

BAB II

LANDASAN TEORI

2.1 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977 (Charras, 2003).

Algoritma Knuth-Morris-Pratt (KMP) merupakan algoritma yang digunakan untuk melakukan proses pencocokan string. Algoritma ini merupakan jenis *Exact String Matching Algorithm* yang merupakan pencocokan string secara tepat dengan susunan karakter dalam string yang dicocokkan memiliki jumlah maupun urutan karakter dalam string yang sama.

Algoritma Knuth-Morris-Pratt menggunakan fungsi pembatas (*border function*) yang digunakan untuk menghitung urutan keberapa perbandingan harus dilakukan. *Border function* dihitung dengan menghitung panjang *prefix* yang ada disebuah *pattern* yang sama dengan *suffix*-nya (Wibowo, 2011).

Misalkan A adalah alfabet dan $x = x_1x_2\dots x_k$, $k \in \mathbb{N}$, adalah *string* yang panjangnya k yang dibentuk dari karakter-karakter di dalam alfabet A . Awalan (*prefix*) dari x adalah *substring* u dengan $u = x_1x_2\dots x_{k-1}$, $k \in \{1, 2, \dots, k-1\}$ dengan kata lain, x diawali dengan u . Akhiran (*suffix*) dari x adalah *substring* u dengan $u = x_{k-b}x_{k-b+1}\dots x_k$, $k \in \{1, 2, \dots, k-1\}$ dengan

kata lain, x diakhiri dengan v . Pinggiran (*border*) dari x adalah *substring* r sedemikian sehingga $r = x_1x_2\dots x_{k-1}$ dan $u = x_k - b \ x_{k-b+1} \dots x_k$, $k \in \{1, 2, \dots, k-1\}$ dengan kata lain, pinggiran dari x adalah *substring* yang keduanya awalan dan juga akhiran sebenarnya dari x .

2.1.1 Fungsi Pinggiran

Algoritma Knuth-Morris-Pratt melakukan proses awal terhadap *pattern* dengan menghitung fungsi pinggiran. Dengan adanya fungsi pinggiran ini, maka dapat dicegah pergeseran yang tidak berguna, seperti yang terjadi pada algoritma Brute Force.

Fungsi pinggiran $b(j)$ didefinisikan sebagai ukuran awalan terpanjang dari P yang merupakan akhiran dari $P[1..j]$. Sebagai contoh, tinjau *pattern* $P = ababaa$. Nilai F untuk setiap karakter di dalam P adalah sebagai berikut:

j	1	2	3	4	5	6
$P[j]$	a	b	a	b	a	a
$b(j)$	0	0	1	2	3	1

Kompleksitas algoritma pencocokan string dengan Knuth-Morris-Pratt ini dihitung dari kompleksitas untuk menghitung fungsi pinggiran dan pencarian *string*. Untuk menghitung fungsi pinggiran dibutuhkan waktu $O(m)$, sementara untuk pencarian string dibutuhkan $O(n)$, sehingga kompleksitas waktu algoritma Knuth-Morris-Pratt adalah $O(m + n)$.

2.2 Al-Qur'an

2.2.1 Pengertian Al-Qur'an

Secara etimologi, lafadz Al-Qur'an berasal dari bahasa Arab, yaitu akar kata dari *qara'a*, yang berarti membaca. Al-Qur'an adalah kalam Allah yang menjadi mukjizat yang diturunkan kepada Nabi Muhammad saw dengan lafaz dan maknanya dari dengan perantaraan malaikat Jibril as yang tertulis dalam mushaf yang disampaikan secara mutawatir, dimulai dengan Surat al-fatihah dan diakhiri dengan surat an-nas (Tim Penyusun Yayasan Bimantara, 1997).

Al-Qur'an secara harfiah berarti bacaan yang sempurna merupakan suatu nama pilihan Allah yang sungguh tepat, karena tiada satu bacaan pun sejak manusia mengenal tulis-baca lima ribu tahun yang lalu yang dapat menandingi Al-Qur'an Al Karim, bacaan sempurna lagi mulia (Shihab, 1996).

2.2.2 Sejarah Turunnya Al-Qur'an

Allah SWT menurunkan Al-Qur'an dengan perantaraan malaikat Jibril sebagai pengantar wahyu yang disampaikan kepada Nabi Muhammad SAW di goa hiro pada tanggal 17 ramadhan. Ketika itu Nabi Muhammad SAW berusia 41 tahun, dan ayat yang pertama kali diturunkan yaitu surat al-alaq ayat 1-5. Sedangkan ayat yang terakhir diturunkan yaitu surat al-maidah ayat 3, diturunkan pada tanggal 10 zulhijjah tahun 10 hijriah. Al-Qur'an diturunkan secara berangsur-angsur dengan lama waktu 22 tahun, 2 bulan, 22 hari, atau 23 tahun. 13 tahun turun di kota Makkah dan 10 tahun turun di kota Madinah.

2.2.3 Klasifikasi Ayat-Ayat Al-Qur'an

Berdasarkan syamil Al-Qur'an terjemah tafsir per kata klasifikasi ayat-ayat Al-Qur'an dibagi menjadi 15 bab yaitu :

1. Sekitar Arkanul Islam
2. Iman
3. Al-Quran
4. Ilmu dan Cabang-cabangnya
5. Amal
6. Dakwah Kepada Allah
7. Jihad
8. Manusia dan Hubungan Kemasyarakatan
9. Akhlak
10. Peraturan yang Berhubungan dengan Harta
11. Hal-hal yang Berkaitan dengan Hukum
12. Negara dan Masyarakat
13. Pertanian dan Perdagangan
14. Sejarah dan Kisah-kisah
15. Agama-agama

2.3 *System Development Life Cycle (SDLC)*

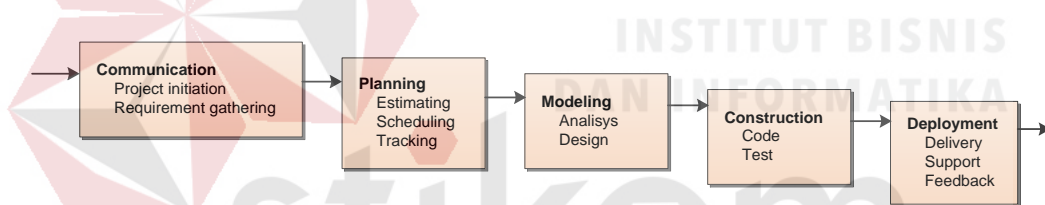
Menurut Turban, et al (2003), System Development Life Cycle (SDLC) atau Siklus Hidup Pengembangan Sistem adalah metode pengembangan sistem tradisional yang digunakan sebagian besar organisasi saat ini. SDLC adalah

kerangka kerja (framework) yang terstruktur yang berisi proses-proses sekuensial di mana sistem informasi dikembangkan.

Menurut Pressman (2010), *System Development Life Cycle* atau Siklus Hidup Pengembangan Sistem adalah proses perancangan sistem serta metodologi yang digunakan untuk mengembangkan sistem-sistem tersebut

2.3.1 *Waterfall Model*

Menurut Pressman (2010), salah satu model pengembangan sistem adalah dengan model waterfall. Waterfall model adalah model yang paling populer dan sering dianggap sebagai pendekatan klasik dalam daur hidup pengembangan sistem. Adapun tahapannya sebagai berikut :



Gambar 2.1 Waterfall Model (Pressman,2010)

a. *Communication*

Pada tahap ini dilakukan inisialisasi proyek, seperti identifikasi masalah, menganalisis masalah yang ada dan tujuan yang akan dicapai dalam membangun sebuah sistem. Selain itu dilakukan juga *requirements gathering*, dimana akan dikumpulkan *requirement* dari user. *Requirement* juga bisa dikatakan tahap pengumpulan data dari user dengan menggunakan teknik-teknik pengumpulan data yang telah ada.

b. *Planning*

Tahap ini merupakan tahap dimana akan dilakukan estimasi mengenai kebutuhan-kebutuhan yang diperlukan untuk membuat sebuah sistem. Pada tahap jua ini akan dirumuskan tujuan pengembangan, serta mengidentifikasi apakah masalah-masalah yang ada dapat diselesaikan melalui pengembangan sistem.

c. *Modeling*

Tahap ini merupakan tahap analisis dan perancangan, dimana perancang menerjemahkan kebutuhan sistem kedalam representasi untuk menilai kualitas sebelum tahap selanjutnya dikerjakan. Tahap ini lebih difokuskan pada atribut program, seperti menganalisa interaksi objek dan fungsi pada sistem, arsitektur perangkat lunak, dan merancang *user interface*.

d. *Construction*

Tahap ini merupakan tahap dimana perancangan yang sebelumnya telah dilakukan diterjemahkan kedalam bahasa yang dimengerti oleh mesin sehingga menjadi sebuah sistem. Setelah itu dilakukan pengetesan / pengujian terhadap sistem yang telah dibuat.

e. *Deployment*

Setelah proses pengkodean dan pengujian selesai, dilakukan pengiriman yang artinya implementasi kepada masyarakat luas. Pada tahap ini juga dilakukan pemeliharaan, perbaikan, dan pengembangan agar sistem tersebut tetap dapat berjalan sebagaimana fungsinya.

Kelebihan dari model ini adalah selain karena pengaplikasian menggunakan model ini mudah, kelebihan dari model ini adalah ketika semua kebutuhan sistem dapat didefinisikan secara utuh, eksplisit, dan benar di awal proyek, maka *Software Engineering* (SE) dapat berjalan dengan baik dan tanpa masalah. Kekurangan yang utama dari model ini adalah kesulitan dalam mengakomodasi perubahan setelah proses dijalani. Fase sebelumnya harus lengkap dan selesai sebelum mengerjakan fase berikutnya.

2.4 Android

Android adalah sebuah sistem operasi untuk perangkat mobile berbasis Linux yang mencakup sistem operasi, *middleware* dan aplikasi (Safaat, 2011). Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka. Android merupakan generasi baru platform mobile, platform yang memberikan pengembang untuk melakukan pengembangan sesuai dengan yang diharapkannya.

Dari perkembangan sistem operasi Android ini yang sekarang menjadi sangat populer karena bersifat *open source* menjadikannya sebagai sistem operasi yang banyak diminati oleh banyak pengguna. Adapun beberapa kelebihan dari sistem operasi Android adalah sebagai berikut (Safaat, 2011) :

a. *Complete Platform*

Sistem operasi Android adalah sistem operasi yang banyak menyediakan *tools* yang berguna untuk membangun sebuah aplikasi yang kemudian aplikasi tersebut dapat lebih dikembangkan lagi oleh para *developer*.

b. *Open Source Platform*

Platform Android yang bersifat *open source* menjadikan sistem operasi ini mudah dikembangkan oleh para *developer* karena bersifat terbuka.

c. *Free Platform*

Developer dengan bebas bisa mengembangkan, mendistribusikan dan memperdagangkan sistem operasi Android tanpa harus membayar royalti untuk mendapatkan *license*.

Pengembang memiliki beberapa pilihan ketika membuat aplikasi yang berbasis Android. Kebanyakan pengembang menggunakan Eclipse yang tersedia secara bebas untuk merancang dan mengembangkan aplikasi Android. Eclipse adalah IDE yang paling populer untuk pengembangan Android, karena memiliki Android plug-in yang tersedia untuk memfasilitasi pengembangan Android. Selain itu Eclipse juga mendapat dukungan langsung dari Google untuk menjadi IDE pengembangan aplikasi Android, ini terbukti dengan adanya penambahan plugins dari Eclipse untuk bisa membuat project Android dimana source software langsung dari situs resminya google. Tetapi hal diatas tidak menutup kemungkinan untuk menggunakan IDE yang lain seperti Netbeans untuk melakukan pengembangan Android.

2.4.1 Versi Android

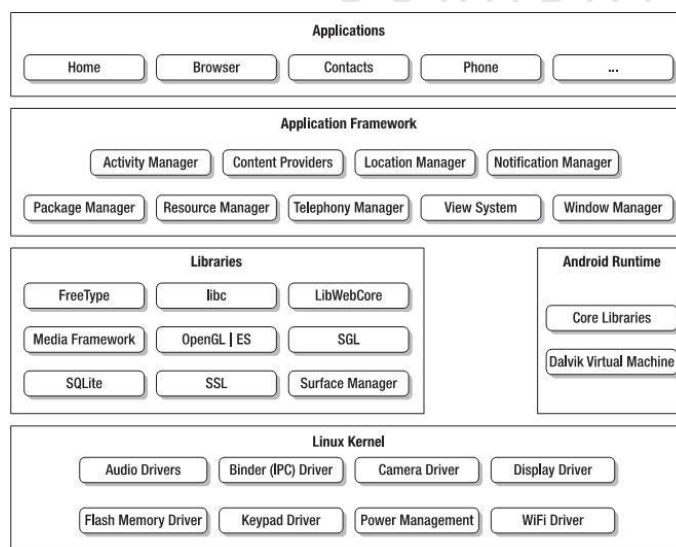
Perkembangan sistem operasi Android dari awal pertama dipakai hingga saat ini terdapat berbagai versi Android yang telah di rilis. Adapun versi – versi Android yang telah dirilis adalah sebagai berikut (Safaat, 2011) :

a. Android versi 1.1

- b. Android versi 1.5 (*Cupcake*)
- c. Android versi 1.6 (*Donut*)
- d. Android versi 2.0 / 2.1 (*Eclair*)
- e. Android versi 2.2 Froyo (*Frozen Yoghurt*)
- f. Android versi 2.3 (*Gingerbread*)
- g. Android versi 3.0/3.1 (*Honeycomb*)
- h. Android versi 4.0 ICS (*Ice Cream Sandwich*)
- i. Android versi 4.1/4.2/4.3(*Jelly Bean*)
- j. Android versi 4.4 (*KitKat*)

2.4.2 Arsitektur Android

Arsitektur sistem terdiri atas 5 layer, pemisahan layer bertujuan untuk memberikan abstraksi sehingga memudahkan pengembangan aplikasi. Layer-layer tersebut adalah layer aplikasi, layer framework aplikasi, layer libraries, layer runtime, dan layer kernel. Gambar 2.1 memberikan gambaran umum komponen-komponen dalam arsitektur sistem operasi Android.



Gambar 2.2 Arsitektur Android (iyonspika27.blogspot.com)

Secara garis besar Arsitektur Android dapat dijelaskan dan digambarkan sebagai berikut:

a. *Applications* dan *Widgets*

Applications dan *Widgets* merupakan layer pertama pada OS android. ini adalah layar dimana berhubungan dengan aplikasi saja, dimana biasanya kita download aplikasi kemudian kita lakukan instalasi dan jalankan aplikasi tersebut, di layer inilah terdapat seperti aplikasi inti termasuk klien email, program sms, kalender, peta, browser, kontak, dan lain-lain. Semua aplikasi ditulis menggunakan bahasa pemrograman java.

b. Aplikasi *Frameworks*

Applications Framework merupakan layer dimana para pembuat aplikasi menggunakan komponen-komponen yang ada di sini untuk membuat aplikasi mereka. Pada layer ini terdapat komponen seperti *views*, *content provider*, *resource manager*, *notification manager*, *activity manager*.

c. *Libraries*

Libraries ini adalah layer dimana fitur-fitur android berada, biasanya para pembuat aplikasi kebanyakan mengakses *libraries* untuk menjalankan aplikasinya. Berjalan di atas kernel, layer ini meliputi berbagai library C / C++ inti seperti Libc dan SSL.

d. *Android Runtime*

Layer yang membuat aplikasi android dapat dijalankan dimana dalam prosesnya menggunakan implementasi Linux. Dalvik Virtual Machine (DVM) merupakan mesin yang membentuk dasar kerangka aplikasi Android.

e. Linux Kernel

Linux kernel adalah layer dimana inti dari sistem operasi android itu sendiri, berisi file-file sistem yang mengatur sistem *processing*, *memory*, *resource*, *drivers*, dan sistem-sistem operasi android lainnya. Linux Kernel yang digunakan Android adalah Linux Kernel *release 2.6* dan versi 3.x pada Android versi 4.0 ke atas. Kernel ini berbasis *monolithic*.

2.4.3 Android Development Tools (ADT)

ADT adalah kepanjangan dari *Android Development Tools* yang menjadi penghubung antara IDE Eclipse dengan Android SDK (Safaat, 2011). ADT ini adalah sebuah *plugin* untuk Eclipse yang didesain untuk membangun aplikasi Android baru, membuat *user interface*, menambahkan komponen berdasarkan *framework* API Android, debug aplikasi dan menjalankan emulator Android.

2.4.4 Android Software Development Kit (SDK)

Android SDK adalah *tools* API (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada platform Android menggunakan bahasa pemrograman Java (Safaat, 2011). Android merupakan subset perangkat lunak untuk smartphone yang meliputi sistem operasi, middleware dan aplikasi kunci yang direlease oleh Google.

Saat ini disediakan Android SDK sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada platform Android menggunakan bahasa pemrograman Java. Sebagai platform aplikasi netral, Android memberi Anda

kesempatan untuk membuat aplikasi yang bukan merupakan aplikasi bawaan Smartphone.

Beberapa fitur-fitur Android yang paling penting adalah:

- a. Framework Aplikasi yang mendukung penggantian komponen dan reusable.
- b. Mesin Virtual Dalvik dioptimalkan untuk perangkat mobile.
- c. Integrated browser berdasarkan engine open source WebKit.
- d. Grafis yang dioptimalkan dan didukung oleh libraries grafis 2D, grafis 3D berdasarkan spesifikasi OpenGL ES 1.0 (Opsional akselerasi hardware).
- e. SQLite untuk menyimpan data.
- f. Media Support yang mendukung audio, video, dan gambar.
- g. Bluetooth, EDGE, dan WiFi (tergantung hardware).
- h. Kamera, GPS, kompas, dan accelerometer (tergantung hardware)
- i. Lingkungan Development yang lengkap dan kaya termasuk perangkat emulator, tool untuk debugging, profil dan kinerja memori, dan plugin untuk IDE Eclipse.

2.4.5 Android Virtual Device (AVD)

Android *Virtual Device* (AVD) yang merupakan emulator untuk menjalankan program aplikasi Android yang dibuat (Safaat, 2011). AVD ini yang selanjutnya digunakan sebagai tempat untuk test dan menjalankan aplikasi Android yang telah dibuat. Dengan AVD ini, *developer* bisa mengembangkan dan mencoba aplikasi Android tanpa harus menggunakan perangkat Android yang sebenarnya. Sebelum menggunakan AVD harus menentukan karakteristiknya, misalkan dalam menentukan versi Android, jenis

dan ukuran layar dan besarnya memori. AVD bisa dibuat sebanyak yang kita inginkan.

2.5 *Unified Modeling Language (UML)*

UML merupakan singkatan dari *Unified Modelling Language* adalah sekumpulan pemodelan konvensi yang digunakan untuk menentukan atau menggambarkan sebuah sistem perangkat lunak dalam kaitannya dengan objek. (Whitten, 2004).

Notasi UML dibuat sebagai kolaborasi dari Grady Booch, DR. James Rumbaugh, Ivar Jacobson, Rebecca Wirfs-Brock, Peter Yourdon, dan lainnya. Jacobson menulis tentang pendefinisian persyaratan – persyaratan sistem yang disebut *use case*. Juga mengembangkan sebuah metode untuk perancangan sistem yang disebut *Object Oriented Software Engineering (OOSE)* yang berfokus pada analisis. Booch, Rumbaugh, dan Jacobson biasa disebut dengan tiga sekawan (*tree amigos*). Semuanya bekerja di Rational Software Corporation dan berfokus pada standarisasi dan perbaikan ulang UML.

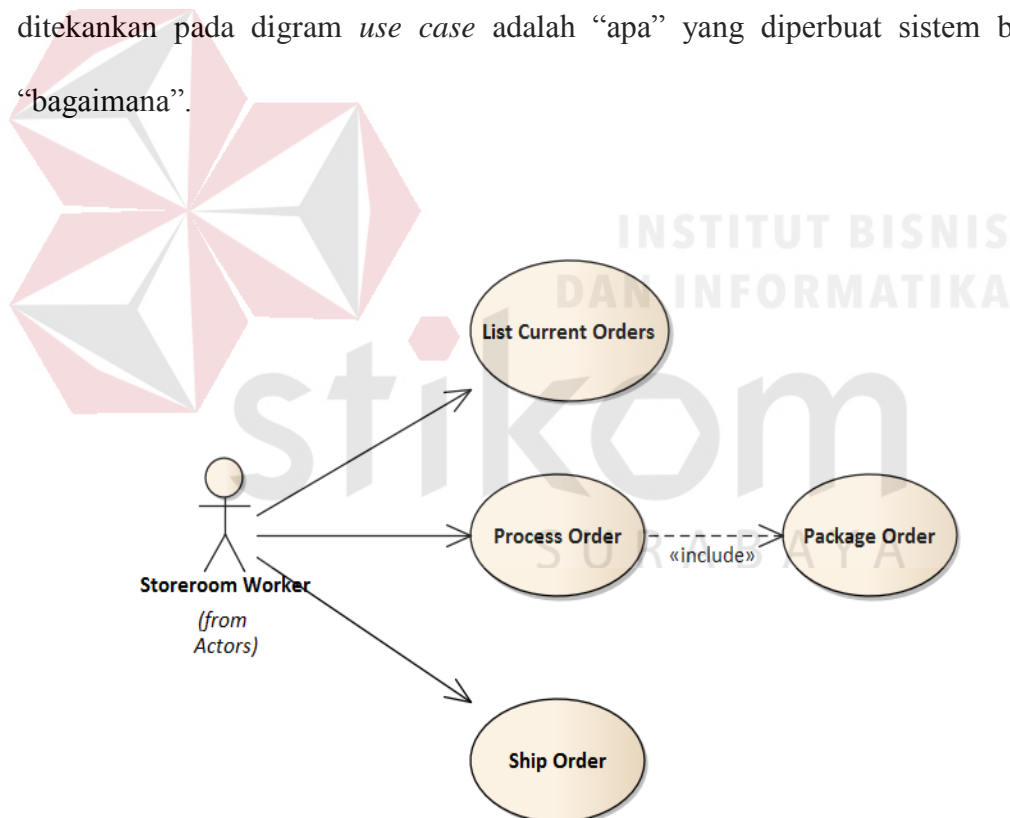
Penggabungan beberapa metode menjadi UML dimulai 1993. Setiap orang dari tiga sekawan di rational mulai menggabungkan idenya dengan metode – metode lainnya. Pada akhir tahun 1995 *Unified Method* versi 0.8 diperkenalkan. *Unified Method* diperbaiki dan diubah menjadi UML pada tahun 1996, UML 1.0 disahkan dan diberikan pada Object Technology Group (OTG) pada tahun 1997, dan pada tahun itu juga beberapa perusahaan pengembang utama perangkat lunak mulai mengadopsinya. Pada tahun yang sama OTG merilis UML 1.1 sebagai standar industri.

2.5.1 Diagram UML

Menurut Whitten, et al (2007) UML menawarkan diagram-diagram dengan perspektif yang berbeda untuk memodelkan suatu sistem. Berikut adalah macam-macam diagram serta tujuannya:

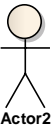

1. Diagram *Use case*

Secara grafis diagram *use case* menggambarkan interaksi antara pengguna dengan sistem. fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan pada digram *use case* adalah “apa” yang diperbuat sistem bukan “bagaimana”.



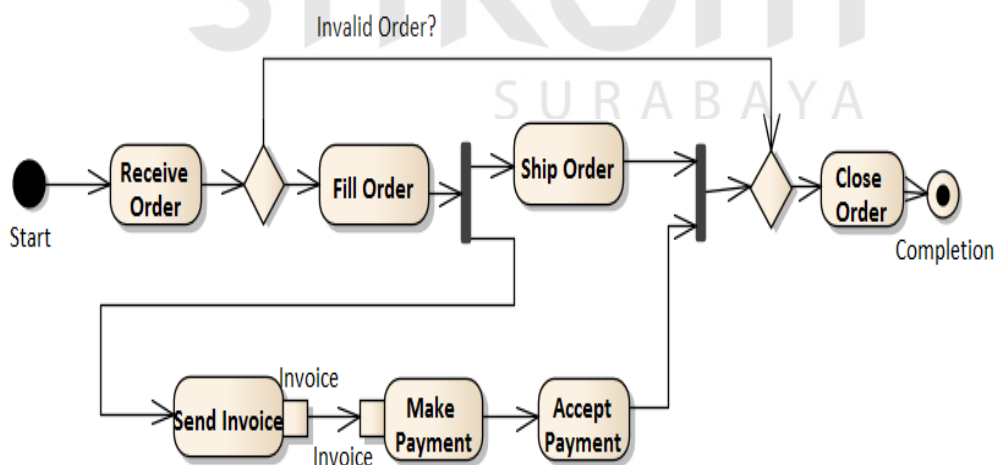
Gambar 2.3 Contoh Diagram *Use Case* (sparxsystems.com)

Tabel 2.1 Penjelasan Diagram *Use Case*


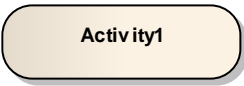





Nama Simbol	Gambar	Penjelasan
Aktor		Apapun yang mempresentasikan seseorang atau sistem, yang berinteraksi dengan sistem.
Use Case		Gambaran fungsionalitas dari suatu sistem, sehingga pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun.

2. Diagram Activity

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

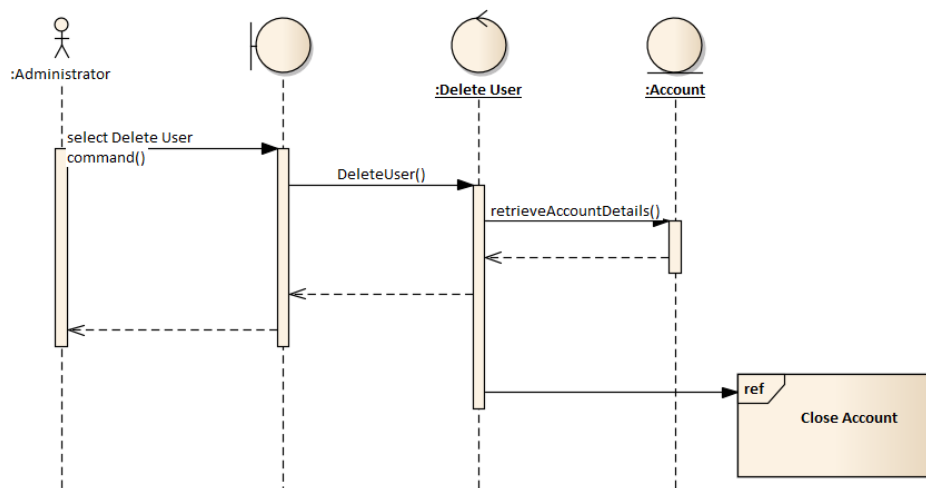
Gambar 2.4 Contoh Diagram *Activity* (sparxsystems.com)

Tabel 2.2 Penjelasan Diagram *Activity*

Nama Simbol	Gambar	Penjelasan
Initial Node		menggambarkan awal dari proses.
Activity		Menggambarkan aktifitas atau kegiatan yang perlu dilakukan.
Flow		Menggambarkan alur atau sasaran yang mengawali kegiatan.
Decision		Menggambarkan sebuah keputusan.
Fork		Menggambarkan kegiatan parallel.
Join		Menggambarkan penggabungan.
Activity Final		Menggambarkan akhir dari sebuah proses.

3. Diagram Sequence

Diagram *Sequence* secara grafis menggambarkan bagaimana objek berinteraksi antara satu sama lain melalui pesan ekusi pada sebuah *use case* atau operasi.

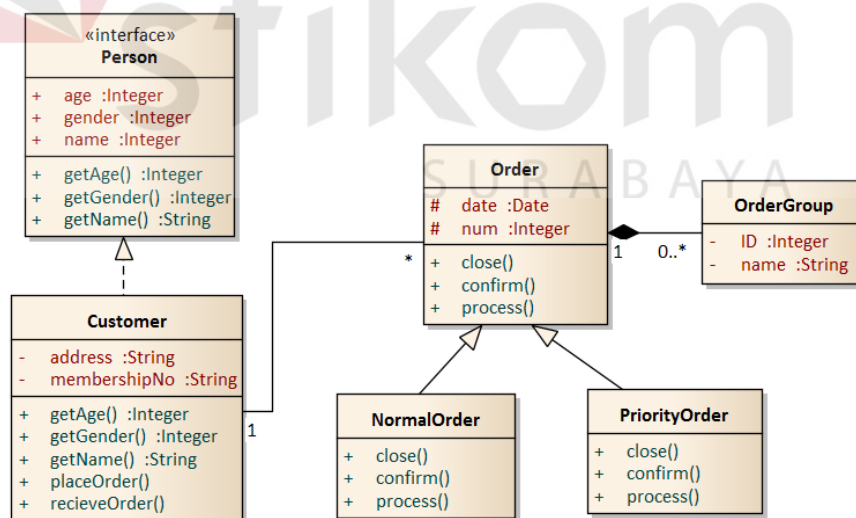
Gambar 2.5 Contoh Diagram *Sequence* (sparxsystems.com)

Menurut Pressman (2010), elemen-elemen yang terdapat pada diagram sequence antara lain yaitu :

- Actor (user)*, sebagai subjek yang berinteraksi dengan sistem.
- Object*, sebagai objek atau komponen suatu sistem.
- Separator*, sebagai batas antar sub sistem.
- Message*, prosedur pesan atau proses antar elemen komponen.
- Return message*, sebagai pesan balik antar objek.

4. Diagram Kelas

Diagram kelas adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Kelas menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).



Gambar 2.6 Contoh Diagram Kelas (sparxsystems.com)

Elemen utama pada Diagram kelas adalah kotak yang digunakan untuk merepresentasikan kelas-kelas dan *interface*. Setiap kotak dibagi menjadi bagian-bagian horisontal. Bagian paling atas menyatakan nama kelas tersebut. Bagian

tengah menyatakan daftar atribut-atribut yang dimiliki oleh kelas tersebut. Sebuah atribut menunjuk pada suatu objek yang dikenali oleh kelas tersebut. Dan bagian paling bawah menyatakan operasi-operasi yang dilakukan oleh kelas tersebut.

2.6 SQLite

Menurut A Kreibich (2010), SQLite Merupakan perangkat lunak yang menyediakan *relational database management system* (RDBMS). *relational database system* digunakan untuk menyimpan data pada suatu tabel yang besar dan memproses perintah *query* yang kompleks dan menggabungkan data dari berbagai tabel untuk menghasilkan laporan dan rangkuman data. Selain itu untuk menyimpan dan mengelola data, SQLite juga dapat melakukan perintah *query* yang kompleks dari beberapa tabel untuk menghasilkan data yang diinginkan. SQLite merupakan *database engine* yang ringan dibandingkan dengan *database engine* yang lainnya.

Beberapa fitur dari SQLite adalah sebagai berikut (A Kreibich, 2010) :

a. *Serverless*

SQLite memerlukan proses server atau sistem yang terpisah untuk mengoperasikannya. SQLite *library* mengakses databasenya secara langsung.

b. *Zero Configuration*

Tidak ada *server* berarti tidak ada pengaturan. Membuat sebuah *instance* database semudah membuka file.

c. *Cross-Platform*

Database pada SQLite berada dalam file *cross-platform* tunggal yang tidak memerlukan administrasi. Sehingga dapat digunakan di berbagai sistem operasi.

d. *Self-Contained*

Sebuah *library* berisi seluruh sistem database yang terintegrasi langsung ke *application-host*.

e. *Small Runtime Footprint*

Menggunakan sedikit memory untuk *library* databasenya.

f. *Transactional*

g. *Full-featured*

SQLite mendukung penggunaan bahasa SQL standar.

h. *Highly Reliable*

Dalam sistem android memiliki beberapa teknik untuk melakukan penyimpanan data. Teknik yang umum digunakan adalah sebagai berikut :

- a. *Shared Preference* yaitu menyimpan data beberapa nilai dalam bentuk *group key* yang dikenal dengan *preference*.
- b. *Files* yaitu menyimpan data dalam file, juga dapat berupa *write* dan *read* pada file.
- c. *SQLite Database* yaitu menyimpan data dalam bentuk *database*.
- d. *Content Providers* yaitu menyimpan data dalam *content provider service*.

2.7 Software Testing

Menurut Hetzel (1998), testing adalah segala kegiatan yang bertujuan untuk mengevaluasi kualitas program atau sistem dan menentukan bahwa program atau sistem memenuhi kebutuhan yang diharapkan.

Menurut Lewis (2009), *software testing* adalah aktivitas menjalankan serangkaian eksekusi yang dinamis pada program *software* setelah *source code software* tersebut telah dikembangkan. *Software testing* dilakukan untuk menemukan dan memperbaiki sebanyak mungkin potensi kesalahan sebelum *software* tersebut digunakan oleh pelanggan atau *end user*.

2.7.1 Unit Testing

A. White Box Testing

White box testing kadang disebut juga glass box testing atau clear box testing, adalah suatu metode desain test case yang menggunakan struktur kendali dari desain prosedural (Romeo, 2003).

Sedangkan menurut Pressman (2010), White Box Testing adalah metode desain test case yang menggunakan struktur kontrol desain procedural untuk memperoleh test case. Dengan menggunakan metode pengujian ini akan didapatkan test case yang :

1. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali.
2. Menggunakan semua keputusan logis pada sisi *true* dan *false*.
3. Mengeksekusi semua *looping* pada batasan tertentu.
4. Menggunakan struktur data internal yang menjamin validitasnya.

A.1 Cyclomatic Complexity

Cyclomatic Complexity adalah pengukuran software yang memberikan pengukuran kuantitatif dari kompleksitas logika program (Romeo, 2003). Pada konteks basis path testing, nilai yang dihitung bagi *cyclomatic complexity* menentukan jumlah jalur-jalur yang independen dalam kumpulan basis suatu program.

Jalur independen adalah tiap jalur pada program yang memperlihatkan 1 kelompok baru dari pernyataan proses atau kondisi baru. Rumus yang digunakan untuk menghitung *cyclomatic complexity* yaitu :

$$V(G) = E (\text{Edge}) - N (\text{Node}) + 2$$

$$V(G) = P (\text{Predicted Node}) + 1$$

B. Black Box Testing

Menurut Black (2007), *black box testing* adalah melakukan pengujian terhadap apa yang dilakukan oleh sistem, khususnya perilaku dan juga masalah bisnis. *Black box testing* bertujuan untuk mengidentifikasi *bug-bug* yang ada pada hasil, kinerja dan juga perilaku sistem. Pengujian ini biasanya dilakukan oleh pihak penguji ketika *integration test*, *system test*, dan *acceptance test*, tetapi juga berguna untuk tahap yang lebih awal untuk membantu membangun *unit test case* dan *component test case* yang lebih baik.

Menurut Perry (2006), *functional testing* juga dapat disebut sebagai *black box testing* karena tidak ada pengetahuan dari logika internal sistem yang digunakan untuk membuat *test case*. Biasanya dalam pengujian fungsional, teknik

validasi lebih digunakan untuk melakukan pengujian. Tim penguji melakukan validasi terhadap *function key* yang ada dan mengobservasi hasilnya.

2.7.2 *System Testing*

System testing adalah serangkaian tes yang berbeda yang tujuan utamanya adalah untuk membangun sistem berbasis komputer (Pressman, 2010). Meskipun setiap test memiliki tujuan yang berbeda, setiap pekerjaan harus diverifikasi bahwa elemen-elemen pada sistem telah terintegrasi dengan baik.

A. *Compatibility Testing*

Compatibility testing merupakan bagian dari non-fungsional testing pada perangkat lunak, adalah pengujian yang dilakukan pada aplikasi untuk mengevaluasi kompatibilitas aplikasi dengan lingkungan komputasi. Lingkungan komputasi mencakup beberapa aspek seperti *hardware*, sistem operasi, database, *browser*, dll.

Pengujian *compatibility* digunakan untuk memeriksa apakah sistem yang dikembangkan mampu berjalan pada *hardware*, *software*, *operating system*, ataupun lingkungan jaringan yang berbeda. Pengujian kompatibilitas berfungsi untuk menentukan set lingkungan yang diharapkan dapat menjalankan sistem yang dikembangkan. Semakin sistem dapat berjalan di banyak jenis perangkat yang berbeda, maka semakin baik aspek kompatibilitasnya.

B. Performance Testing

Performance Testing merupakan pengujian kinerja aplikasi yang dirancang untuk menguji kinerja perangkat lunak dalam konteks sistem yang terintegrasi (Pressman, 2010).

Performance testing bisa digabung dengan *stress testing* dan biasanya membutuhkan instrumentasi *software* dan *hardware*. Oleh karena itu, sering kali membutuhkan *utilitas* dari sumber daya secara eksak. Dengan menginstrumentasikan sistem, tester dapat melingkupi situasi-situasi yang mungkin mempunyai kecenderungan untuk mengarah ke degradasi dan kegagalan sistem.

2.8 Precision

Menurut Hasugian (2006), Perolehan (*recall*) berhubungan dengan kemampuan sistem untuk memanggil dokumen yang *relevan*, sedangkan ketepatan (*precision*) berkaitan dengan kemampuan sistem untuk tidak memanggil dokumen yang tidak *relevan*. *Precision* biasanya menjadi salah satu ukuran yang digunakan untuk menilai keefektifan suatu sistem temu balik informasi.

Precision dapat diartikan sebagai kepersisan atau kecocokan (antara permintaan informasi dengan jawaban terhadap permintaan itu). Jika seseorang mencari informasi di sebuah sistem, dan sistem menawarkan beberapa dokumen, maka kepersisan ini sebenarnya juga adalah relevansi. Artinya, seberapa persis atau cocok dokumen tersebut untuk keperluan pencari informasi, bergantung pada seberapa relevan dokumen tersebut bagi si pencari.

Pada dasarnya, nilai *precision* bernilai antara 0-1. Oleh karena itu, suatu sistem IR (*Information Retrieval*) yang baik diharapkan dapat memberikan

nilai *precision* mendekati 1. Rumus dari *precision* adalah jumlah dokumen *relevan* yang ditemukan dibagi jumlah semua dokumen yang ditemukan.

$$precision = \frac{\text{jumlah dokumen retcieve yang relevan}}{\text{total jumlah dokumen yang diretcive}} \dots(2.1)$$

