisme (polymorphism)**
Kemampuan suatu objek untuk digunakan di banyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.
- **Package**
Package adalah sebuah kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam package yang berbeda.

7.3 Perbandingan Pendekatan OO dan Terstruktur

Perbedaan yang paling dasar dari pendekatan terstruktur dan pendekatan OO (Object Oriented) atau berorientasi objek adalah pada metode berorientasi fungsi atau aliran data/Data Flow Diagram (DFD) / pendekatan terstruktur, dekomposisi permasalahan dilakukan berdasarkan fungsi atau proses secara hirarki, mulai dari konteks sampai proses-proses yang paling kecil. Sementara pada metode berorientasi objek, dekomposisi permasalahan dilakukan berdasarkan objek-objek yang ada dalam sistem.

Pendekatan terstruktur berkembang cukup pesat pada tahun 1990an karena pendekatan terstruktur mempunyai kelebihan dalam kemudahan untuk memahami sistem. Konsep tentang dekomposisi permasalahan mulai dari paling dasar (diagram konteks) sampai paling detail cukup memudahkan dalam pemahaman bentuk sistem khususnya bagi user yang mempunyai pemahaman tentang sistem cukup rendah.

Pendekatan berorientasi objek saat ini berkembang cukup pesat karena mempunyai kelebihan dalam peningkatan produktivitas karena mempunyai reusability yang cukup tinggi dibanding dengan pendekatan lain. Kelas-kelas dalam pemograman berorientasi objek dapat dengan mudah dimanfaatkan untuk sistem lain yang dikembangkan.

Latihan Soal

1. Sebutkan kelebihan dan kekurangan metodologi berorientasi objek
2. Jelaskan secara singkat sejarah perkembangan metodologi berorientasi objek
3. Jelaskan yang dimaksud dengan:
 - a. Kelas
 - b. Objek
 - c. Method
 - d. Atribut
 - e. Interface
 - f. Package
4. Mengapa saat ini metodologi berorientasi objek berkembang lebih pesat dibandingkan dengan metode-metode yang lain dalam bidang rekayasa perangkat lunak?
5. Sebutkan paradigm lain dalam bahasa pemrograman selain paradigma berorientasi objek, bandingkan dan sebutkan masing-masing kekurangan dan kelemahannya
6. Mengapa bahasa pemrograman PHP generasi terbaru berkembang menjadi bahasa pemrograman berorientasi objek.