

PERTEMUAN 12

(STUDI KASUS)

A. TUJUAN PEMBELAJARAN

Pada pertemuan ini Mahasiswa harus mampu dalam :

1. Alur Pemodelan Pada Perangkat Lunak
2. Perancangan Perangkat Lunak

B. URAIAN MATERI

1. Use Case Diagram

Diagram use case merupakan proses menghubungkan diagram antara sistem dan actor. Diagram use case dapat menjelaskan suatu interaksi antara satu atau lebih sistem dan actor yang akan dibuat, diagram use case dapat digunakan juga sebagai fungsi untuk mengetahui apa saja yang telah ada didalam sebuah sistem dan dapat digunakan untuk mempresentasikan suatu interaksi sistem dan actor. Komponen tersebut lalu menjelaskan interaksi yang terjadi pada suatu sistem dan actor yang telah ada, maka dari itu sebuah use case bisa dipresentasikan dengan perhitungan yang sederhana agar memudahkan user memahami alur yang telah dibuat.

Dalam sebuah use case ada tiga komponen yang harus diketahui, yaitu :

a. System

Menunjukkan suatu batasan pada sistem untuk menghubungkan dengan actor yang akan digunakan diluar sistem dengan fungsi yang sudah ada di dalam sistem

b. Actor

Actor merupakan suatu hal yang ada diluar sistem yang akan menggunakan sistem tersebut untuk mengerjakan sesuatu seperti manusia, sistem dan device yang memiliki peranan dalam keberhasilan operasi suatu sistem

c. Use case

Use case sendiri merupakan gambaran sebuah fungsi pada suatu sistem. Maka dari itu user akan memahami sistem tersebut mengenai apa saja fungsi

dan kegunaan pada perangkat yang dibuat

a. Simbol Use Case

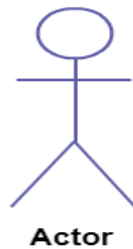
1) Use Case



Gambar 12. 1 Simbol Usecase

Merupakan gambaran sebuah fungsi yang sudah terdapat di dalam sistem sebagai unit yang saling bertukar pesan antar unit lain ataupun dengan actor. Pada saat pemberian nama di dalam use case akan diberikan dengan menggunakan kata kerja di awal frase pada nama use case.

2) Aktor



Gambar 12. 2 Simbol Actor

Aktor merupakan pengguna, proses, atau sistem yang akan berinteraksi dengan sistem informasi yang akan di buat, jadi walaupun simbol actor seperti menggambarkan seorang pengguna belum tentu itu adalah pengguna bisa saja sebuah sistem ataupun proses.

3) Asosiasi

**Gambar 12. 3** Simbol Asosiasi

Asosiasi adalah komunikasi antara aktor dan use case yang berpartisipasi pada use case diagram atau use case yang memiliki interaksi dengan aktor. Asosiasi merupakan sebuah simbol yang dapat menghubungkan antar element

4) Extend

**Gambar 12. 4** Simbol Extend

Extend adalah relasi use case tambahan ke sebuah use case dimana use case yang di tambahkan dapat berdiri sendiri meski tanpa use case tambahan itu. Arah panah mengarah pada use case yang ditambahkan

5) Include

**Gambar 12. 5** Simbol Include

Include adalah relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan membutuhkan use case ini untuk menjalankan fungsi sebagai syarat untuk menjalankan use case ini. Arah panah include mengarah pada use case yang dipakai atau mengarah pada use case yang ditambahkan

6) Generalisasi

**Gambar 12. 6** Simbol Generalisasi

Hubungan generalisasi dan spesialisasi antara dua buah use case dimana fungsi yang satu merupakan fungsi yang lebih umum dari fungsi lainnya

Arah panah mengarah pada use case yang menjadi generalisasinya

b. Langkah pembuatan pada Use Case Diagram

Dalam pembuatan use case terdapat langkah yang harus diperhatikan dengan baik agar tepat pada sasaran pengguna perangkat lunak, yaitu :

- 1) Mengidentifikasi para pelaku bisnis
- 2) Mengidentifikasi use case sebagai persyaratan bisnis
- 3) Membuat use case sebagai model use case
- 4) Mendokumentasikan naratif use case sebagai persyaratan bisnis

Adapun panduan yang harus diperhatikan dalam membangun diagram use case adalah :

- 1) Set konteks untuk target sistem
- 2) Mengidentifikasi semua aktor
- 3) Mengidentifikasi semua use case
- 4) Mendefinisikan semua asosiasi dari setiap aktor dan use case
- 5) Mengevaluasi setiap aktor dan use case untuk mendapatkan kemungkinan perbaikan
- 6) Mengevaluasi setiap use case untuk dependensi include
- 7) Mengevaluasi setiap use case untuk dependensi exclude
- 8) Mengevaluasi setiap aktor dan use case untuk generalisasi

c. Activity Diagram

Activity diagram merupakan diagram yang dapat memodelkan proses yang terjadi pada sebuah sistem. Runtutan dari suatu sistem digambarkan secara vertikal. Activity diagram adalah pengembangan dari use case yang memiliki alur aktifitas.

Alur atau aktifitas bisa berupa urutan menu-menu atau proses bisnis yang terdapat dalam sistem tersebut.






Aktifiti diagram mesti digunakan sejajar dengan tehnik pemodelan lainnya, seperti diagram use case dan diagram state. Aktifiti diagram dapat digunakan agar dapat memodelkan alur kerja sistem dengan baik. Aktifiti diagram juga berfungsi untuk menganalisa diagram use case dengan cara mendeskripsikan aktor, tindakan yang perlu dilakukan dan kapan harus terjadi. Diagram ini menggambarkan sebuah algoritma dan pemodelan sequensial yang kompleks dengan proses paralel.

Berikut ini adalah fungsi dan tujuan dari aktifiti diagram :

- 1) Menjelaskan urutan aktifitas dalam suatu proses
- 2) Di dalam dunia bisnis biasanya digunakan untuk memperlihatkan urutan proses pada bisnis
- 3) Mudah dalam memahami proses yang ada dalam sistem secara keseluruhan
- 4) Merupakan metode perancangan yang terstruktur, mirip dengan flowchart maupun data flow diagram
- 5) Mengetahui aktifitas aktor atau pengguna berdasarkan use case pada diagram use case yang dibuat sebelumnya

Berikut ini adalah simbol pada activity diagram :

Tabel 12. 1 Komponen usecase Diagram

Nama	Simbol	Fungsi
Initial State		Menggambarkan awal dimulainya suatu aliran aktivitas
Final State		Menggambarkan berakhirnya suatu aliran aktivitas
Activity		Menggambarkan aktivitas yang dilakukan dalam suatu aliran aktivitas
Decision		Menggambarkan pilihan kondisi atau cabang-cabang aktivitas tertentu
Transition		Berguna untuk menghubungkan satu komponen dengan komponen lainnya.

2. Perancangan Terstruktur

Desain terstruktur adalah konseptualisasi masalah menjadi beberapa elemen solusi yang terorganisir dengan baik. Ini pada dasarnya berkaitan dengan desain solusi. Manfaat dari desain terstruktur adalah, memberikan pemahaman yang lebih baik tentang bagaimana masalah ini diselesaikan. Desain terstruktur juga membuat desainer lebih mudah berkonsentrasi pada masalah dengan lebih akurat. Desain terstruktur sebagian besar didasarkan pada strategi 'membagi dan menaklukkan' di mana masalah dipecah menjadi beberapa masalah lebih kecil dan setiap masalah kecil diselesaikan secara terpisah sampai seluruh masalah diselesaikan. Potongan-potongan kecil masalah diselesaikan dengan menggunakan modul solusi. Penekanan desain terstruktur bahwa modul-modul ini diatur dengan baik untuk mencapai solusi yang tepat. Modul-modul ini disusun dalam hierarki. Mereka berkomunikasi satu sama lain. Desain terstruktur yang baik selalu mengikuti beberapa aturan untuk komunikasi di antara banyak modul, yaitu :

a. Kohesi

Keberadaan kohesi diperlukan untuk menentukan seberapa dekat dengan elemen – elemen suatu modul yang terkait. Kohesi suatu modul menunjukkan betapa terikat eratnya elemen-elemen internal modul satu sama lain. Kohesi suatu modul memberi perancang gagasan tentang apakah elemen-elemen yang berbeda dari suatu modul menjadi satu dalam modul yang sama. Kohesi dan kopling jelas terkait. Biasanya semakin besar kohesi masing-masing modul dalam sistem, semakin rendah kopling antar modul. Ada beberapa tingkatan kohesi :

1) Coincidental

Terjadi jika program yang ada di modulasi dengan memotongnya menjadi beberapa bagian dan membuat modul yang berbeda

2) Logical

Jika ada beberapa hubungan logis antara elemen – elemen modul, dan elemen – elemen melakukan fungsi yang mengisi kelas logis yang sama

3) Temporal

Kohesi temporal sama dengan kohesi logis, kecuali bahwa unsur-unsurnya juga berkaitan dalam waktu dan dieksekusi bersama.

Modul yang seperti ini biasanya bersifat sementara

4) Procedural

berisi elemen-elemen yang dimiliki oleh unit prosedural umum. Misalnya, loop atau urutan pernyataan keputusan dalam suatu modul dapat digabungkan untuk membentuk modul terpisah.

5) Communicational

Memiliki elemen yang terkait dengan referensi ke input atau output data yang sama. Yaitu dalam modul yang terikat secara komunikasi elemen-elemennya bersama karena mereka beroperasi pada input atau output data yang sama

6) Sequential

Memiliki elemen yang bersama – sama pada modul karena output yang satu membentuk input ke yang lain

7) Functional

Kohesi functional adalah kohesi terkuat. Dalam modul yang terikat secara fungsional, semua elemen modul terkait dengan melakukan fungsi tunggal.

b. Coupling

Dua modul dianggap independen jika satu dapat berfungsi sepenuhnya tanpa kehadiran yang lain. Jelas, jika dua modul independen, mereka dapat dipecahkan dan dimodifikasi secara terpisah. Namun, semua modul dalam suatu sistem tidak dapat independen satu sama lain, karena mereka harus berinteraksi sehingga mereka bersama-sama menghasilkan perilaku eksternal yang diinginkan dari sistem. Semakin banyak koneksi antar modul, semakin tergantung mereka dalam arti bahwa lebih banyak pengetahuan tentang satu modul diperlukan untuk memahami atau menyelesaikan modul lainnya. Oleh karena itu, semakin sedikit dan semakin sederhana koneksi antar modul, semakin mudah untuk memahami satu modul tanpa harus memahami yang lain. Kopling antar modul adalah kekuatan interkoneksi antar modul atau ukuran independensi antar modul. Untuk mengatasi dan memodifikasi modul secara terpisah, diinginkan agar modul tersebut digabungkan secara fleksibel dengan modul lainnya. Pilihan modul menentukan sambungan antar modul. Kopling adalah konsep abstrak dan tidak mudah diukur. Jadi, tidak ada rumus yang dapat diberikan untuk menentukan sambungan antara dua modul.

3. Perancangan Berorientasi Fungsi

Dalam desain berorientasi fungsi, sistem ini terdiri dari banyak subsistem yang lebih kecil yang dikenal sebagai fungsi. Fungsi-fungsi ini mampu melakukan tugas penting dalam sistem. Sistem dianggap sebagai kombinasi dari semua fungsi. Desain berorientasi fungsi mewarisi beberapa bagian dari desain terstruktur di mana adanya aktifitas membagi dan menyelesaikan metodologi yang digunakan. Mekanisme desain ini membagi keseluruhan sistem menjadi fungsi yang lebih kecil, yang menyediakan sarana abstraksi dengan menyembunyikan informasi dan operasinya. Modul-modul fungsional ini dapat berbagi informasi di antara mereka sendiri dengan menggunakan informasi yang lewat dan menggunakan informasi yang tersedia secara global. Karakteristik lain dari fungsi adalah apabila sebuah program memanggil fungsi, maka fungsi mengubah keadaan program, yang terkadang tidak dapat diterima oleh modul lain. Desain berorientasi fungsi, berfungsi dengan baik di mana status sistem tidak penting dan program/fungsi bekerja pada input pada suatu bagian.

Berikut ini adalah proses perancangan berorientasi fungsi :

- a. Keseluruhan system dilihat sebagai bentuk bagaimana data mengalir dalam system melalui DFD
- b. DFD menggambarkan bagaimana fungsi mengubah data dan keadaan system secara keseluruhan
- c. Seluruh system secara logis dipecah menjadi unit yang lebih kecil yang dikenal sebagai fungsi atau dasar operasi di dalam sistem
- d. Setiap fungsi kemudian dijelaskan secara lengkap

a. Perancangan Berbasis Obyek

Dalam metode perancang ini kita mengambil pandangan yang sedikit berbeda dari pengembangan sistem dari apa yang telah kita bahas sebelumnya. Pada bagian sebelumnya, kita cenderung melihat sistem dari sudut pandang fungsional atau berbasis proses, dan menghubungkannya dengan struktur data. Sementara pendekatan ini menghasilkan sistem kerja yang dirancang dengan baik. Pandangan di antara praktisi ini merupakan sistem yang telah dihasilkan pada umumnya cenderung kaku dan

membuatnya sulit untuk merespon dengan cepat terhadap perubahan dalam kebutuhan pengguna.

Tidak seperti dua pendahulunya, pendekatan berorientasi objek menggabungkan data dan proses (disebut metode) menjadi entitas tunggal yang disebut objek. Objek biasanya sesuai dengan hal-hal nyata yang ditangani sistem, seperti pelanggan, pemasok, kontrak, dan faktur. Model berorientasi objek mampu secara menyeluruh merepresentasikan hubungan yang kompleks dan untuk merepresentasikan data dan pemrosesan data dengan notasi yang andal. Hal ini memungkinkan campuran analisis dan desain yang lebih mudah dalam proses pertumbuhan aplikasi. Tujuan dari pendekatan Berorientasi Objek adalah untuk membuat elemen sistem lebih modular, sehingga meningkatkan kualitas sistem dan efisiensi analisis dan desain sistem.

Dalam pendekatan Berorientasi Objek, kita cenderung lebih fokus pada perilaku sistem. Fitur utama yang kami dokumentasikan adalah Object atau Class. Object Oriented Design (OOD) bekerja antar entitas dan karakteristiknya, bukan fungsi yang terlibat dalam sistem perangkat lunak. Strategi desain ini berfokus pada entitas dan karakteristiknya. Seluruh konsep solusi perangkat lunak bersiklus di areal entitas yang terlibat. Konsep-konsep pokok dalam konsep Object Oriented Design, adalah sebagai berikut :

1) Obyek

Semua entitas yang ikut dalam perancangan solusi dikenal sebagai obyek. Misalkan, universitas, perusahaan, pengguna diperlakukan sebagai obyek. Setiap entitas memiliki beberapa atribut yang terkait dengannya dan memiliki beberapa metode untuk dilakukan pada atribut

2) Class

Merupakan deskripsi umum dari suatu obyek. Obyek adalah turunan class. Class mendefinisikan semua atribut, yang bisa dimiliki obyek dan metode, yang mendefinisikan fungsionalitas obyek. Dalam desain solusi, atribut disimpan sebagai variabel dan fungsionalitas didefinisikan sebagai metode atau prosedur

3) Enkapsulasi

Dalam obyek orientasi design, atribut dan metode digabungkan bersama disebut enkapsulasi. Enkapsulasi tidak hanya memadukan informasi penting dari suatu objek secara bersamaan, tetapi juga membatasi akses data dan metode dari dunia luar. Ini disebut menyembunyikan informasi

4) Pewarisan (inheritance)

Obyek orientasi design memungkinkan kelas serupa untuk menumpuk secara berurutan dimana kelas bawah atau subkelas dapat mengimpor, menerapkan, dan menggunakan kembali variable dan metode yang diperbolehkan. Property obyek orientasi design ini dikenal sebagai warisan, ini membuatnya lebih mudah menentukan kelas secara spesifik dan untuk membuat kelas umum dari kelas spesifik

5) Polimorfisme

Bahasa obyek orientasi menyediakan mekanisme dimana metode yang melakukan tugas serupa tetapi banyak model dalam argumen, dapat diberi nama yang sama. Ini disebut polimorfisme yang memungkinkan satu antar muka melakukan tugas untuk berbagai jenis. Tergantung pada bagaimana fungsi di panggil dan masing – masing bagian dari kode dijalankan

Pada mekanisme obyek orientasi design cobalah untuk mengidentifikasi beberapa obyek umumdand kemudian lihat apakah anda dapat mengidentifikasi kelas yang ada. Dalam aplikasi perbankan otomatis, biasanya diharapkan sistem multi-level karena mengandung hal – hal berikut:

- 1) Antar muka pengguna
- 2) Perangkat lunak aplikasi untuk melakukan pemrosesan
- 3) Database

Di bagian antar muka, contoh objek bisa berupa menu, tombol, atau kotak dialog. Di tingkat perangkat lunak aplikasi, objek dapat berupa transaksi atau proses bisnis. Di tingkat model data objek akan menjadi tabel database. Setiap objek memiliki Identity (nama), State (data yang terkait dengannya) dan perilaku (hal-hal yang dapat kita lakukan untuk itu). Seperti metodologi terstruktur lainnya, dalam pengembangan berorientasi objek ada beberapa

metodologi dan tool yang tersedia untuk digunakan. Salah satu yang dapat digunakan adalah tool UML (Unified Modeling Language). Ini masih merupakan metode yang relatif baru dan mulai diminati oleh para pengembang perangkat lunak.

b. Pengertian UML (Unified modeling language

Unified modeling language merupakan bahasa berorientasi obyek dalam menentukan, memvisualisasikan, membangun, dan mendokumentasikan artefak sistem perangkat lunak, serta untuk pemodelan bisnis.

Dengan menawarkan bahasa pengembangan umum, Unified Modelling Language membebaskan pengembang dari ikatan kepemilikan yang begitu umum dalam rekayasa perangkat lunak dan dapat membuat sistem pengembangan menjadi sulit dan mahal.

UML memiliki beberapa diagram yang dapat kita gunakan untuk memodelkan suatu sistem, tetapi minimum yang diperlukan adalah :

1) Diagram kelas (class)

Untuk menentukan obyek dalam sistem dan tautan apapun diantara mereka. Ini adalah alat yang berguna pada tahapan menganalisa atau design

2) Diagram use case

Untuk membantu memahami apa yang dilakukan sistem dan siapa yang berinteraksi dengannya. Ini paling berguna untuk menunjukkan tujuan sistem

3) Sequence diagram

Untuk memahami apa yang dilakukan sistem dan memeriksa cara sistem berperilaku

4) Model data

untuk menangkap data. Ini biasanya lebih bermanfaat bagi tahap implementasi model siklus hidup.

Seiring dengan meningkatnya perangkat lunak yang mempunyai nilai strategis bagi banyak perusahaan, kalangan industry mencari teknik untuk

mengotomatisasi produksi perangkat lunak dan meningkatkan kualitas serta mengurangi biaya dan waktu. Teknik yang terdapat ini termasuk dalam teknologi pemrograman visual, pola, komponen dan kerangka kerja. Pada bagian bisnis juga mencari teknik untuk mengelola sistem yang kompleks agar meningkatkan ruang lingkup dan skala. Secara khusus mereka mengakui perlunya menyelesaikan masalah arsitektur berulang, seperti distribusi fisik, keamanan, replikasi, keseimbangan beban, dan toleransi kesalahan. UML dirancang untuk memenuhi kebutuhan ini.

c. Proses perancangan

Proses desain perangkat lunak dapat dianggap sebagai serangkaian langkah yang terdefinisi dengan baik. Meskipun bervariasi sesuai dengan pendekatan desain (berorientasi fungsi atau berorientasi objek), namun ia memiliki langkah-langkah berikut:

- 1) Desain solusi dibuat berdasarkan kebutuhan atau sistem yang digunakan sebelumnya dan diagram urutan sistem
- 2) Obyek diidentifikasi dan dikelompokkan kedalam kelas atas kesamaan nama dalam satu karakteristik atribut
- 3) Urutan kelas dan hubungan diantaranya di definisikan
- 4) Kerangka kerja aplikasi di definisikan

d. Pendekatan pada perancangan perangkat lunak

Pada pendekatan perancangan perangkat lunak diketahui ada dua macam yaitu :

1) Pendekatan Top Down

Suatu sistem terdiri dari lebih satu sub-sistem dan mengandung sejumlah komponen. Selanjutnya, sub-sistem dan komponen ini mungkin memiliki sub-sistem dan komponennya sendiri, dan menciptakan struktur urutan di dalam sistem. Desain top-down mengambil seluruh sistem perangkat lunak sebagai satu kesatuan dan kemudian menguraikannya untuk mencapai lebih dari satu sub-sistem atau komponen berdasarkan beberapa karakteristik. Setiap sub-sistem atau komponen kemudian diperlakukan sebagai suatu sistem dan didekomposisi lebih lanjut. Proses ini terus berjalan hingga tingkat sistem terendah dalam urutan top-down tercapai.

Desain top-down dimulai dengan model umum sistem dan terus mendefinisikan bagian yang lebih spesifik dari itu. Ketika semua komponen tersusun, seluruh sistem menjadi ada. Dalam bahasa yang sederhana pendekatan top down adalah membuat modul kompleks dibagi menjadi beberapa sub modul.

2) Pendekatan Bottom Up

Model design bottom up dimulai dengan komponen yang paling spesifik dan mendasar. Menyusun tingkat komponen yang lebih tinggi dengan menggunakan komponen tingkat dasar atau yang rendah. Sistem terus menciptakan komponen yang lebih tinggi sampai sistem yang diinginkan tidak berevolusi sebagai satu komponen tunggal. Dengan masing – masing tingkat yang lebih tinggi, jumlah abstraksi meningkat.

C. SOAL LATIHAN/TUGAS

1. Jelaskan proses pembuatan Diagram use case ?
2. Sebutkan fungsi dari generalisasi ?
3. Jelaskan fungsi dari extend ? berikan contohnya !
4. Jelaskan Tujuan dari pendekatan Berorientasi Objek untuk membuat elemen system ?
5. Jelaskan definisi class pada studi kasus ?

D. REFERENSI

1. Pressman, Roger S. Rekayasa Perangkat Lunak : Pendekatan Praktisi (Buku Satu). Yogyakarta : Andi Offset. 2002.
2. Ganong, William F. Pembelajaran Berbantuan Komputer (alih bahasa M. Djauhari Wijayakusumah) Edisi tiga, Jakarta : EGC. 2002.
3. Heinich, R, dkk. Instructional media and technology for learning. 7th edition. New Jersey: Prentice Hall, Inc. 2002.

GLOSARIUM

Kohesi keberadaan kohesi diperlukan untuk menentukan seberapa dekat dengan elemen – elemen suatu modul yang terkait.

DFD menggambarkan bagaimana fungsi mengubah data dan keadaan system secara keseluruhan.

Object Oriented Design (OOD) bekerja antar entitas dan karakteristiknya, bukan fungsi yang terlibat dalam sistem perangkat lunak.