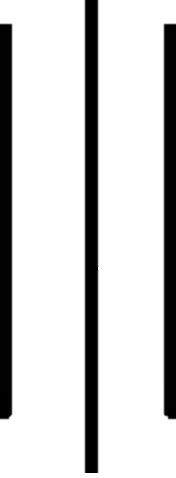
**MAKALAH**

**PERMODELAN**



**DISUSUN OLEH:**

**NAMA: JASON DEVITO**

**KELAS: XI-RPL**

# KATA PENGANTAR

Puji Syukur saya panjatkan kepada Tuhan Yang Maha Esa karna telah melimpahkan berkat dan rahmatnya sehingga saya dapat menyelesaikan tugas makalah Permodelan yang saya pelajari di jurusan RPL (Rekayasa Perangkat Lunak)

Makalah ini saya susun untuk memperluas ilmu pengetahuan para pembaca tentang segala jenis materi yang terdapat dalam memodelkan suatu kinerja System Berorientasi Objek. Makalah ini disusun dengan mengambil materi dari berbagai sumber yang tersedia, beserta dari apa yang saya pahami Ketika berada di jenjang SMK ini.

Semoga makalah ini dapat memberikan wawasan yang cukup bagi sang pembaca. Walau makalah ini penuh dengan kekurangan, saya mohon akan saran dan kritiknya dari pembaca makalah ini. Terimakasih.

# DAFTAR ISI

[KATA PENGANTAR i](#_Toc151331385)

[DAFTAR ISI ii](#_Toc151331386)

[BAB 1 1](#_Toc151331387)

[1.1 Pendahuluan 1](#_Toc151331388)

[1.2 Rumusan Masalah 2](#_Toc151331389)

[1.3 Tujuan 2](#_Toc151331390)

[BAB 2 3](#_Toc151331391)

[2.1 UML 3](#_Toc151331392)

[2.2 Sejarah UML 3](#_Toc151331393)

[2.3 Diagram 4](#_Toc151331394)

[2.4 View 11](#_Toc151331395)

[2.5 Langkah membuat Software dengan bantuan UML 12](#_Toc151331396)

[2.6 Hubungan / Relationship 14](#_Toc151331397)

[2.7 Metodologi Pemrograman Berorientasi Objek 14](#_Toc151331398)

[2.8 Metode SDLC 16](#_Toc151331399)

[BAB 3 21](#_Toc151331400)

[3.1 Penutup 21](#_Toc151331401)

# BAB 1

## Pendahuluan

Saat ini, kebutuhan software (Perangkat lunak) semakin luas dan besar lingkupnya, sehingga tidak bisa lagi dibuat asal-asalan. Perangkat lunak saat ini seharusnya dirancang dengan memperhatikan hal-hal seperti *scalability, security,* dan eksekusi yang *robust* walaupun dalam kondisi yang sulit. Selain itu arsitekturnya harus didefinisikan dengan jelas, agar *bug* mudah ditemukan dan diperbaiki, bahkan oleh orang lain selain programmer aslinya. Keuntungan lain dari perencanaan arsitektur yang matang adalah dimungkinkannya penggunaan kembali modul atau komponen untuk aplikasi perangkat lunak lain yang membutuhkan fungsionalitas yang sama.

Pemodelan (modeling) adalah proses merancang perangkat lunak sebelum melakukan pengkodean (coding). Model perangkat lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin komplek sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik.

Dengan menggunakan model, diharapkan pengembangan perangkat lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti *scalability, robustness, security,* dan sebagainya.

Kesuksesan suatu pemodelan perangkat lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuan segitiga sukses (*the triangle for success*). Ketiga unsur tersebut adalah metode pemodelan (*notation*), proses (*process*) dan tool yang digunakan.

Di makalah kali ini, saya akan menjelaskan bagaimana cara menggunakan UML sebagai alat bantu untuk memodelkan suatu system berorientasi objek.

## Rumusan Masalah

Berdasarkan latar belakang di atas, dapat di rumuskan sebuah permasalahan sebagai berikut:

1. Bagaimana cara melakukan perancangan sistem berorientasi objek menggunakan alat bantu UML.
2. Apa saja jenis-jenis Diagram UML
3. Langkah membuat Diagram UML
4. Prinsip Pemrograman Berorientasi Objek
5. Metode SDLC

## Tujuan

1. Mengetahui tujuan penggunaan UML.
2. Mengetahui sejarah singkat UML.
3. Mengenal bagian-bagian (diagram-diagram) UML.
4. Menggunakan UML untuk membuat model sederhana.
5. Mengetahui konsep dasar Sistem Berorientasi Objek
6. Mengetahui tujuan dan fungsi SDLC

# BAB 2

## 2.1 UML

**Apa itu UML?** UML (Unified Modeling Language) adalah sebuah bahasa untuk menentukan, visualisasi, kontruksi, dan mendokumentasikan artifact (bagian dari informasi yang digunakan atau dihasilkan dalam suatu proses pembuatan perangkat lunak. Artifact dapat berupa model, deskripsi atau perangkat lunak) dari system perangkat lunak, seperti pada pemodelan bisnis dan system non perangkat lunak lainnya.

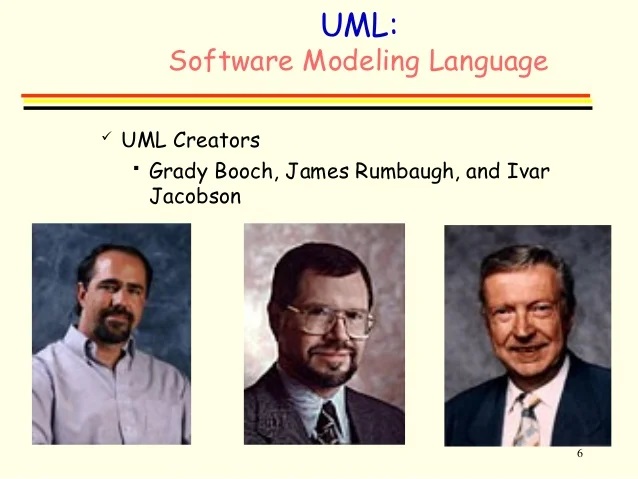
UML merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan system yang besar dan kompleks. UML tidak hanya digunakan dalam proses pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada perangkat keras, sistem operasi, dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun.

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan syntax /semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML syntax mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan.

## 2.2 Sejarah UML

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990, puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (method war) dalam pendesainan berorientasi objek. Masing masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.

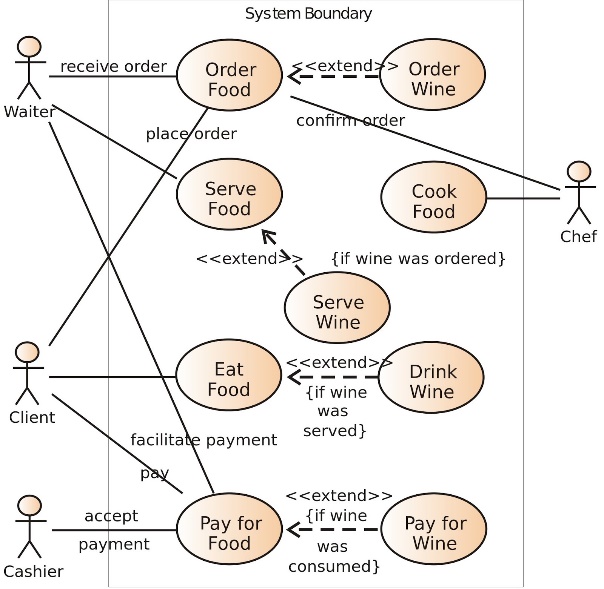
Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh Object Management Group (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.



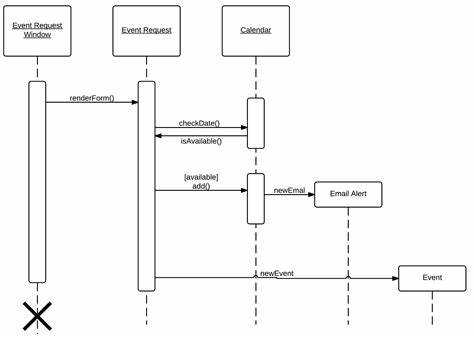
## 2.3 Diagram

Diagram di UML berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu view tertentu dan ketika digambarkan, biasanya dialokasikan untuk view tertentu. Adapun jenis diagram antara lain :

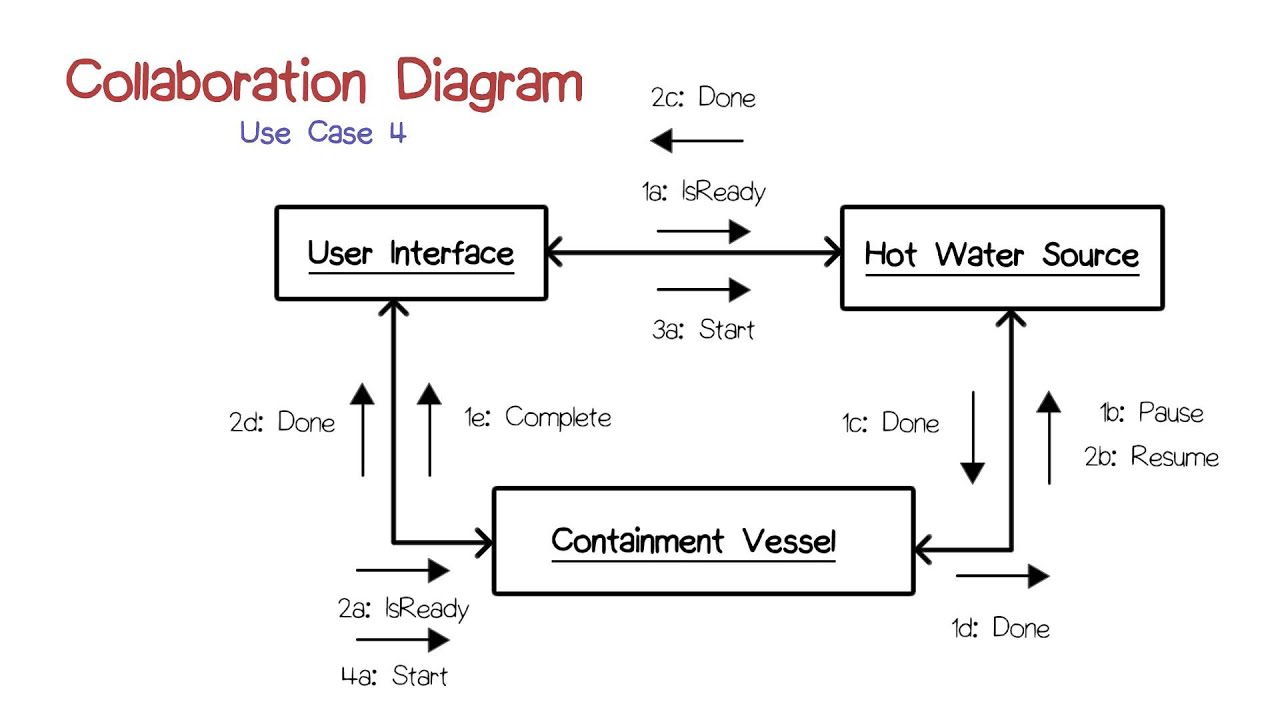
1. **Use Case Diagram** untuk memodelkan proses bisnis.



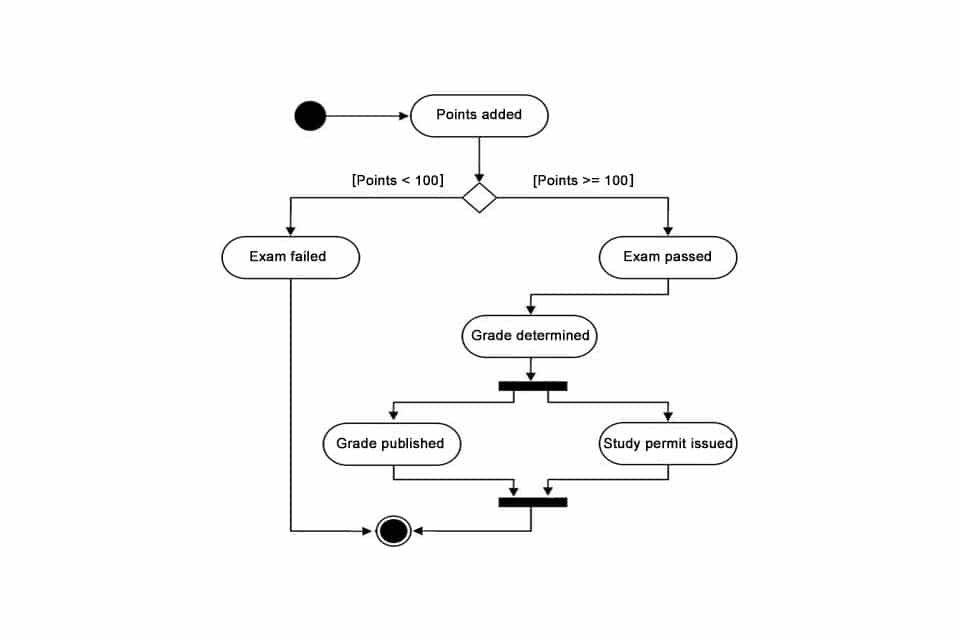
1. **Sequence Diagram** untuk memodelkan pengiriman pesan (message) antar objects.

****

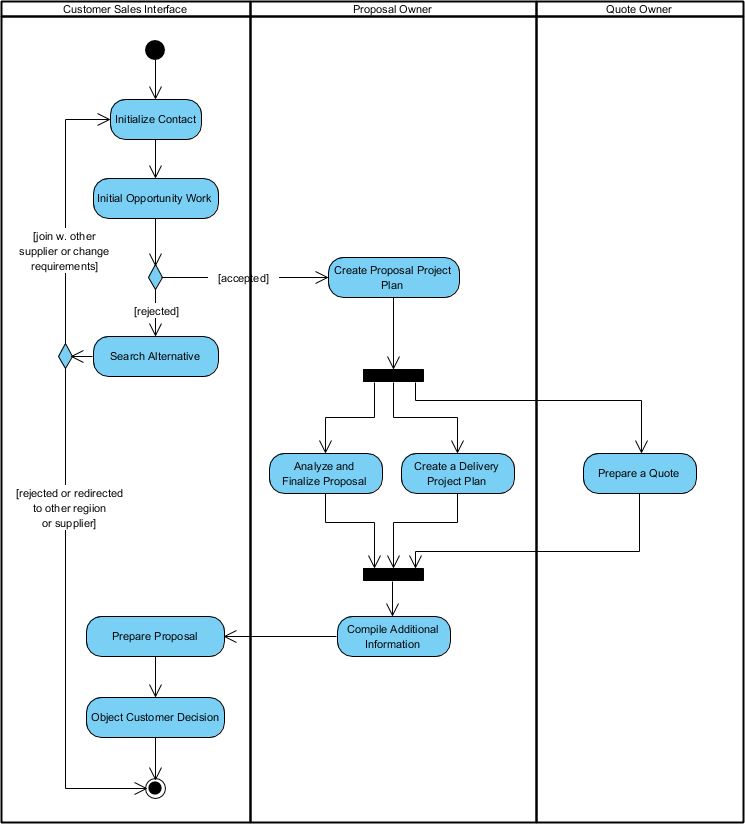
1. **Collaboration Diagram** untuk memodelkan interaksi antar objects.



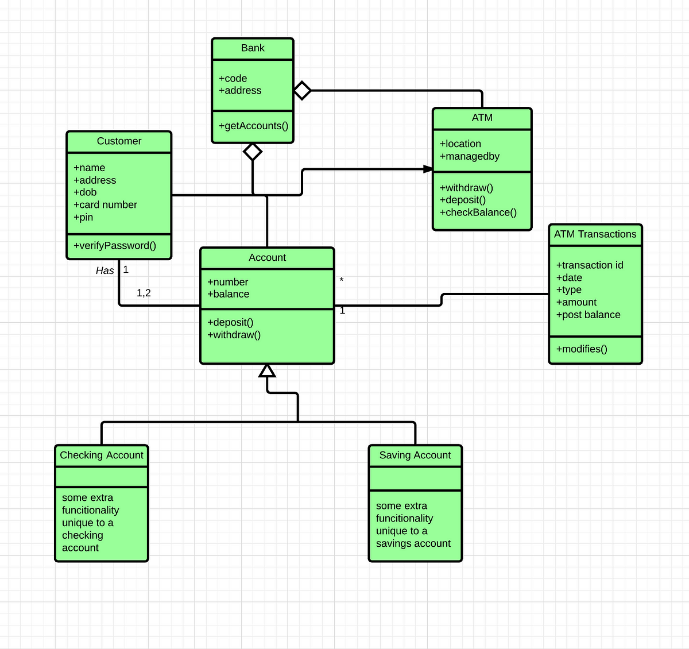
1. State Diagram untuk memodelkan perilaku objects di dalam sistem.



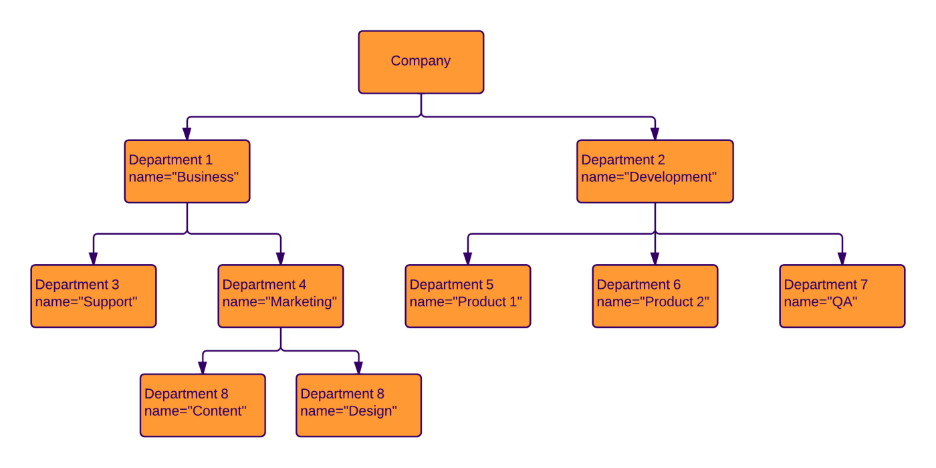
1. Activity Diagram untuk memodelkan perilaku Use Cases dan objects di dalam system.



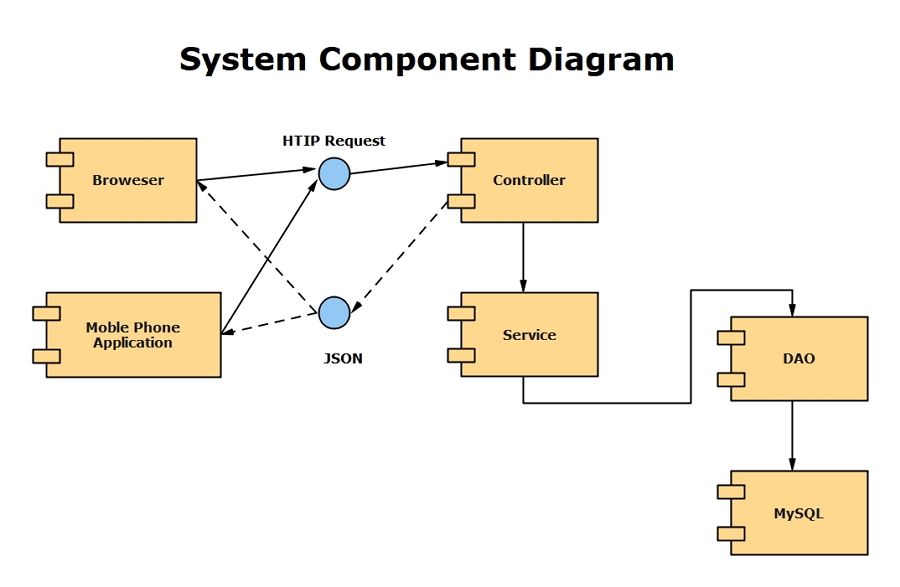
1. Class Diagram untuk menampilkan kelas-kelas maupun paket-paket yang ada pada suatu sistem yang nantinya akan digunakan



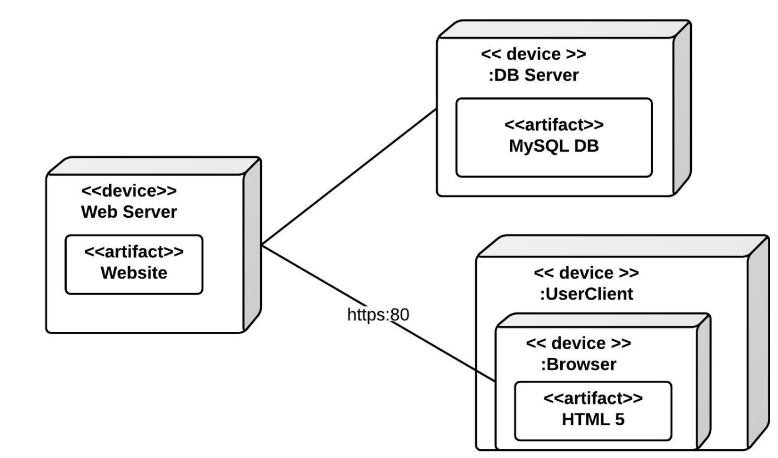
1. Object Diagram untuk menggambarkan objek-objek pada suatu sistem dan hubungan antarnya.



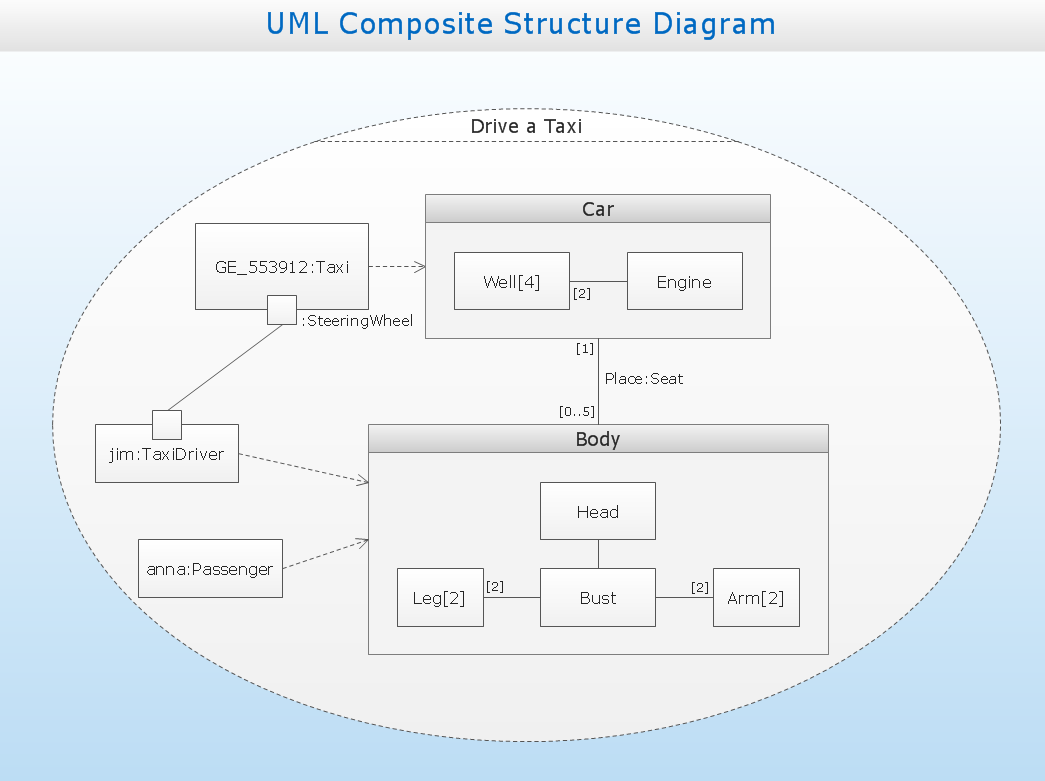
1. Component Diagram untuk memodelkan komponen object.



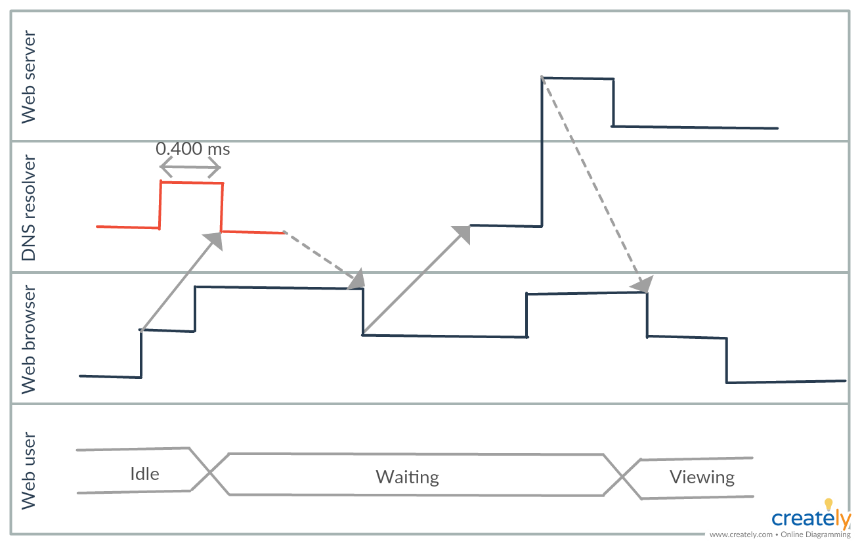
1. Deployment Diagram untuk memodelkan distribusi aplikasi, untuk menunjukan letak software pada hardware yang digunakan sistem.



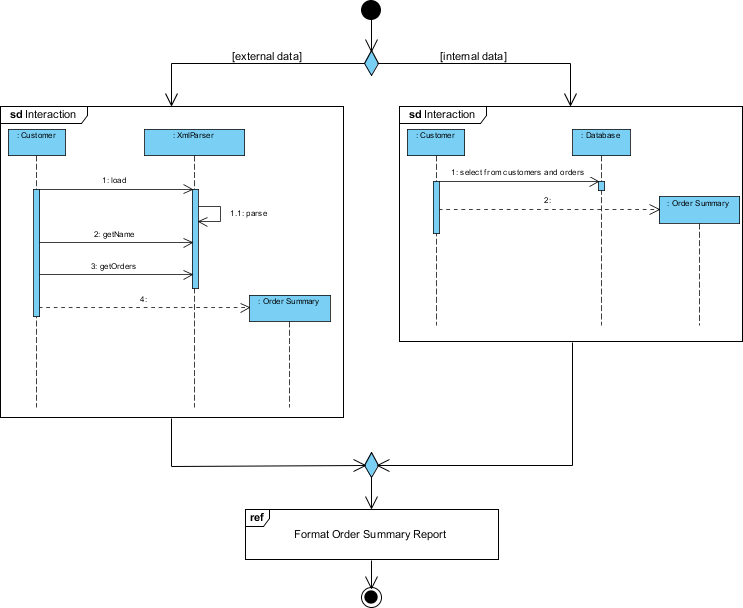
1. Composite Diagram untuk menggambarkan struktur internal dari penklasifikasian (class, component atau use case) dan termasuk titik-titik interaksi penklasifikasian kebagian lainnya dari suatu system



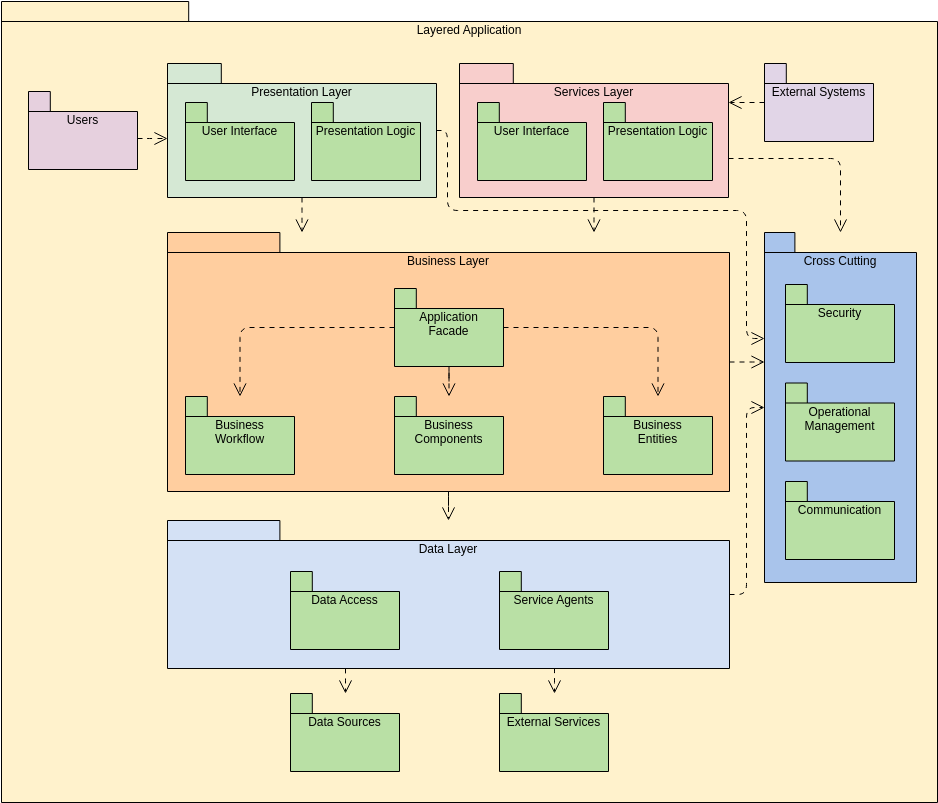
1. Timing Diagram untuk menunjukan faktor-faktor yang membatasi waktu antara perubahan state terhadap objek yang berbeda.



1. Interaction Overview Diagram untuk men-visualisasikan kerjasama dan hubungan antara activity diagram dengan sequence diagram



1. Package Diagram untuk mengelompokkan kelas dan juga menunjukan bagaimana elemen model akan disusun serta mengambarkan ketergantungan antara paket-paket.



## 2.4 View

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. View bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis view dalam UML antara lain: use case view, logical view, component view, concurrency view, dan deployment view. Berikut ini adalah penjelasannya:

1. Use case view

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan external actors. Actor yang berinteraksi dengan sistem dapat berupa user atau sistem lainnya. View ini digambarkan dalam use case diagram dan kadang-kadang dengan activity diagrams. View ini digunakan terutama untuk pelanggan, perancang (designer), pengembang (developer), dan penguji sistem (tester).

1. Logical view

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (class, object,dan relationship) dan kolaborasi dinamis yang terjadi ketika object mengirim pesan ke object lain dalam suatu fungsi tertentu. View ini digambarkan dalam class diagrams untuk struktur statis dan dalam state, sequence, collaboration, dan activity diagram untuk model dinamisnya. View ini digunakan untuk perancang (designer) dan pengembang (developer).

1. Component view

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari code module diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi administrative lainnya. View ini digambarkan dalam component view dan digunakan untuk pengembang (developer).

1. Concurrency view

Membagi sistem ke dalam proses dan prosesor.View ini digambarkan dalam diagram dinamis (state, sequence, collaboration, dan activity diagrams) dan diagram implementasi (component dan deployment diagrams) serta digunakan untuk pengembang (developer), pengintegrasi (integrator), dan penguji (tester).

1. Deployment view

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (nodes) dan bagaimana hubungannya dengan lainnya. View ini digambarkan dalam deployment diagramsdan digunakan untuk pengembang (developer), pengintegrasi (integrator), dan penguji (tester).

## 2.5 Langkah membuat Software dengan bantuan UML

Berikut ini adalah tips pengembangan Software dengan menggunakan UML:

1. Buatlah daftar business process dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan use case untuk tiap business process untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus use case diagram dan lengkapi dengan requirement, constraints dan catatan-catatan lain.
3. Buatlah deployment diagram secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan requirement lain (non-fungsional, security dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan use case diagram, mulailah membuat activity diagram.
6. Definisikan objek-objek level atas (package atau domain) dan buatlah sequence dan/atau collaboration diagram untuk tiap alir pekerjaan. Jika sebuah use case memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing-masing alir.
7. Buarlah rancangan user interface model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario use case.
8. Berdasarkan model-model yang sudah ada, buatlah class diagram. Setiap package atau domain dipecah menjadi hirarki class lengkap dengan atribut dan metodanya. Akan lebih baik jika untuk setiap class dibuat unit test untuk menguji fungsionalitas class dan interaksi dengan class lain.
9. Setelah class diagram dibuat, kita dapat melihat kemungkinan pengelompokan class menjadi komponen-komponen. Karena itu buatlah component diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.
10. Perhalus deployment diagram yang sudah dibuat. Detilkan kemampuan dan requirement piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam node.
11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :

* Pendekatan use case, dengan meng-assign setiap use case kepada tim pengembang tertentu untuk mengembangkan unit code yang lengkap dengan tes.
* Pendekatan komponen, yaitu meng-assign setiap komponen kepada tim pengembang tertentu.

1. Lakukan uji modul dan uji integrasi serta perbaiki model berserta codenya. Model harus selalu sesuai dengan code yang aktual.
2. Software siap dirilis.

## 2.6 Hubungan / Relationship

Ada 4 macam hubungan dalam penggunaan UML, yaitu;

1. Dependency, hubungan semantik antara dua objek yang mana sebuah objek berubah mengakibatkan objek satunya akan berubah pula.
2. Association, hubungan antar benda secara struktural yang terhubung diantara objek dalam kesatuan objek.
3. Generalizations, hubungan khusus dalam objek anak yang menggantikan objek induk . dan memberikan pengaruhnya dalam hal struktur dan tingkah lakunya kepada objek induk
4. Realizations, hubungan semantik antarpengelompokkan yang menjamin adanya ikatan diantaranya yang diwujudkan diantara interface dan kelas atau elements, serta antara use cases dan collaborations.

## 2.7 Metodologi Pemrograman Berorientasi Objek

object-oriented programming atau OOP adalah suatu metode pemrograman yang berorientasi pada objek. Program-program yang telah ada merupakan gabungan dari beberapa komponen-komponen kecil yang sudah ada sebelumnya. Hal itu dapat mempermudah pekerjaan seorang programmer dalam melakukan pengembangan program. Objek-objek yang saling berkaitan dan disusun kedalam satu kelompok ini disebut dengan class. Nantinya, objek-objek tersebut akan saling berinteraksi untuk menyelesaikan masalah program yang rumit.

Jika sebelumnya developer harus berfokus pada logic yang akan dimanipulasi, dengan OOP, developer dapat lebih terfokus pada objeknya saja untuk dimanipulasi. Pendekatan ini menawarkan cara yang mudah untuk menangani kerumitan suatu pemrograman. Di OOP ada juga istilah Variabel. Variabel adalah sebuah wadah untuk menampung nilai. Di sisi lain, Tujuan utama OOP adalah untuk mengatasi kelemahan pendekatan pemrograman konvensional. Banyak bahasa pemrograman populer yang menggunakan prinsip OOP, seperti Java dan Python.

Berikut ini adalah konsep-konsep dasar OOP:

1. Encapsulation

Encapsulation adalah salah satu konsep dasar OOP yang menyembunyikan detail suatu objek dari akses luar. Encapsulation membantu dalam menciptakan kode yang lebih aman dengan mencegah modifikasi data yang tidak disengaja. Encapsulation merupakan dasar untuk pembatasan ruang lingkup program terhadap data yang diproses. Data dan prosedur atau fungsi dikemas bersama – sama dalam suatu objek, sehingga prosedur atau fungsi lain dari luar tidak dapat mengaksesnya kecuali prosedur yang berada dalam objek itu sendiri.

1. Inheritance

Inheritance adalah mekanisme dalam OOP yang memungkinkan suatu objek mewarisi properti dan metode dari objek induknya. Inheritance memungkinkan terciptanya hubungan antara objek yang memiliki karakteristik dan perilaku yang sama. Inheritance adalah teknik yang menyatakan bahwa anak dari objek akan mewarisi data atau atribut dan metode dari induknya langsung. Atribut dan metode dari sebuah objek induk akan diturunkan kepada anak objek dan demikian seterusnya. Ini menandakan bahwa atribut dan operasi yang dimiliki secara bersama dalam sebuah kelas objek mempunyai hubungan secara hirarki. Metode ini menggambarkan generalisasi dari sebuah kelas objek. Dengan mengatur objek ke dalam subkelas berdasarkan karakteristik yang sama, seorang programmer dapat membuat basis kode yang dapat digunakan berulang kali.

1. Polymorphism

Polimorfisme merupakan konsep yang menyatakan bahwa sesuatu yang sama dapat mempunyai bentuk dan perilaku berbeda. Ini berarti bahwa operasi yang sama mungkin saja mempunyai perbedaan dalam kelas yang berbeda atau bergantung kepada kelas yang menciptakan objek tersebut.Polymorphism merupakan kemampuan objek untuk mengambil bentuk yang berbeda atau memiliki banyak perilaku, tergantung pada konteks penggunaannya. Polymorphism memfasilitasi pemrograman yang lebih fleksibel, karena objek dapat digunakan dalam berbagai konteks berbeda sambil tetap berperilaku yang sesuai.

1. Abstraction

Abstraction adalah praktik menyederhanakan sistem yang kompleks dengan memecahnya menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola. Dengan meringkas detail dari sebuah objek, seorang programmer dapat menyederhanakan suatu desain program.

## 2.8 Metode SDLC

SDLC (Software Development Life Cycle) adalah rangkaian proses yang dilakukan dalam industry perangkat lunak. Tujuan SDLC adalah menghasilkan perangkat lunak yang memenuhi keinginan pelanggan dan menggunakan waktu dan biaya secara efektif dan efisien. Di buku, terdapat penjelasan 2 metode SDLC, antara lain:

1. **Metode Waterfall**

Merupakan metode pengembangan perangkat lunak yang paling umum digunakan. Metode ini mempunyai pendekatan yang kaku. Setiap tahapan pengembangan harus diselesaikan secara keseluruhan sebelum melangkah pada tahap pengembangan berikutnya.

Metode Waterfall terdiri atas tahapan analisis kebutuhan, desain, implementasi, pengujian, dan pemeliharaan. Jika diperhatikan, tahapan akan dilakukan secara berurut linear mulai dari tahapan analisis kebutuhan sampai pemeliharaan. Berikut ini penjelasan setiap tahapan metode waterfall:

1. Tahap analisis kebutuhan

Metode Waterfall dimulai dengan tahapan mengumpulkan dan menganalisis kebutuhan sistem. Kebutuhan sistem tersebut harus diverifikasi dengan pengguna sebelum ditetapkan sebagai sebuah kebutuhan sistem. Semua kebutuhan sistem harus diidentifikasi dan dijelaskan secara detail dalam sebuah dokumentasi yang baik.

Tahapan ini merupakan tahap pertama dalam sebuah pengembangan sistem dan merupakan tahapan yang paling penting dan krusial. Hal ini dikarenakan, tahapan ini akan menentukan sistem seperti apa yang akan dihasilkan dalam sebuah proyek pengembangan perangkat lunak.

1. Tahap desain

Tahapan berikutnya setelah tahap analisis kebutuhan sistem adalah tahapan desain. Pada tahapan ini, kebutuhan sistem diterjemahkan menjadi sebuah rencana pengembangan sistem dalam bentuk model. Model tersebut dapat dianggap sebagai sebuah blueprint dari perangkat lunak yang dikembangkan.

Oleh sebab itu, model yang dibangun harus dapat dimengerti oleh semua tim pengembang dan menyertakan detail-detail yang diperlukan. Dengan kata lain, dengan membaca model yang ada, anggota tim harus dapat membangun sebuah perangkat lunak sesuai dengan yang diinginkan.

1. Tahap implementasi

Tahap ketiga dari metode Waterfall adalah tahap implementasi perangkat lunak. Pada tahapan ini model yang dibangun diterjemahkan menjadi perangkat lunak dengan cara menuliskan kode program. Sebuah perangkat lunak umumnya dibagi dalam unit-unit kecil yang disebut dengan fungsi dan prosedur, yang akan dikelompokkan dalam sebuah class dan modul-modul.

1. Tahap pengujian

Setelah proses implementasi, langkah selanjutnya adalah tahap pengujian. Pada tahap ini, dilakukan pengecekan bahwa perangkat lunak yang sudah dibangun telah bekerja sesuai dengan yang diharapkan. Tujuannya untuk melihat bahwa kinerja sistem sudah memenuhi tingkat yang dapat diterima oleh pelanggan, serta telah dapat mendeteksi bug dan error yang muncul dan memperbaikinya.

1. Tahap pemeliharaan

Tahap terakhir dari metode Waterfall adalah tahap pemeliharaan (maintenance). Tahap ini dilakukan untuk memperbaiki jika ternyata masih ada bug yang bersifat tidak kritis. Namun pada umumnya, tahap maintenance adalah tahap pengembangan. Jika ternyata ada beberapa kebutuhan organisasi yang belum diakomodasi dalam sebuah sistem, atau karena tuntutan perkembangan bisnis, pengembangan akan dilakukan pada tahap ini.

**Kelemahan metode waterfall**

Salah satu kelemahan dari metode Waterfall adalah sifatnya yang kaku sehingga sulit untuk merespons perubahan. Karena struktur pekerjaan yang bersifat linear, perubahan kebutuhan sistem sulit dilakukan jika sudah memasuki tahapan berikutnya. Hal ini akan menjadi kendala bagi pengguna yang sering berubah pikiran atau organisasi yang berada dalam kondisi bisnis yang tidak menentu. Untuk kasus seperti ini, metode pendekatan yang lain mungkin akan lebih sesuai.

1. **Metode Agile**

Pada metode Waterfall, Anda sudah melihat bahwa tahap-tahap yang ada dalam sebuah proses pengembangan perangkat lunak terdiri atas tahap analisis kebutuhan, desain, implementasi, pengujian, dan pemeliharaan. Semua tahapan ini dilakukan sekali. Ketika semua tahapan sudah dijalankan, proses pengembangan perangkat lunak sudah selesai.

Metode Agile dikerjakan dengan cara sprint**.** Karena dikerjakan dengan cara Sprint, metode Agile lebih mengakomodasi perubahan. Ketika sebuah fungsi diidentifikasi dan dikembangkan sampai pada tahap implementasi, dan didapati bahwa fungsi tersebut tidak sesuai atau ingin diubah, fungsi tersebut dapat diubah pada Sprint berikutnya. Hal inilah yang menyebabkan metode Agile lebih disukai oleh pengguna pada organisasi yang berada pada kondisi yang tidak menentu atau pengguna yang belum memiliki lingkungan bisnis yang stabil dan kaku.

Metode Agile berpandangan bahwa sistem yang dikembangkan harus dapat digunakan untuk keunggulan kompetisi dari pengguna. Hal ini dikarenakan di tengah kondisi persaingan bisnis yang semakin ketat, dengan strategi bisnis organisasi yang dapat berubah-ubah. Perangkat lunak yang dikembangkan juga harus dapat menyesuaikan dengan strategi bisnis organisasi. Dengan metode ini, sistem yang dikembangkan disesuaikan dengan bisnis organisasi dan lingkungan bisnisnya sehingga akan memberikan kepuasan kepada pelanggan.

3. Perbandingan Waterfall dan Agile

Masing-masing metode memiliki kelebihan dan kekurangan dan cocok digunakan untuk kondisi yang sesuai dengan sifat masing-masing metode. Dari sisi pengembangan, perubahan kebutuhan sistem akan menyebabkan perubahan pada tahap yang lain. Hal ini akan menyebabkan bertambahnya biaya untuk pengembangan sebuah perangkat lunak.

Untuk itu, metode Waterfall lebih sesuai untuk digunakan. Sebaliknya, pada sebuah sistem yang mempunyai kondisi lingkungan yang kurang menentu, yaitu kebutuhan sistem tidak dapat didefinisikan secara jelas dan tetap. Penggunaan metode Agile akan lebih sesuai karena dapat mengakomodasi lingkungan dinamis yang ada.

**Metode-metode mendapatkan kebutuhan system**

1. Interview

Interview merupakan salah satu cara yang paling sering digunakan untuk mendapatkan kebutuhan sistem. Interview digunakan untuk menggali berbagai aspek yang berhubungan dengan pengembangan sebuah perangkat lunak. Aspek tersebut mencakup identifikasi masalah bisnis yang sering terjadi, keluhan-keluhan manajemen, pengguna, dan pelanggan yang sering muncul, ide-ide pengembangan, dan harapan-harapan pengguna.

Oleh karena itu, perlu dibangun sebuah suasana yang dapat menghilangkan batas-batas antara pengembang dan pengguna. Pengembang perlu menyiapkan daftar pertanyaan, tetapi harus fleksibel menggunakan pertanyaan-pertanyaan tersebut pada saat proses interview berjalan. Proses interview harus didesain untuk tidak menjadi terlalu serius sehingga orang yang di-interview merasa bebas untuk menyampaikan pendapatnya. Namun, interview perlu dijaga agar tetap terarah sehingga tujuannya dapat tercapai.

1. Melakukan Diskusi

Cara lain untuk mendapatkan kebutuhan sistem yang dibangun adalah dengan melakukan diskusi. Diskusi dilakukan antara tim pengembang dengan tim pengguna. Tim pengguna dapat terdiri atas manajer puncak, manajer tingkat menengah dan bawah, serta beberapa perwakilan pengguna. Semakin banyak orang yang terlibat dalam diskusi, semakin banyak masukan yang diperoleh. Hal ini akan berguna untuk membuat sebuah basis data yang andal dan bermanfaat tinggi.

1. Pengamatan di Lapangan

Pengamatan langsung di lapangan merupakan cara yang paling efektif untuk mempelajari bisnis organisasi dan mengidentifikasi kebutuhan sistem untuk menunjang bisnis organisasi. Pengamatan langsung akan memberikan pemahaman bagi pengembang mengenai operasi bisnis yang dilakukan organisasi. Pengamat akan merasakan langsung hal yang dialami pelanggan dan mengetahui cara pandang pegawai organisasi yang melakukan kegiatan bisnis. Pengamatan di lapangan akan membuat pengembang dapat menemukan masalah bisnis yang sering terjadi, cara memperbaiki sebuah proses agar menjadi lebih baik, bidang pelayanan yang perlu ditingkatkan, dan sebagainya.

# BAB 3

## 3.1 Penutup

Sekian hanya ini saja isi materi makalah Permodelan saya. Saya menyadari sebagai manusia, saya tidak terlepas dari kesalahan yang bisa saja saya lakukan dalam menulis dan membuat makalah ini, oleh karena itu demi perbaikan isinya, saya sangat mengharapkan adanya saran serta kritik yang bersifat menyempurnakan makalah ini.

semoga makalah Permodelan ini dapat membantu menambah pengetahuan dan wawasan kita, serta dapat memberikan skill dan kemampuan bagi sang pembaca untuk menciptakan diagram UML suatu system berbasis objek. Akhir kata, saya mengucapkan Terimakasih.