BAB I
PENDAHULUAN

1.1. Latar Belakang

Gabungan kelompok tani (Gapoktan) adalah kumpulan beberapa kelompok tani yang bergabung dan bekerja sama untuk meningkatkan skala ekonomi dan efisiensi usaha [PER–07 : 419]. Gapoktan juga merupakan ujung tombak pemasaran produk pertaian yang dihasilkan petani. Peran gapoktan diharapkan mampu meningkatkan kesejahteraan petani. Rendahnya kesejahteraan petani ini disebabkan diantaranya oleh rendahnya nilai tambah produk yang dinikmati oleh petani. Petani menjual produk pertanian hasil panen begitu saja kepada tengkulak. Banyak petani yang menjual hasil pertanian, misalnya padi, ketika masih berada di sawah [SOH - 12]. Penulis telah memberikan solusi pada penelitian sebelumnya yang berjudul "Perancangan dan Implementasi Aplikasi Berbasis *Web* dan *Web Services* sebagai Sistem Informasi Gapoktan (Gabungan Kelompok Tani)" [PKL - 12]. Pada penelitian tersebut dihasilkan sistem informasi berbasis *website* yang mampu memasarkan produk gapoktan melalui *internet*. *Internet* merupakan salah satu strategi jitu dalam dunia pemasaran. Jangkauan wilayah dan biaya operasional yang murah menjadi alasan banyak pedagang memasarkan produknya dengan *internet* termasuk gapoktan. Tantangan yang dihadapi saat ini adalah kemudahan, mobilitas dan kecepatan akses bagi petani, sebab sebagian besar petani masih hidup di pedesaan yang belum terjangkau *internet*.

Sistem pemasaran pada penelitian sebelumnya berupa sistem informasi gapoktan berbasis *web* yang bisa digunakan secara optimal jika menggunakan *Personal Computer* (PC) atau laptop. *Web* dibuat untuk memiliki tampilan atau *user interface*. Kondisi tersebut membuat akses informasi menjadi berat, karena *user* harus mengunduh *interface web*. Hal itu dapat diatasi dengan aplikasi mobile berbasis android yang memanfaatkan layanan *web services* yang ada di sistem informasi gapoktan berbasis *web*. *User* hanya perlu mengunduh data dari *web services* saja dengan tampilan yang ditanampkan pada aplikasi android. Android

dipilih sebagai solusi karena pengembangan aplikasi pada *platform* ini bersifat *open source*.

Aplikasi *mobile* untuk sistem informasi gapoktan berbasis android (SI Gapoktan *Mobile*) akan mengimplementasi fitur dari sistem informasi gapoktan berbasis *web*. SI Gapoktan *Mobile* mengakses layanan *web* service dalam format XML dan memetakan informasi sesuai dengan fitur yang ada. Hasil yang diharapkan dari penelitian ini adalah terciptanya aplikasi *mobile* yang memudahkan petani dan calon pembeli mengakses sistem informasi gapoktan berbasis *web*.

1.2. Rumusan Masalah

Berdasarkan permasalahan yang diangkat pada bagian latar belakang, maka rumusan masalah dikhususkan pada :

1. Bagaimana gambaran umum perangkat lunak SI Gapoktan *Mobile*?
2. Bagaimana perancangan perangkat lunak SI Gapoktan *Mobile*?
3. Bagaimana implementasi pengembangan perangkat lunak SI Gapoktan *Mobile* menggunakan bahasa pemrograman *java* android?
4. Bagaimana skenario, proses dan hasil pengujian perangkat lunak SI Gapoktan *Mobile*?

1.3. Batasan Masalah

Agar diperoleh hasil pembahasan yang sesuai dengan apa diharapkan, maka perlu diberikan pembatasan masalah pada pengembangan perangkat lunak ini, yaitu :

1. Pembahasan difokuskan pada rekayasa perangkat lunak dari pembuatan perangkat lunak SI Gapoktan *Mobile* .
2. Pengembangan perangkat lunak dilakukan dalam lingkungan sistem operasi *Microsoft Windows 7 Home Premium* 64-bit.
3. IDE (*Integrated Development Environment*) yang dipakai dalam pengembangan perangkat lunak SI Gapoktan *Mobile* adalah *Eclipse* ADT *Build*: v22.0.5-757759.
4. Pengembangan perangkat lunak dilakukan dengan bahasa pemrograman *java* android.

5. *Platform* JDK (*Java Development Kit*) yang dipakai adalah JDK 8.
6. *Emulator* yang dipakai adalah *emulator* dengan target Google APIs (Google Inc.) - API level 18 dengan CPU ARM (armeabi-v7a).
7. Target SDK yang digunakan adalah Android 4.3 *jelly bean.*
8. Pengiriman notifikasi menggunakan GCDM.
9. Pengembangan perangkat lunak SI Gapoktan *Mobile* menggunakan metode *Component-Based Software Engineering* (CBSE).
10. Tahapan pengembangan perangkat lunak yang terdiri atas tahap perancangan, tahap implementasi, dan tahap pengujian dilakukan pada *notebook* / *laptop* Lenovo G480.
11. Pengujian yang dilakukan adalah pengujian validasi menggunakan metode *black-box testing* dan pengujuan *User* Acceptance Test (UAT) kepada penyuluh dan gapoktan.

1.4. Tujuan

Tujuan dari pengembangan perangkat lunak ini adalah untuk mendapatkan gambaran dari perangkat lunak yang sesuai dengan kondisi di lapangan. Pengembangan perangkat lunak ini juga bertujuan untuk mengetahui perancangan dan implementasi perangkat lunak sesuai dengan metode dan bahasa pemrograman yang digunakan. Pada tahap pengujian diharapkan perangkat lunak ini dapat memudahkan para petani dan calon pembeli produk-produk gapoktan mengakses Sistem Informasi Gapoktan Berbasis *Web* yang diketahui melalui pengujian *User Acceptance Test* (UAT).

1.5. Manfaat

Manfaat yang nantinya dapat diambil dari pengembangan perangkat lunak ini adalah sebagai berikut :

a. Bagi penulis
   1. Menerapkan ilmu yang telah didapatkan di Teknik Informatika Laboraturium Rekayasa Perangkat Lunak Universitas Brawijaya.
   2. Mendapatkan pemahaman lebih lanjut dalam pengembangan perangkat lunak SI Gapoktan *Mobile* .

b. Bagi pengguna

1. Menyediakan aplikasi *mobile* yang dapat digunakan untuk mengakses semua fitur dari sistem informasi gapoktan.

1.6. Sistematika Pembahasan

Sistematika pembahasan ditunjukkan untuk memberikan gambaran dan uraian dari penulisan skripsi ini secara garis besar yang meliputi beberapa bab, sebagai berikut :

Bab I : Pendahuluan

Menguraikan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika pembahasan.

Bab II : Tinjauan Pustaka

Menguraikan tentang dasar teori dan referensi yang mendasari pengembangan perangkat lunak SI Gapoktan *Mobile* .

Bab III : Metode Penelitian

Menguraikan tentang metode dan langkah kerja yang dilakukan dalam proses perancangan dan implementasipada pelaksanaan skripsi dan perancangan sistem yang menjadi objek studi kasus skripsi.

Bab IV :Analisis dan Perancangan

Membahas analisis kebutuhan dan perancangan perangkat lunak SI Gapoktan *Mobile* sesuai dengan dasar teori dan literatur yang ada.

Bab V : Implementasi

Membahas implementasi dari perangkat lunak SI Gapoktan *Mobile* sesuai dengan perancangan perangkat lunak yang telah dibuat.

Bab VI : Pengujian dan Analisis

Memuat hasil pengujian dan analisis terhadap perangkat lunak yang telah direalisasikan.

Bab VII : Penutup

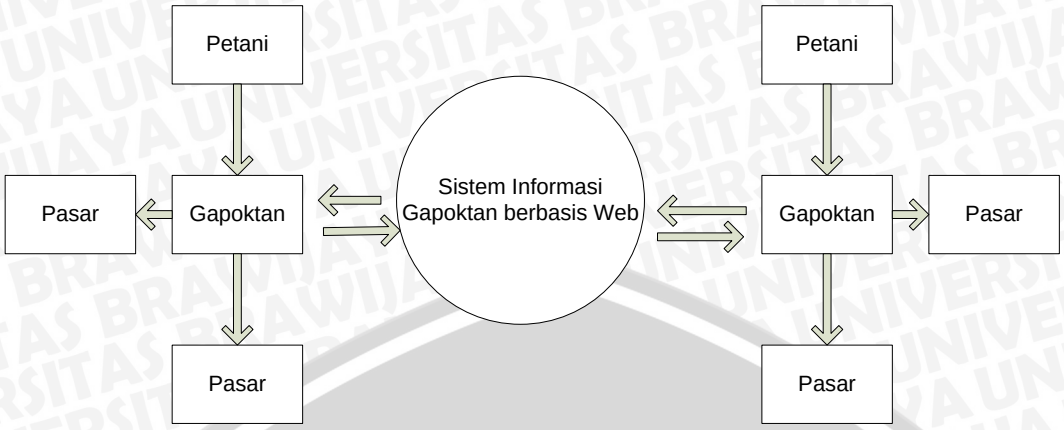
Memuat kesimpulan yang diperoleh dari pembuatan dan pengujian perangkat lunak yang dikembangkan dalam skripsi ini serta saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

Bab tinjauan pustaka terdiri dari kajian pustaka dan dasar teori. Kajian pustaka membahas penelitian terdahulu yang menjadi acuan pengembangan skripsi. Penelitian dengan judul "Perancangan dan Implementasi Aplikasi Berbasis *Web* dan *Web Services* sebagai Sistem Informasi Gapoktan (Gabungan Kelompok Tani)" membahas sistem pemasaran produk pertanian dari petani ke konsumen. Penelitian tersebut juga memberi solusi berupa sistem informasi gapoktan berbasis *web* sebagai jembatan antara petani dan konsumen. Sistem informasi tersebut selanjutnya dikembangkan pada penelitian saat ini menjadi SI Gapoktan *Mobile.* Dasar teori yang menjadi landasan SI Gapoktan *Mobile* meliputi konsep dasar perangkat lunak, pengembangan dan rekayasa perangkat lunak, konsep *Component-Based Software Engineering*, konsep dasar *Unified Modelling Language* yang dipakai pada rekayasa perangkat lunak, konsep dasar teknik dan strategi pengujian perangkat lunak, pemrograman *java* android, *web services* , MySQL, dan XML.

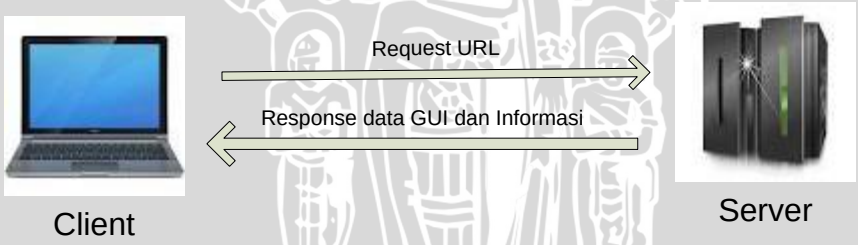
2.1. Kajian Pustaka

Penelitian terdahulu yang berjudul "Perancangan dan Implementasi Aplikasi Berbasis *Web* dan *Web Services* sebagai Sistem Informasi Gapoktan (Gabungan Kelompok Tani)" [PKL – 12] digunakan sebagai acuan pengembangan penelitian saat ini. Pada penelitian tersebut dijelaskan sistem distribusi pemasaran produk pertanian dari petani ke konsumen yang sangat merugikan petani. Penelitian tersebut juga menjelaskan keterlibatan tengkulak dan pedagang besar yang sangat merugikan petani. Harga jual yang rendah dan tidak stabil sering menambah kerugian petani. Pada penelitian tersebut penulis memberikan solusi berupa sistem informasi gapoktan berbasis *web* sebagai jembatan antara petani dan konsumen. Sistem informasi gapoktan diharapkan memutus rantai pemasaran produk pertanian yang terlalu panjang. Berikut adalah alur pemasaran setelah ada sistem informasi gapoktan berbasis *web* :



Gambar 2.1 Alur pemasaran produk pertanian dengan sistem informasi gapoktan berbasis *web*
Sumber : [PKL-12]

Sistem informasi gapoktan berbasis *web* berfungsi sebagai penghubung antar gapoktan. Sistem ini mendukung tujuan gapoktan sebagai unit usaha pemasaran pemasaran produk pertanian [PER – 07 : 432]. Pada sistem ini gapoktan harus mengusai pasar di daerah dan berhubungan dengan gapoktan lain untuk melakukan kegiatan jual beli produk pertanian yang dibutuhkan oleh pasar. Gambar berikut menjelaskan arsitektur sistem informasi gapoktan berbasis *web* :



Gambar 2.2 Diagram blok sistem informasi gapoktan berbasis *web*
Sumber : [PKL-12]

Pada sistem informasi gapoktan berbasis *web* client melakukan *request* dengan mengirimkan URL ke *server* dan *server* memberikan *response* berupa data informasi dan *Graphic User Interface* (GUI) ke *client*. Hal ini membuat akses sistem informasi menjadi berat. Pada penelitian kali ini penulis akan membuat aplikasi mobile yang memanfaatkan *web services* dari sistem informasi gapoktan berbasis *web*. Penelitian ini bertujuan untuk mempercepat akses sistem informasi gapoktan berbasis *web*. Diagram blok SI Gapoktan *Mobile* akan digambarkan pada gambar 2.3. Pada SI Gapoktan *Mobile* *user* merequest *service* ke sistem

informasi gapoktan dan mendapatkan data berupa objek XML berisi data yang diinginkan. GUI sudah ditanamkan di client.

Gambar 2.3 Diagram blok SI Gapoktan *Mobile*
Sumber : Perancangan

SI Gapoktan *Mobile* juga dilengkapi dengan fitur *push notification* dengan memanfaatkan layanan dari *Google Cloud to Device Messaging* (GCDM). Fitur ini akan memberikan notifikasi secara *real time* kepada *user*. Fitur ini bekerja setelah *user* mendaftarkan akun google pada perangkat bergerak androidnya. Berikut adalah diagram blok dari GCDM.

Gambar 2.4 Diagram blok registrasi akun google android
Sumber : Perancangan

Layanan GCM menangani semua aspek antrian pesan dan pengiriman ke target aplikasi Android yang berjalan pada perangkat target [DEV].

2.2. Web Service

Web service adalah suatu sistem perangkat lunak yang dirancang untuk mendukung interoperabilitas dan interaksi antar sistem pada suatu jaringan. *Web service* digunakan sebagai suatu fasilitas yang disediakan oleh suatu *web* site untuk menyediakan layanan (dalam bentuk informasi) kepada sistem lain, sehingga sistem lain dapat berinteraksi dengan sistem tersebut melalui layanan-layanan (*service*) yang disediakan oleh suatu sistem yang menyediakan *web*

service. *Web service* menyimpan data informasi dalam format XML, sehingga data ini dapat diakses oleh sistem lain walaupun berbeda *platform* , sistem operasi, maupun bahasa compiler [YAU-12].

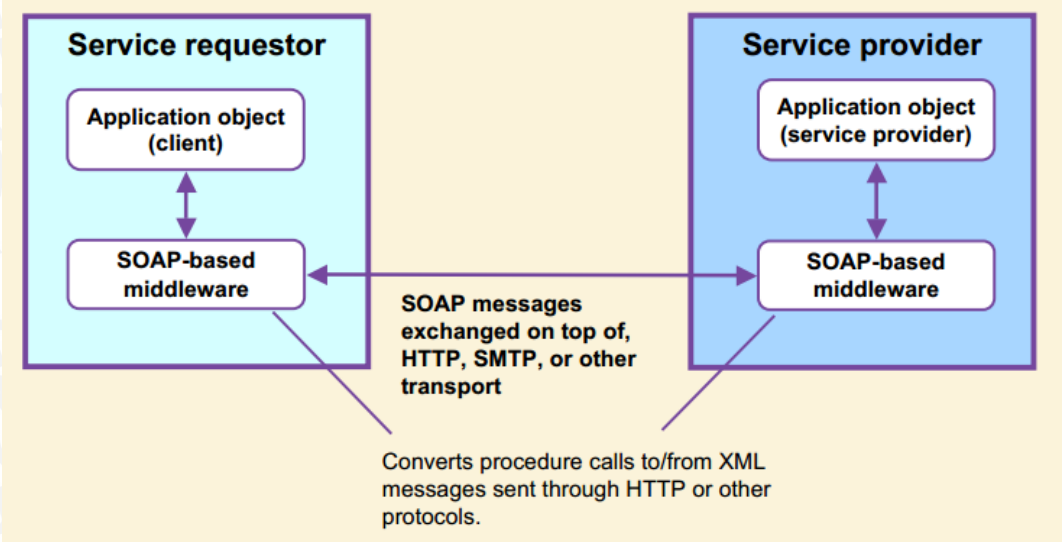
Web service bertujuan untuk meningkatkan kolaborasi antar pemrogram dan perusahaan, yang memungkinkan sebuah fungsi di dalam *web service* dapat dipinjam oleh perangkat lunak lain tanpa perlu mengetahui detil pemrograman yang terdapat di dalamnya.

Beberapa alasan mengapa digunakannya *web service* adalah sebagai berikut [YAU-12] :

1. *Web service* dapat digunakan untuk mentransformasikan satu atau beberapa bisnis logic atau class dan objek yang terpisah dalam satu ruang lingkup yang menjadi satu, sehingga tingkat keamanan dapat ditangani dengan baik.
2. *Web service* memiliki kemudahan dalam proses deployment-nya, karena tidak memerlukan registrasi khusus ke dalam suatu sistem operasi. *Web service* cukup di-upload ke *web server* dan siap diakses oleh pihak-pihak yang telah diberikan otorisasi.
3. *Web service* berjalan di port 80 yang merupakan protokol standar HTTP, dengan demikian *web service* tidak memerlukan konfigurasi khusus di sisi firewall.

2.2.1. *Simple Object Access Protocol* (SOAP) *Web* Service

SOAP adalah protokol pesan standar yang digunakan oleh layanan Web. SOAP digunakan untuk komunikasi antar aplikasi. SOAP menggunakan format data XML sebagai skema pengkodean untuk data *request* dan parameter respon menggunakan HTTP sebagai sarana pengiriman data. Berikut adalah arsitektur SOAP web services :



Gambar 2.5 Arsitektur SOAP *web services*
sumber : [PAP – 08 :5]

SOAP meliputi empat bidang utama sebagai berikut[PAP – 08 : 6]:

- Sebuah format pesan untuk komunikasi satu arah menggambarkan bagaimana pesan bisa dimasukkan ke dalam dokumen XML.
- Penjelasan tentang bagaimana SOAP harus dikirim menggunakan HTTP (untuk interaksi berbasis Web) atau SMTP (untuk interaksi e-mail-based).
- Satu set aturan yang harus diikuti ketika memproses SOAP dan klasifikasi sederhana dari entitas terlibat dalam pengolahan SOAP.
- Satu set konvensi tentang bagaimana mengubah panggilan RPC menjadi pesan SOAP dan punggung.

2.2.2. Restfull *Web Services*

RESTful *web* service atau juga dikenal dengan nama RESTful *Web* API merupakan sebuah *web* service yang di implemantasikan dengan menggunakan http dengan menggunakan prinsip-prinsip REST. Service yang digunakan menggunakan method milik HTTP [*WEB* - 11].

Aplikasi client atau *server* dengan dukungan HTTP dapat dengan mudah memanggil *service* tersebut dengan command HTTP GET. Berdasar pada bagaimana cara penyedia *service* menulis script, hasil respons HTTP kan menjadi lebih simpel seperti beberapa header standar dan string teks yang mengandung

harga terkini untuk symbol yang diberikan. Atau, dapat berupa dokumen XML [ADR-12]. Penggunaan metode REST dalam *web service* menggunakan beberapa komponen sebagai berikut :

1. XML : format yang merepresentasikan sumber, berfungsi sebagai bahasa pesan antar *platform* ,
2. HTTP : metode GET, HEAD, POST, PUT dan DELETE mengindikasikan action yang akan diambil,
3. URI : URI adalah resource identifier lokasi dari *web service.*

2.2.3. *Web Services Description Language* (WSDL)

WSDL adalah sebuah XML-based language untuk mendeskripsikan XML. WSDL menyediakan *service* atau layanan yang mendeskripsikan *service request* dengan menggunakan protokol-protokol yang berbeda dan juga encoding. WSDL memfasilitasi komunikasi antar aplikasi. WSDL akan mendeskripsikan apa yang akan dilakukan oleh *web service*, bagaimana menemukannya dan bagaimana untuk mengoperasikannya.

Spesifikasi WSDL mendefinisikan tujuh tipe element:

1. Types : element untuk mendefinisikan tipe data. Mereka akan mendefinisikan tipe data (seperti string atau integer) dari element didalam sebuah message.
2. Message : abstract, pendefinisian tipe data yang akan dikomunikasikan.
3. Operation : sebuah deskripsi abstract dari sebuah action yang didukung oleh service.
4. Port Type : sebuah koleksi abstract dari operations yang didukung oleh lebih dari satu endpoints.
5. Binding : mendefinisikan penyatuan dari tipe port (koleksi dari operasioperasi) menjadi sebuah protokol transport dan data format (ex. SOAP 1.1 pada HTTP). Ini adalah sebuah protokol konkret dan sebuah spesifikasi data format didalam tipe port tertentu.
6. Port : mendefinisikan sebuah komunikasi endpoint sebagai kombinasi dari binding dan alamat network. Bagi protokol HTTP, sebuah bentuk

dari URL sedangkan bagi protokol SMTP, ini adalah sebuah form dari email address.

7. Service : satu set port yang terkorelasi atau suatu endpoints.

WSDL mendefinisikan service sebagai sebuah koleksi dari endpoints network. Sebuah definisi abstrak dari endpoints dan messages adalah ia bersifat terpisah dari pembangunan network atau penyatuan data format. Pembagian ini menyebabkan penggunaan kembali *abstract description* dari data yang akan dipertukarkan (message exchange) dan abstract collection dari operasi (*ports*) Protokol konkret dan spesfikasi data format bagi tipe port tertentu menentukan binding yang dapat digunakan kembali(reusable). Sebuah port adalah sebuah network address yang dikombinasikan reusable binding; sebuah service adalah koleksi dari port-port.

2.3. Hypertext Transfer Protocol (HTTP) Methods

HTTP digunakan dalam arsitektur REST. Representasi adalah mapping resource yang ditransfer antara client dan *server*. Pada *web service* representasi tersebut diajukan dengan komunikasi antara client dan *server* melalui protokol komunikasi HTTP [ADR-13].

Membangun sebuah RESTful *web service* akan sejalan dengan cara membangun sebuah aplikasi *web* masa kini. Akan tetapi hal paling dasar yang berbeda untuk membangun aplikasi *web* yang modern dan tradisional adalah bagaimana memodelkan aksi ke abstraksi data. Modern development menitik beratkan pada pertukaran resource, sedangkan tradisional development menitik beratkan pada aksi remote untuk mengambil data [RER-12].

2.4. Rekayasa Perangkat Lunak (RPL)

Ian Sommerville dalam bukunya yang berjudul *Software Engineering* menyebutkan bahwa rekayasa perangkat lunak adalah disiplin ilmu yang membahas semua aspek produksi perangkat lunak, mulai dari tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan.

Pada definisi ini, ada dua istilah kunci yaitu disiplin rekayasa dan semua aspek produksi perangkat lunak.

1. Disiplin rekayasa

   Perekayasa membuat suatu alat bekerja. Mereka menerapkan teori, metode, dan alat bantu yang sesuai, selain itu mereka menggunakannya dengan selektif dan selalu mencoba mencari solusi terhadap permasalahan, walaupun tidak ada teori atau metode yang mendukung. Perekayasa juga menyadari bahwa mereka harus bekerja dalam batasan organisasi dan keuangan, sehingga mereka berusaha mencari solusi dalam batasan – batasan ini.

2. Semua aspek produksi perangkat lunak

   Rekayasa perangkat lunak tidak hanya berhubungan dengan proses teknis dari pengembangan perangkat lunak tetapi juga dengan kegiatan seperti manajemen proyek perangkat lunak dan pengembangan alat bantu, metode dan teori untuk mendukung produksi perangkat lunak.

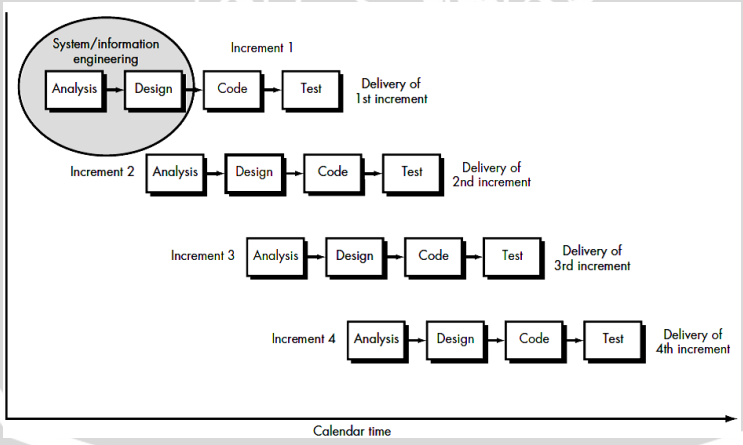
Secara umum, perekayasa perangkat lunak memakai pendekatan yang sistematis dan terorganisir terhadap pekerjaan mereka karena cara ini seringkali paling efektif untuk menghasilkan perangkat lunak berkualitas tinggi. Namun demikian, rekayasa ini sebenarnya mencakup masalah pemilihan metode yang paling sesuai untuk satu set keadaan dan pendekatan yang lebih kreatif, informal terhadap pengembangan yang mungkin efektif pada beberapa keadaan [SOM-03:7].

Perekayasa perangkat lunak memakai pendekatan yang sistematis dan terorganisir terhadap pekerjaan mereka karena cara ini seringkali paling efektif untuk menghasilkan perangkat lunak berkualitas tinggi. Rekayasa ini mencakup masalah pemilihan metode yang paling sesuai untuk satu set keadaan dan pendekatan yang lebih kreatif, informal terhadap pengembangan yang mungkin efektif pada beberapa keadaan [SOM-03:7].

Seorang *software engineer* atau sekumpulan *software engineer* harus menggabungkan strategi pengembangan perangkat lunak yang meliputi proses, metode, dan alat bantu yang digunakan dalam proses pengembangan. Strategi ini disebut sebagai model proses (*process model*) atau paradigma rekayasa perangkat lunak (*software engineering paradigm*). Model proses untuk rekayasa perangkat lunak dipilih sesuai dengan sifat dari proyek dan aplikasi yang akan dibuat.

Terdapat beberapa model proses untuk rekayasa perangkat lunak antara lain *linear sequential model* (*waterfall*), *prototyping*, *Rapid Application Development* (RAD), *incremental model* dan *spiral model* [PRE-10:20-42]. Model proses yang digunakan dalam skripsi ini adalah *incremental model*.

Incremental model merupakan gabungan dari elemen *linear sequensial model* dan filosofi dari *prototyping* sehingga memudahkan seorang pengembang (*developer*) dalam mengidentifikasi kebutuhan pengguna [PRE-10:35]. *Incremental model* merupakan metodologi pengembangan perangkat lunak yang efisien dan efektif untuk *project* dengan skala kecil hingga skala menengah. *Incremental model* menawarkan strategi pengembangan perangkat lunak yang memungkinkan pengguna (*user*) dapat menggunakan versi awal (*increment* pertama) sebagai *prototype* untuk memperoleh informasi tentang persyaratan kebutuhan (*requirement*) mereka untuk pengembangan sistem selanjutnya [SOM-11:47]. Kelebihan dari model proses ini antara lain adalah proses *development* yang lebih cepat, lebih mudah dalam mengetahui kebutuhan pengguna, dan *resource* yang dibutuhkan untuk melakukan perubahan terhadap perangkat lunak yang dikembangkan lebih sedikit [SOM-11:33]. Proses pengembangan menggunakan model proses *incremental* ini dapat dilihat pada Gambar 2.4.



Gambar 2.6 *Incremental model*
Sumber : [PRE-10:35]

Versi awal dari perangkat lunak (*increment* 1) dievaluasi oleh pengguna sehingga pihak pengembang mengetahui dengan jelas apa yang dibutuhkan oleh pengguna. Pihak pengembang kemudian mengembangkan perangkat lunak untuk

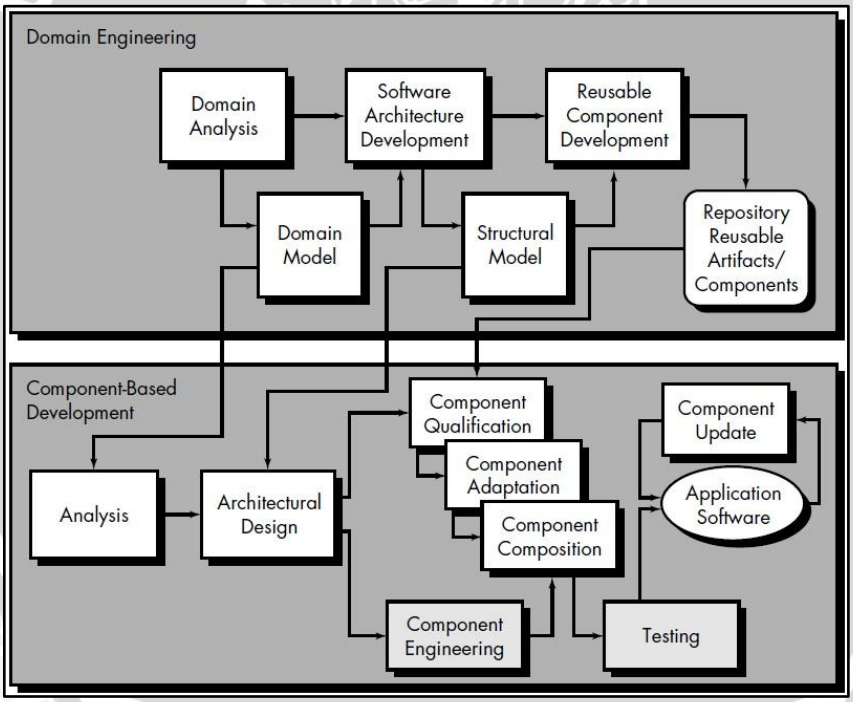
increment selanjutnya sesuai dengan saran dan permintaan pengguna. Proses ini berulang sampai perangkat lunak selesai atau mencapai tahap yang diinginkan oleh pengguna [PRE-10:35].

Pada skripsi ini digunakan metode analisis kebutuhan, perancangan, implementasi, dan pengujian berorientasi objek menggunakan bahasa pemodelan UML (*Unified Modelling Language*).

2.5. *Component-Based Software Engineering* (CBSE)

CBSE adalah metode pengembangan perangkat lunak yang menekankan pada perancangan dan pembangunan perangkat lunak dengan menggunakan komponen perangkat lunak yang sudah ada dan bersifat dapat digunakan kembali. [PRE-01:721]. Alur *software process* dari CBSE digambarkan pada Gambar 2.4.

CBSE memiliki *software process* yang terdiri dari dua bagian utama yang berjalan paralel, yaitu *domain engineering* dan *component-based development*.



Gambar 2.7 *Software process* dari CBSE
Sumber : [PRE-01:725]

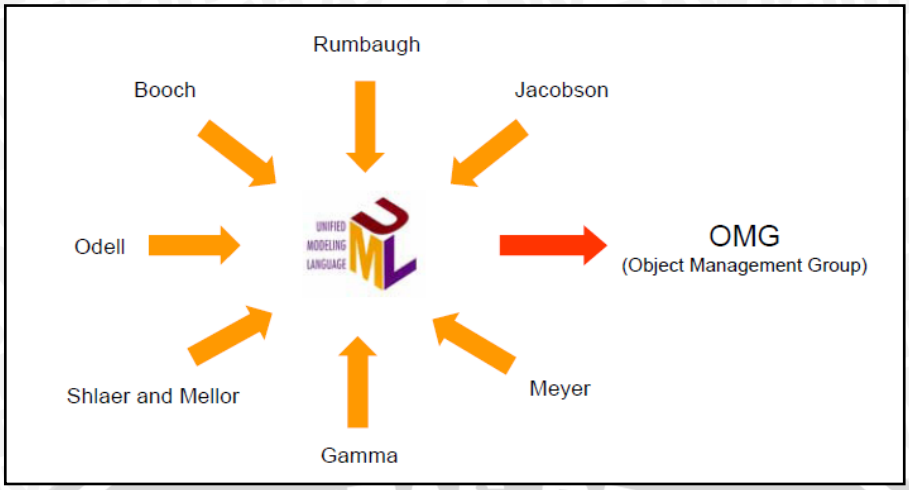
2.6. *Unified Modelling Language* (UML)

UML adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi perangkat lunak, dimana aplikasi tersebut dapat berjalan pada perangkat keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan perangkat lunak dalam bahasa berorientasi objek seperti C++, *Java*, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk *modeling* aplikasi prosedural dalam VB atau C [DHA-03:2].

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram perangkat lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*).

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui 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 dan sebagainya. 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 grup / perusahaan lain yang menggunakan metodologi yang berlainan.



Gambar 2.8 Penyatuan metode UML
Sumber : [DHA-03:3]

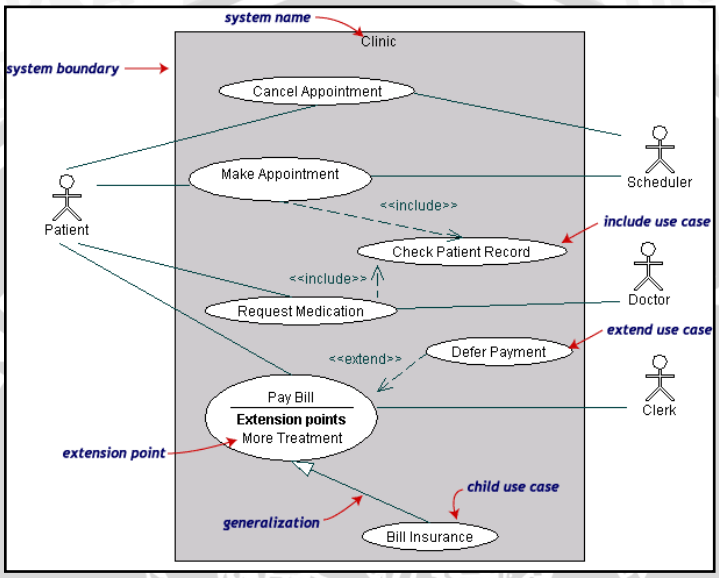
Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 di-*releasedraft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* 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 [DHA-06:3]. Dalam UML pemodelan digambarkan dengan penggunaan diagram – diagram yang masing – masing memiliki fungsi dan makna sendiri – sendiri. Diagram – diagram ini dapat digunakan sesuai dengan kebutuhan saat kita akan merancang perangkat lunak. Beberapa diantaranya adalah diagram *use case*, diagram *class* dan diagram *sequence*.

2.6.1 *Use Case Diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor

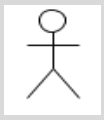
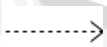
adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu [DHA-03:4].

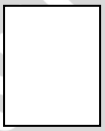
Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.



Gambar 2.9 Contoh *use case diagram*
Sumber : [DHA-03:5]

Tabel 2.1 Keterangan simbol - simbol *use case diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Actor*	Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
2		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempengaruhi elemen yang bergantung padanya sebagai elemen yang tidak mandiri (*independent*).
3		*Generalization*	Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di

			atasnya (objek induk / *ancestor*).
4		*Include*	Menspesifikasikan bahwa *use case* sumber secara *eksplisit*.
5		*Extend*	Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.
6		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
7		*System*	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
8		*Use Case*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu actor.
9		*Collaboration*	Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi).
10		*Note*	Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi.

Sumber : Kajian Pustaka dan Dasar Teori

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

secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

2.6.2 *Class Diagram*

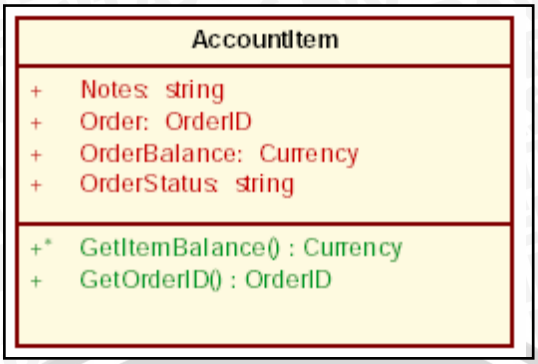
Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah *object* dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan *object* beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain [DHA-03:5].

Class memiliki tiga area pokok :

1. Nama (dan *stereotype*)
2. Atribut
3. Metoda

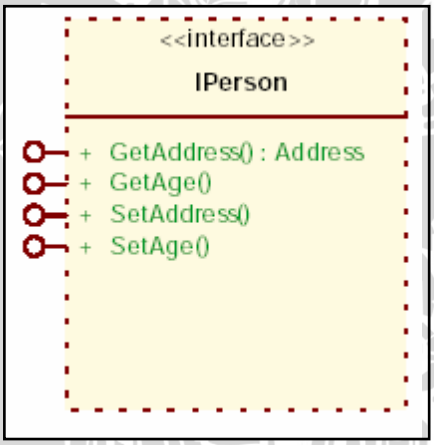
Atribut dan metoda dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak -anak yang mewarisinya
3. *Public*, dapat dipanggil oleh siapa saja



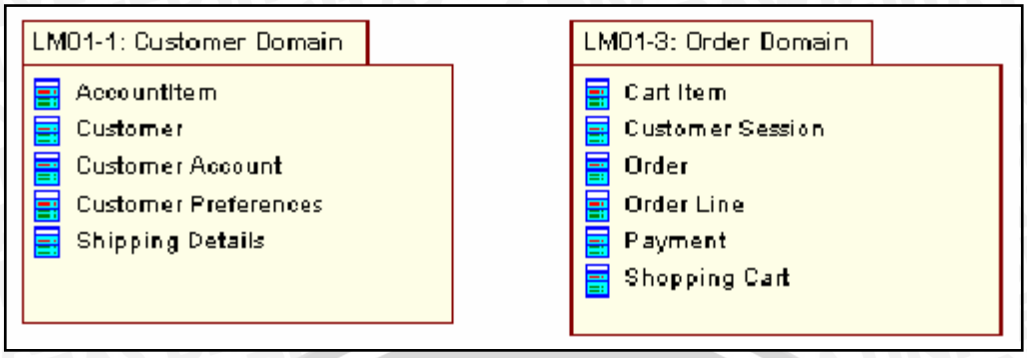
Gambar 2.10 Contoh sebuah *class*
Sumber : [DHA-03:5]

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*.



Gambar 2.11Contoh sebuah *interface*
Sumber : [DHA-03:6]

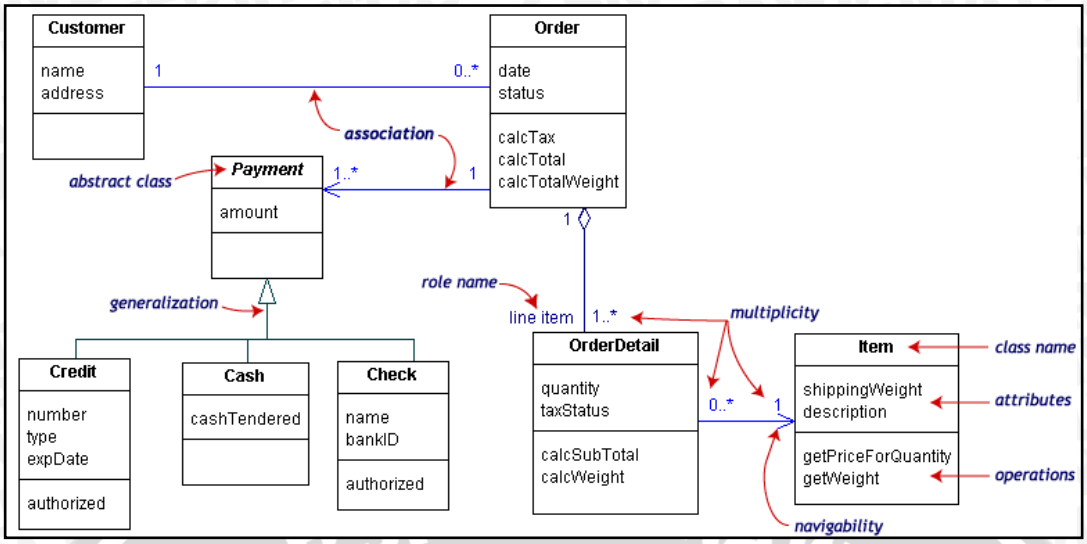
Sesuai dengan perkembangan *class model*, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.



Gambar 2.12 Contoh *package*
Sumber : [DHA-03:6]

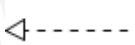
Dalam penggunaan metode *object-oriented* dapat dipastikan *class* – *class* memiliki hubungan – hubungan tertentu sesuai dengan fungsinya sendiri – sendiri. Hubungan antar *class* ada beberapa, yaitu :

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.



Gambar 2.13 Contoh *classdiagram*
Sumber : [DHA-03:6]

Tabel 2.2 Keterangan simbol - simbol *classdiagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Generalization*	Hubungan dimana objek anak *(descendent)* berbagi perilaku dan struktur data dari objek yang ada di atasnya (objek induk / *ancestor*).
2		*N-ary Association*	Upaya untuk menghindari asosiasi dengan lebih dari 2 objek.
3		*Class*	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
4		*Collaboration*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor.
5		*Realization*	Operasi yang benar-benar dilakukan oleh suatu objek.
6		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempengaruhi elemen yang bergantung padanya atau

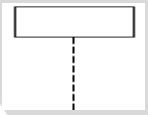
			elemen yang tidak mandiri.
7	———	*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
8	1..*, 0...*, 1...1	*Multiplicity*	Menunjukkan jumlah objek yang diperbolehkan menurut logika.

Sumber : Kajian Pustaka dan Dasar Teori

2.6.3 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan [DHA-03:8].

Tabel 2.3 Keterangan simbol - simbol *sequencediagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*LifeLine*	Objek *entity*, antarmuka yang saling berinteraksi.
2		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi

3		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi
4		*Activation*	Menunjukkan saat dimana *entity* diaktifkan di sepanjang *Life Line.*

Sumber : Kajian Pustaka dan Dasar Teori

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity*.

2.7 Bahasa Pemrograman *Java*

Java adalah bahasa pemrograman serbaguna. *Java* dapat digunakan untuk membuat suatu program sebagaimana program yang dibuat dengan bahasa lain seperti Pascal atau C++. Yang lebih menarik, *Java* juga mendukung sumber daya *internet* yang saat ini sangat populer, yaitu *world wide web* atau yang sering disebut sebagai *web* saja. *Java* juga mendukung aplikasi klien/*server*, baik dalam jaringan lokal (LAN) maupun jaringan berskala luas (WAN).

Java dikembangkan oleh Sun Microsystems pada Agustus 1991, dengan nama semula Oak. Konon Oak adalah pohon semacam jati yang terlihat dari jendela tempat pembuatnya, James Gosling, bekerja. Ada yang mengatakan bahwa Oak adalah singkatan dari “Object Application Kernel”, tetapi ada yang mengatakan hal itu muncul setelah nama Oak diberikan. Pada Januari 1995, karena nama Oak dianggap kurang komersial, maka diganti menjadi *Java*.

Dalam sejumlah literatur disebutkan bahwa *Java* merupakan hasil perpaduan sifat dari sejumlah bahasa pemrograman, yaitu C, C++, Object-C, SmallTalk dan Common LISP. Selain itu *Java* juga dilengkapi dengan unsur

keamanan. Yang tak kalah penting adalah bahwa *Java* menambahkan paradigma pemrograman yang sederhana. Misalnya dalam bahasa C atau C++ yang menggunakan pointer, namun *Java* justru meninggalkannya karena sistem pointer dapat dikatakan sangat rumit, sehingga *Java* menjadi lebih mudah digunakan [KAD-09:2].

2.8 Extensible Markup Language (XML)

XML (eXtensible Markup Language) merupakan bahasa web turunan dari SGML (Standart Generalized Markup Language) yang ada sebelumnya. XML hampir sama dengan HTML, dimana keduanya sama-sama turunan dari SGML. Teknologi XML dikembangkan mulai tahun 1966 dan mendapatkan pengakuan dari Worl Wide Web Consortium (W3C) pada bulan Februari 1998. Sedangkan SGML sendiri telah dikembangkan pada awal tahun 1980-an. Pada saat HTML dikembangkan pada tahun 1990, para penggagas XML mengadopsi bagian paling penting SGML dan dengan berpedoman pada pengembangan HTML menghasilkan bahasa markup yang tidak kalah hebatnya dengan SGML.[ITS -13 : 1]. Berikut adalah contoh format data XML :

```
<?xml version="1.0"?>
        <product barcode="2394287410">
        <manufacturer>Verbatim</manufacturer>
        <name>DataLife MF 2HD</name>
        <quantity>10</quantity>
        <size>3.5"</size>
        <color>black</color>
        <description>floppy disks</description>
</product>
```

Gambar 2.14 Format data XML
Sumber : [ITS – 13 : 2]

XML merubah cara kita berpikir untuk mengembangkan suatu software terutama aplikasi web. Masalah yang kita sekarang adalah bagaimana caranya untuk bertukar informasi antar satu aplikasi dengan aplikasi lain. Kadang kolaborasi antara satu aplikasi dengan aplikasi yang lain masih harus ditentukan dengan spesifikasi aplikasi tersebut. Padahal seharusnya kita hanya perlu mendapatkan informasi data bukan mengerti cara kerja aplikasi lain itu, disinilah visi internet yang belum tercapai. Visi ini adalah dunia internet dimana PC, server, smart devices dan internet-based device dapat berkolaborasi tanpa ada

halangan. Bisnis-bisnis akan dapat bertukar data menyediakan customized dan comprehensive solusi kepada customer. Dan yang paling utama adalah informasi yang dibutuhkan dapat diakses dari mana saja dan dengan computing device, platform, atau aplikasi yang kita gunakan. XML dapat memungkinkan pertukaran informasi atau data antar device (server, PCs, smart device, aplikasi, dan situs web). Data ini akan menjadi independent (unlocked), memudahkannya untuk diorganisir, diprogram, dan dirubah, dan ditukar antar situs web atau aplikasi apa saja. Karena kebutuhan ini, maka makin banyak teknologi berbasis XML yang keluar. Contohnya adalah SOAP (Simple Project Acces Protocol) dan UDDI (Universal Description Discovery and Integration).

2.9 Pemrograman Android

Pengertian Android adalah sistem operasi berbasis linux yang dipergunakan sebagai pengelola sumber daya perangkat keras, baik untuk ponsel, *smartphone* dan juga PC tablet. Secara umum Android adalah *platform* yang terbuka (*Open source*) bagi para pengembang untuk menciptakan aplikasi mereka sendiri untuk digunakan oleh berbagai piranti bergerak.

Android merupakan system operasi berbasis linux yang bahasa pemrograman aplikasinya dapat kita buat menggunakan *java*. Dengan pengetahuan seputar *java* yang telah mencukupi, anda dapat membuat aplikasi berbasis android [PRA - 12].

2.10 Black-Box Testing

Black-box testing atau *behavioral testing* berfokus pada persyaratan fungsional perangkat lunak [PRE-01:459]. Dengan demikian, pengujian *black-box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi *input* yang sepenuhnya menggunakan semua persyaratan fungsional untuk semua program. Pengujian *black-box* bukan merupakan alternatif dari teknik *white-box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *white-box*.

Pengujian *black-box* berusaha menemukan kesalahan dalam kategori berikut :

1. Fungsi-fungsi yang tidak benar atau hilang

2. Kesalahan *interface*
3. Kesalahan dalam struktur data atau akses *database* eksternal
4. Kesalahan kinerja
5. Inisialisasi dan kesalahan terminasi.

Tidak seperti pengujian *white-box*, yang dilakukan pada saat awal proses pengujian, pengujian *black-box* cenderung diaplikasikan selama tahap akhir pengujian. Karena pengujian *black-box* memperhatikan struktur kontrol, maka perhatian berfokus pada domain informasi. Pengujian didesain untuk menjawab pertanyaan-pertanyaan berikut :

1. Bagaimana validitas fungsional diuji ?
2. Kelas input apa yang akan membuat *test case* menjadi baik ?
3. Apakah sistem sangat sensitif terhadap harga input tertentu ?
4. Bagaimana batasan dari suatu data diisolasi ?
5. Kecepatan dan *volume* data apa yang dapat ditolerir oleh sistem ?
6. Apa pengaruh kombinasi tertentu dari data terhadap operasi sistem ?

2.10.1. Strategi Pengujian

Strategi untuk pengujian perangkat lunak mengintegrasikan metode desain *test case* perangkat lunak ke dalam sederetan langkah yang direncanakan dengan baik, dan hasilnya adalah konstruksi perangkat lunak yang berhasil [PRE-01:477]. Sejumlah strategi pengujian perangkat lunak telah diusulkan di dalam literatur. Strategi pengujian harus mengakomodasi pengujian tingkat rendah yang diperlukan untuk membuktikan bahwa segmen kode sumber yang kecil telah diimplementasikan dengan tepat, demikian juga pengujian tingkat tinggi yang memvalidasi fungsi-fungsi sistem mayor yang berlawanan dengan kebutuhan pelanggan [PRE-01:481].

2.10.2. Pengujian Validasi

Pada kulminasi pengujian terintegrasi, perangkat lunak secara lengkap dirakit sebagai suatu paket; kesalahan *interfacing* telah diungkap dan dikoreksi, dan seri akhir dari pengujian perangkat lunak, yaitu pengujian validasi dapat dimulai. Validasi dapat ditentukan dengan berbagai cara, tetapi definisi yang

sederhana adalah bahwa validasi berhasil bila perangkat lunak berfungsi dengan cara yang dapat diharapkan secara bertanggung jawab oleh pelanggan. Validasi perangkat lunak dicapai melalui sederetan pengujian *black-box* yang memperlihatkan konformitas dengan persyaratan. Rencana pengujian menguraikan kelas-kelas pengujian yang akan dilakukan, dan prosedur pengujian menentukan *test case* spesifik yang akan digunakan untuk mengungkap kesalahan dalam konformitas dengan persyaratan. Baik rencana dan prosedur didesain untuk memastikan apakah semua persyaratan fungsional dipenuhi; semua persyaratan kinerja dicapai; dokumentasi benar dan direkayasa oleh manusia: dan persyaratan lainnya dipenuhi (transportabilitas, kompatibilitas, pembetulan kesalahan, maintanabilitas) [PRE-01:495].

2.10.3. User Acceptance Test (UAT)

UAT adalah konfirmasi melalui pengujian, bahwa sistem disampaikan memenuhi semua *requirement*, fungsi yang sesuai dengan parameter desain, dan memenuhi semua proses bisnis, teknis, dan kepentingan *user*.

Perencanaan UAT dimulai dengan tahap pengembangan konsep dengan definisi kriteria pengujian. Parameter pengujian pertama kali ditentukan dengan *project scope* kemudian dilanjutkan ke tahap *analisis requirement.* Kriteria pengujian akan digunakan sebagai bagian dari *requirement traceability* untuk membuat desain, pengembangan, pengujian, dan penerimaan sistem di masyarakat. Berikut adalah kriteria pengujian UAT yang efektif:

- Pengujian didasarkan pada persyaratan fungsional dan non fungsional.
- Pengujian harus spesifik , jelas , terukur, dan realistis.
- Mengandung parameter berhasil atau gagal.
- Detail aspek pengujian sistem.
- Dapat menunjukkan fitur sistem itu kritis (diperlukan) atau tidak kritis.
- Dapat mengidentifikasi kesalahan yang tidak dapat diterima.

Dalam tahap perencanaan, tim mendefinisikan jadwal dan memperkirakan biaya bagi pengguna UAT pada tahapan pelaksanaan. Rencana untuk pengujian secara progresif diuraikan dalam tahap analisis *requirement* dengan

pengembangan rencana induk test, yang membahas pengguna pengujian penerimaan yang direncanakan secara rinci . Pelaksanaan pengujian penerimaan pengguna berhasil jika:

- Pengguna menerima fitur yang telah disetujui.
- Pengguna sistem melakukan tes.
- Lingkungan pengujian mensimulasika kondisi nyata penggunaan aplikasi.
- Pengujian dilakukan pada sebuah sistem selesai yang telah melewati pengujian unit, pengujian integrasi, dan pengujian sistem.
- Pengujian dilakukan menggunakan uji kasus yang mencakup semua skenario. Uji kasus menggambarkan fungsi (skenario) sedang diuji, input, hasil yang diharapkan, hasil aktual, status keberhasilan dan strategi perbaikan untuk kondisi masalah yang ditemukan, tanggal uji coba dan waktu, nama orang / peran user yang melakukan test.
- Semua data telah bermigrasi / dikonversi sebelum pengujian UAT.
- *Test case* otomatis dilakukan dengan *test script*. (Bila memungkinkan)

Pengujian UAT juga membutuhkan verifikasi bahwa sistem disampaikan memenuhi semua kriteria penerimaan pengguna ditentukan sebelumnya dalam proyek untuk dikatakan sukses [DOI - 13].