BAB 2

LANDASAN TEORI

2.1 Teori-Teori Dasar/Umum

2.1.1 Rekayasa Perangkat Lunak

Menurut Pressman (2009), *software* telah menjadi elemen kunci dalam evolusi sistem berbasis komputer dan produk. Selama 50 tahun terakhir, perangkat lunak telah berkembang pemecah masalah khusus dan perangkat informasi analisis untuk sebuah industri dalam dirinya sendiri.

Software terdiri dari program, data, dan dokumen. Masing-masing *item* ini terdiri dari konfigurasi yang diciptakan sebagai bagian dari proses rekayasa perangkat lunak. Tujuan dari rekayasa perangkat lunak adalah untuk menyediakan kerangka kerja untuk membangun perangkat lunak dengan kualitas yang lebih tinggi.

Berikut ini adalah tahap-tahap umum dalam kerangka proses perancangan *software:*

1. *Communication*

   Tahap ini dibutuhkan untuk menetapkan persyaratan yang efektif.

2. *Planning*

   Tahap ini dibutuhkan untuk mendefinisikan sumber daya, jadwal, dan informasi lain yang berhubungan dengan proyek.

3. *Risk analysis*

   Tahap dibutuhkan untuk menilai risiko baik teknis dan manajemen.

4. *Engineering*

   Tahap dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi.

5. *Construction and release*

   Tahap dibutuhkan untuk membangun, menguji, menginstal, dan memberikan dukungan pengguna, misalnya dokumentasi dan pelatihan.

6. *Customer evaluation*

   Tahap dibutuhkan untuk memperoleh umpan balik pelanggan berdasarkan evaluasi representasi *software* yang diciptakan selama rekayasa kegiatan dan dilaksanakan selama kegiatan konstruksi.

2.1.1.1 *Agile Software Development*

Menurut Pressman (2009), *Agile Software Development* adalah sekumpulan metodologi pengembangan perangkat lunak yang berbasis pada pengembangan iteratif, di mana persyaratan dan solusi berkembang melalui kolaborasi antar tim yang terorganisir. Istilah ini diciptakan pada tahun 2001 ketika *Agile Manifesto* dirumuskan.

Metode *Agile* umumnya mempromosikan disiplin proses manajemen proyek yang mendorong inspeksi dan adaptasi; filosofi kepemimpinan yang mendorong kerja sama dalam tim, pengorganisasian dan akuntabilitas; praktek rekayasa yang memungkinkan pengiriman perangkat lunak berkualitas tinggi dengan cepat; dan pendekatan bisnis yang sejalan dengan pengembangan kebutuhan pelanggan dan tujuan perusahaan.

2.1.1.2 *Extreme Programming* (XP)

Menurut Pressman (2009), *Extreme Programming* (XP) adalah metodologi pengembangan perangkat lunak yang ditujukan untuk meningkatkan kualitas perangkat lunak dan tanggap terhadap perubahan kebutuhan pelanggan. Jenis pengembangan perangkat lunak semacam ini dimaksudkan untuk meningkatkan produktivitas dan memperkenalkan pos pemeriksaan di mana persyaratan pelanggan baru dapat diadopsi.

Tahapan-tahapan dari *Extreme Programming* terdiri dari *planning* seperti memahami kriteria pengguna dan perencanaan pengembangan, *designing* seperti perancangan *prototype* dan tampilan, *coding* termasuk pengintegrasian, dan yang terakhir adalah *testing*.

Unsur-unsur lain dari *Extreme Programming* meliputi *paired programming* pada tahapan *coding*, *unit testing* pada semua kode, penghindaran pemrograman fitur kecuali benar-benar diperlukan, struktur manajemen yang datar, kode yang sederhana dan jelas, dan seringnya terjadi komunikasi antara *programmer* dan pelanggan ketika terjadi perubahan kebutuhan pelanggan seiring berlalunya waktu berlalu.

Metode ini membawa unsur-unsur yang menguntungkan dari praktek rekayasa perangkat lunak tradisional ke tingkat “ekstrem”, sehingga metode ini dinamai *Extreme Programming*. Unsur-unsur yang menjadi karakteristik metodologi adalah kesederhanaan, komunikasi, umpan balik, dan keberanian.

2.1.2 *Unified Modelling Language*

Menurut Object Management Group (2011), *Unified Modelling Language* (UML) adalah suatu bahasa pemodelan visual yang digunakan untuk menganalisa, merancang, dan mengimplementasikan sistem yang berbasis perangkat lunak dan pemodelan bisnis. UML merupakan pengembangan dari tiga metode berorientasi objek terkemuka seperti *Booch*, *Object Modelling Technique* (OMT), dan *Object-Oriented Software Engineering* (OOSE).

Pada UML versi 2.4, terdapat empat belas jenis diagram yang dikelompokkan kembali menjadi dua bagian, yaitu *structure diagram* yang berfokus pada struktur statis dari sistem dan *behavior diagram* yang berfokus pada struktur dinamis.

Comment [WSP1]: 11-08-06.pdf,page 710

Diagram-diagram yang dikategorikan ke dalam *structure diagram* di antaranya adalah:

1. *class diagram*,
2. *component diagram*,
3. *composite structure diagram*,
4. *deployment diagram*,
5. *object diagram*,
6. *package diagram*, dan
7. *profile diagram*.

Sedangkan diagram-diagram yang dikategorikan ke dalam *behavior diagram* di antaranya adalah:

1. *activity diagram*,
2. *communication diagram*,
3. *interaction overview diagram*,
4. *sequence diagram*,
5. *state machine diagram*,
6. *timing diagram*, dan
7. *use case diagram*.

2.1.2.1 *Use Case Diagram*

Menurut Satzinger, Jackson, & Burd (2010), *use case diagram* adalah diagram yang menampilkan berbagai peran pengguna dan bagaimana peran ini berfungsi dalam sebuah sistem.

Comment [WSP2]: 242

Use case diagram terdiri dari dua elemen pendukung dan satu pembatas yaitu:

Tabel 2.1 Penjelasan Simbol *Use Case Diagram*

Bentuk	Penjelasan
	*Actor* Berada di lingkungan luar
	*Use case* Berada di sistem internal
	*Boundaries* Pemisah lingkungan luar dan sistem internal

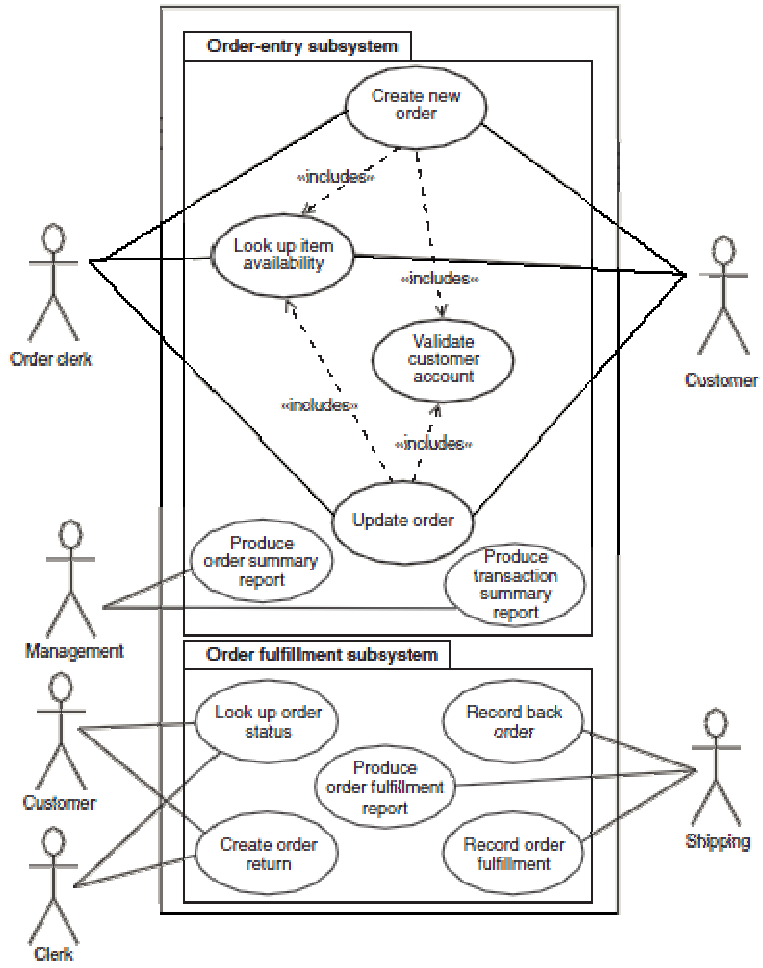
Package

Menurut Satzinger, Jackson, & Burd (2010), *package* adalah simbol yang digunakan untuk menandakan sekelompok elemen yang mirip. *Package* dilambangkan dengan kotak dengan *tab* di kiri atas yang berisi nama sub-sistem.

Inclusion

Menurut Satzinger, Jackson, & Burd (2010), *inclusion* adalah penanda jika sebuah *use case* menggunakan layanan dari sub-rutin umum, misalnya saat melakukan order baru perlu juga dilakukan validasi akun kustomer.

Inclusion dilambangkan dengan sebuah anak panah putus-putus disertai tulisan "<<includes>>".



Gambar 2.1 Contoh *Use Case Diagram*
(Sumber: Satzinger, et al., 2010)

2.1.2.2 *Use Case Descriptions*

Menurut Satzinger, Jackson, & Burd (2010), *use case descriptions* adalah sebuah penjelasan yang memuat rincian proses dari suatu *use case*.

Use Case Name:	*Create new order*	
Scenario:	Create new telephone order	
Triggering Event:	Customer telephones RMO to purchase items from the catalog.	
Brief Description:	When customer calls to order, the order clerk and system verify customer information, create a new order, add items to the order, verify payment, create the order transaction, and finalize the order.	
Actors:	Telephone sales clerk.	
Related Use Cases:	Includes: *Check item availability.*	
Stakeholders:	Sales department: to provide primary definition. Shipping department: to verify information content is adequate for fulfillment. Marketing department: to collect customer statistics for studies of buying patterns.	
Preconditions:	Customer must exist. Catalog, Products, and Inventory items must exist for requested items.	
Postconditions:	Order and order line items must be created. Order transaction must be created for the order payment. Inventory items must have the quantity on hand updated. The order must be related (associated) to a customer.	
Flow of Activities:	**Actor**	**System**
	1. Sales clerk answers telephone and connects to a customer.	
	2. Clerk verifies customer information.	2.1 Display customer information.
	3. Clerk initiates the creation of a new order.	3.1 Create a new order.
	4. Customer requests an item be added to the order.	
	5. Clerk verifies the item (*Check item availability* use case).	5.1 Display item information.
	6. Clerk adds item to the order.	6.1 Add an order item.
	7. Repeat steps 4, 5, and 6 until all items are added to the order.	
	8. Customer indicates end of order; clerk enters end of order.	8.1 Complete order. 8.2 Compute totals.
	9. Customer submits payment; clerk enters amount.	9.1 Verify payment. 9.2 Create order transaction. 9.3 Finalize order.
Exception Conditions:	2.1 If customer does not exist, then the clerk pauses this use case and invokes *Maintain customer information* use case. 2.2 If customer has a credit hold, then clerk transfers the customer to a customer service representative. 4.1 If an item is not in stock, then customer can a. choose not to purchase item, or b. request item be added as a back-ordered item. 9.1 If customer payment is rejected due to bad-credit verification, then a. order is canceled, or b. order is put on hold until check is received.	

Gambar 2.2 Contoh *Fully Developed Use Case Descriptions*
(Sumber: Satzinger, et al., 2010)

Tabel 2.2 Komponen *Use Case Descriptions*

No	Nama	Penjelasan
1	*Use case name*	nama *use case* harus mewakili tujuan *use case* yang sedang dicoba untuk diselesaikan. Nama harus dimulai dengan kata kerja
2	*Scenario*	penjelasan lebih spesifik dari *use case name*
3	*Trigerring event*	hal yang memicu terjadinya *scenario*
4	*Brief Description*	deskripsi ringkasan singkat yang terdiri dari beberapa kalimat yang menguraikan tujuan dari penggunaan *use case* dan kegiatannya
5	*Actors*	pelaku utama dalam *use case*
6	*Related use case*	*use case* lain yang berhubungan
7	*Stakeholders*	orang-orang yang memiliki kepentingan dalam pengembangan dan pengoperasian sistem perangkat lunak
8	*Preconditions*	kondisi yang harus dipenuhi sebelum *use case* dimulai
9	*Postconditions*	kondisi yang akan dicapai setelah menyelesaikan *use case*
10	*Flow of activities*	urutan kegiatan dalam menyelesaikan *use case*
11	*Exception conditions*	kondisi yang mungkin dapat membuat *use case* tidak dapat diselesaikan

2.1.2.3 *Activity Diagram*

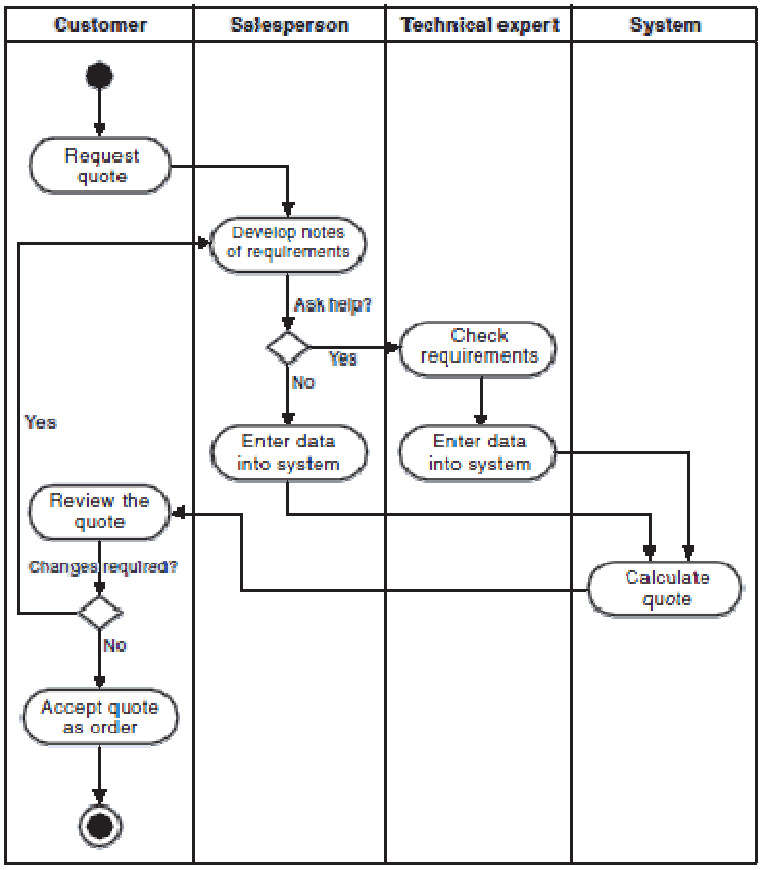
Menurut Satzinger, Jackson, & Burd (2010), *activity diagram* adalah diagram alur kerja yang menjabarkan aktivitas pengguna dan urutannya secara sekuensial.

Ada lima macam bentuk pada *activity diagram*, yaitu:

Tabel 2.3 Penjelasan Simbol *Activity Diagram*

Bentuk	Penjelasan
	*Activity* Tindakan yang dilakukan oleh aktor
	*Decision activity* Pengambilan keputusan berdasarkan syarat tertentu

—	*Synchronization bar* Penanda pemisahan atau penggabungan *activity*
●	*Starting activity pseudo* Keadaan sebelum aktivitas dilakukan
◉	*Ending activity pseudo* Keadaan setelah semua aktivitas dilakukan



Gambar 2.3 Contoh *Activity Diagram*
(Sumber: Satzinger, et al., 2010)

2.1.2.4 *Sequence Diagram*

Menurut Satzinger, Jackson, & Burd (2010), *sequence diagram* adalah diagram yang digunakan untuk menentukan kelas mana yang melakukan kolaborasi dan pesan apa yang masing-masing dari mereka harus kirimkan.

Comment [WSP3]: 435, 434

Sequence diagram terdiri dari beberapa komponen, yaitu:

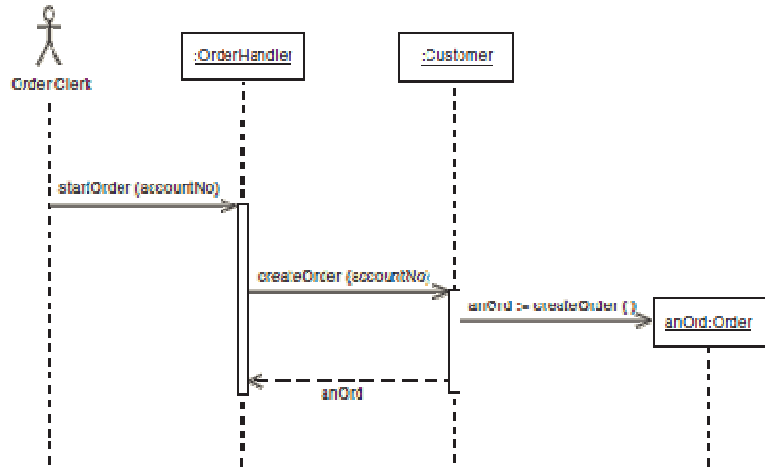
Tabel 2.4 Penjelasan Simbol *Sequence Diagram*

Bentuk	Penjelasan
	*Actor* Orang atau peran yang berinteraksi dengan sistem
	*Input message* Pesan masukan
	*Output message* Nilai yang dikembalikan
	*System object* Objek yang merepresentasikan keseluruhan sistem yang terotomatisasi
	Object lifeline Penunjuk urutan pesan dari atas hingga bawah

Comment [WSP4]: 187, 252

Comment [WSP5]: 190

Comment [WSP6]: 190



Gambar 2.4 Contoh *Sequence Diagram*
(Sumber: Satzinger, et al., 2010)

2.1.2.5 *Class Diagram*

Comment [WSP7]: 187

Menurut Satzinger, Jackson, & Burd (2010), *class diagram* adalah diagram yang menunjukkan kelas-kelas objek yang ada pada sistem. Dalam *class diagram*, kotak melambangkan kelas, dan garis yang menghubungkan kotak tersebut menunjukkan asosiasi antar kelas.

Visibility

Comment [WSP8]: 411,414

Menurut Satzinger, Jackson, & Burd (2010), *visibility* yang biasa dituliskan di depan setiap atribut maupun operasi adalah sebuah simbol untuk menandakan apakah objek lainnya dapat secara langsung mengakses atribut tersebut.

Ada dua simbol yang digunakan untuk menandakan *visibility*, yaitu:

Tabel 2.5 Penjelasan Simbol *Visibility*

Simbol	*Visibility*	Penjelasan
+	*Public*	Menandakan bahwa anggota dapat digunakan oleh semua kelas yang berhubungan (*visible*).
-	*Private*	Menandakan bahwa anggota hanya dapat digunakan oleh kelas yang mendefinisikan (*not visible*).

Multiplicity

Menurut Satzinger, Jackson, & Burd (2010), *multiplicity* adalah jumlah asosiasi yang dapat terjadi di antara hal-hal yang spesifik, misalnya seorang kustomer dapat memesan banyak order.

Comment [WSP9]: 180

Pada UML, dikenal beberapa macam simbol *multiplicity*, yaitu:

Comment [WSP10]: 188

Tabel 2.6 Penjelasan Simbol *Multiplicity*

Simbol	Penjelasan
0..1	Nol atau satu (opsional)
0..* atau *	Nol atau lebih (opsional)
1 atau 1..1	Tepat satu (wajib)
1..*	Satu atau lebih (wajib)

Relationship

Menurut Satzinger, Jackson, & Burd (2010), dalam *class diagram* dikenal adanya hubungan *association*. Pada *class diagram* dengan *whole part hierarchies* dikenal pula *aggregration* dan *composition*.

Comment [WSP11]: 190

Tabel 2.7 Penjelasan Simbol *Relationship*

Bentuk	Penjelasan
	*Association* Hubungan antara sebuah objek dengan objek lainnya
	*Aggregration* Hubungan antara suatu objek dengan bagian-bagiannya.
	Composition Hubungan antara suatu objek dengan bagian-bagiannya di mana masing-masing bagian tersebut tidak dapat berdiri sendiri tanpa objek induk.

Comment [WSP12]: 187

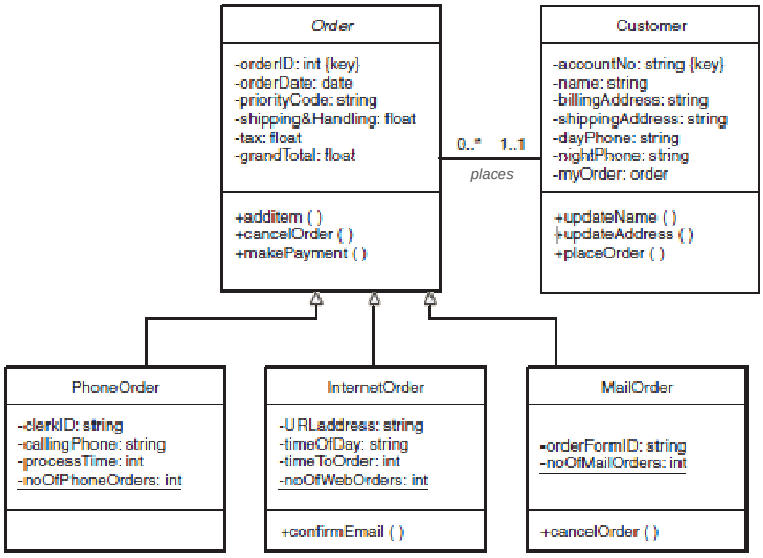
Comment [WSP13]: 190

Generalization

Comment [WSP14]: 189

Menurut Satzinger, Jackson, & Burd (2010), *generalization* adalah hierarki yang menyusun atau menentukan tingkatan kelas dari *superclass* yang lebih umum ke *subclass* yang lebih terspesialisasi, misalnya *superclass* Order memiliki *subclass* PhoneOrder, InternetOrder, dan MailOrder.

Untuk menunjukkan *generalization* dalam *class diagram*, sebuah segitiga yang mengarah ke *superclass*.



Gambar 2.5 Contoh *Class Diagram*
(Sumber: Satzinger, et al., 2010)

2.1.3 *Database*

Menurut Connolly & Begg (2010), *database* adalah repositori data yang dapat digunakan secara bersamaan oleh banyak departemen dan pengguna. Semua data terintegrasi dengan jumlah duplikasi yang minimum. *Database* biasanya tidak dimiliki oleh satu departemen atau pengguna, tetapi merupakan sumber daya bersama milik perusahaan.

Objek-objek yang ada dalam sebuah *database*:

- Tabel, yaitu objek yang berisi tipe-tipe data.
- Kolom, yaitu sebuah tabel berisi kolom untuk menampung data. Kolom mempunyai tipe dan nama yang unik.

- Tipe data, yaitu sebuah kolom mempunyai sebuah tipe data. Tipe data yang dapat dipilih misalnya *character, numeric*, dan sebagainya.
- *Primary key*, yaitu kata kunci utama yang menjamin data agar unik, hingga dapat dibedakan dari data yang lain.
- *Foreign key*, merupakan kolom-kolom yang mengacu pada *primary key* dari tabel yang lain. Dengan kata lain, *primary key* dan *foreign key* digunakan untuk menghubungkan sebuah tabel dengan tabel lain.

2.1.3.1 *Database Management System* (DBMS)

Menurut Connolly & Begg (2010), DBMS adalah *software* yang berinteraksi dengan pengguna, aplikasi *database*, dan *database*. DBMS memungkinkan pengguna untuk menambah, mengubah, menghapus, dan mengambil data dari *database*.

DBMS memiliki beberapa fasilitas, yaitu:

1. *Data storage, retrieval, and update*

   Fasilitas untuk fungsi dasar *database* seperti menyimpan, mengambil, dan mengubah data.

2. *A user-accessible catalog*

   Fasilitas katalog sistem yang terintegrasi untuk menyimpan data struktur, pengguna, dan aplikasi *database*.

3. *Transaction support*

   Fasilitas untuk membatalkan perubahan yang telah dibuat dan mengembalikannya ke keadaan konsisten sebelumnya.

4. *Concurrency control services*

Fasilitas yang memungkinkan banyak pengguna untuk mengakses data secara bersamaan.

5. *Recovery services*

   Fasilitas untuk mengembalikan data yang hilang akibat hal tak terduga seperti *system crash* atau bencana alam.

6. *Authorization services*

   Fasilitas untuk mengatur hak akses dari suatu data agar tidak semua *user* dapat melihat data tersebut.

7. *Support for data communication*

   Fasilitas untuk mengintegrasikan *database* dengan jaringan/perangkat lunak komunikasi.

8. *Integrity services*

   Fasilitas untuk menjaga integritas data.

9. *Services to promote data independence*

   Fasilitas untuk mengeluarkan data agar dapat berdiri sendiri.

10. *Utility services*

    Program utilitas membantu DBA untuk mengelola *database* secara efektif, contohnya fasilitas *import* dan *monitoring*

2.1.3.2 *Structured Query Language* (SQL)

Menurut Connolly & Begg (2010), *Structured Query Language* (SQL) adalah sebuah bahasa yang dirancang untuk mengubah tabel masukan menjadi tabel keluaran yang diharapkan. SQL telah distandarisasi oleh *International*

Standards Organization, membuatnya menjadi bahasa standar untuk mendefinisikan dan memanipulasi relasional *database*.

SQL memiliki dua komponen utama, yaitu:

1. *Data Definition Language* (DDL)

   Bahasa yang digunakan untuk mendefinisikan struktur *database* dan mengontrol akses ke data.

2. *Data Manipulation Language* (DML)

   Bahasa yang digunakan untuk mendapatkan dan memperbaharui data.

2.1.3.3 *SQLite*

Menurut Kreibich (2010), *SQLite* adalah paket perangkat lunak *public-domain* yang menyediakan sistem manajemen *database* relasional (RDBMS). Sistem ini digunakan untuk menyimpan baris-baris yang ditentukan pengguna ke dalam tabel besar. Selain penyimpanan dan manajemen data, dengan sistem ini kita juga dapat memproses *query* kompleks yang menggabungkan data dari beberapa tabel untuk menghasilkan laporan dan ringkasan data.

Kata "*Lite*" pada *SQLite* tidak mengacu pada kemampuannya. Sebaliknya, *SQLite* bersifat ringan dalam hal kompleksitas *set-up*, biaya administrasi, dan penggunaan sumber daya.

SQLite menyediakan lingkungan *database* relasional yang sangat fungsional dan fleksibel dengan tingkat kerumitan yang minimal bagi pengembang dan pengguna, serta konsumsi sumber daya yang minimal. Fitur-fitur yang ada pada *SQLite* adalah:

1. *Serverless*

SQLite tidak memerlukan *server* atau sistem yang terpisah untuk beroperasi. *SQLite* mengakses berkas penyimpanan secara langsung.

2. *Zero Configuration*

   Karena *SQLite* tidak memerlukan *server,* maka tidak ada pula *set-up*. Sebuah instance *database* dalam *SQLite* dapat dibuat semudah membuka berkas.

3. *Cross-Platform*

   Pada *SQLite*, seluruh *database* berada dalam berkas tunggal yang mendukung *cross-platform* dan tidak memerlukan administrasi.

4. *Self-Contained*

   SQLite memiliki *library* tunggal berisi seluruh sistem *database*, yang terintegrasi langsung ke aplikasi *host*.

5. *Small Runtime Footprint*

   SQLite memiliki kode yang kurang dari satu *megabyte*, dan hanya perlu beberapa *megabyte* memori untuk mengoperasikannya. Dengan sedikit penyesuaian, seperti pada ukuran dan penggunaan *library*, memori dapat berkurang secara signifikan.

6. *Transactional*

   Transaksi *SQLite* sepenuhnya *ACID-compliant*, memungkinkan akses aman dari beberapa proses atau *threads*.

7. *Full-Feature*

   SQLite mendukung sebagian besar fitur bahasa *query* yang ada pada standar SQL92 (SQL2).

8. *Highly Reliable*

Kode *SQLite* telah diuji dan diverifikasi oleh para pengembangnya dengan sangat serius.

2.1.4 Interaksi Manusia Komputer

2.1.4.1 Pengertian Interaksi Manusia Komputer

Menurut Shneiderman & Plaisant (2010), interaksi manusia komputer adalah sebuah ilmu yang mempelajari bagaimana manusia berinteraksi dengan komputer dan pengaruh dari interaksi antara manusia dengan komputer. Fokus dari interaksi manusia komputer adalah perancangan dan evaluasi antarmuka pengguna.

2.1.4.2 Lima Faktor Manusia Terukur

Menurut Shneiderman & Plaisant (2010), ada lima faktor manusia terukur yaitu:

1. Waktu belajar

   Waktu yang dibutuhkan seseorang untuk belajar.

2. Kecepatan kinerja

   Kecepatan untuk menyelesaikan pekerjaan.

3. Akurasi

   Berapa banyak kesalahan yang mungkin dilakukan oleh pengguna.

4. Daya ingat

   Kemampuan pengguna untuk mengingat atau mempertahankan pengetahuan dalam jangka waktu tertentu.

5. Kepuasan

   Kepuasan subjektif pengguna terhadap *interface*.

2.1.4.3 Delapan Aturan Emas

Menurut Shneiderman & Plaisant (2010), ada delapan aturan yang dapat digunakan sebagai petunjuk dasar yang baik untuk merancang suatu *user interface*. Delapan aturan yang disebut dengan *Eight Golden Rules of Interface Design* ini terdiri dari:

1. Konsistensi

   Konsistensi dilakukan pada urutan tindakan, perintah, dan istilah yang digunakan pada *prompt*, menu, serta layar bantuan.

2. Melayani untuk semua orang

   Pengguna yang beragam dan desain harus mempertimbangkan perbedaan dan pertimbangan dalam hal usia, cacat dan keanekaragaman teknologi.

3. Memberikan umpan balik yang informatif

   Untuk setiap tindakan operator, sebaiknya disertakan suatu sistem umpan balik. Untuk tindakan yang sering dilakukan dan tidak terlalu penting, dapat diberikan umpan balik yang sederhana. Tetapi ketika tindakan merupakan hal yang penting, maka umpan balik sebaiknya lebih substansial. Misalnya muncul suatu suara ketika salah menekan tombol pada waktu memasukkan data atau muncul pesan kesalahannya.

4. Merancang dialog untuk menghasilkan suatu penutupan

   Urutan tindakan sebaiknya diorganisir dalam suatu kelompok dengan bagian awal, tengah, dan akhir. Umpan balik yang informatif akan memberikan indikasi bahwa cara yang dilakukan sudah benar dan dapat mempersiapkan kelompok tindakan berikutnya.

5. Memberikan penanganan kesalahan yang sederhana

   Sedapat mungkin sistem dirancang sehingga pengguna tidak dapat melakukan kesalahan fatal. Jika kesalahan terjadi, sistem dapat mendeteksi kesalahan dengan cepat dan memberikan mekanisme yang sederhana dan mudah dipahami untuk penanganan kesalahan.

6. Mudah kembali ke tindakan sebelumnya

   Hal ini dapat mengurangi kekhawatiran pengguna karena pengguna mengetahui kesalahan yang dilakukan dapat dibatalkan; sehingga pengguna tidak takut untuk mengeksplorasi pilihan-pilihan lain yang belum biasa digunakan.

7. Mendukung tempat pengendali internal (*internal locus of control*)

   Pengguna dapat menjadi pengendali sistem dan sistem akan merespons tindakan yang dilakukan pengguna, bukan pengguna merasa bahwa sistem mengendalikan pengguna. Sebaiknya sistem dirancang sedemikian rupa sehingga pengguna menjadi inisiator daripada responden.

8. Mengurangi beban ingatan jangka pendek

   Keterbatasan ingatan manusia membutuhkan tampilan yang sederhana atau banyak tampilan halaman yang sebaiknya disatukan, serta diberikan cukup waktu pelatihan untuk kode, *mnemonic*, dan urutan tindakan.

2.1.5 Teknologi *Mobile*

2.1.5.1 *Mobile Phone*

Menurut Ballard (2007), karakteristik dari pengguna *mobile phone* adalah seseorang yang tidak ingin diganggu atau ingin mempunyai *privacy* sendiri, suka

bersosialisasi, yang mendorong orang untuk menciptakan sebuah desain *mobile phone* baru yang dapat mencakup kebutuhan.

Banyaknya aktivitas atau pekerjaan yang perlu dilakukan dan sedikitnya waktu yang akan tersedia membuat orang-orang ingin tetap berhubungan dengan informasi di mana pun dan kapan pun. teknologi *mobile phone* merupakan inovasi yang muncul sebagai jawaban atas kebutuhan-kebutuhan tersebut. Semakin berkembangnya teknologi *mobile phone*, maka muncul istilah *smartphone*.

2.2 *Smartphone*

Menurut Ciaramitaro (2012), peningkatan utama dalam sistem operasi *mobile* datang dengan pengenalan pada *smartphone* yang di mana didukung oleh sistem operasi *mobile* berfitur lengkap yang memungkinkan kemampuan komputasi yang lebih canggih, *web browsing*, dan instalasi dari bermacam-macam aplikasi dan *game*. Sebagian besar sistem operasi *mobile* terkemuka sekarang mendukung fungsi tambahan yang disediakan oleh *smartphone*.

Fitur-fitur yang terdapat di *smartphone* seperti teknologi *touchscreen*, portable media player, *global positioning system* (GPS), *keyboard* QWERTY dan *wireless fidelity* (Wi-Fi) merupakan fitur yang membedakan antara *smartphone* dengan *mobile phone*.

2.2.1 *Global Positioning System* (GPS)

Menurut Elangovan (2006), *Global Positioning System* adalah sistem yang digunakan untuk mengukur lintang, bujur, dan ketinggian. GPS memiliki tiga segmen, segmen luar angkasa, segmen pengguna, dan segmen kendali. Segmen

luar angka terdiri dari 28 satelit yang mengorbit Bumi di ketinggian 20.200 km. Segmen pengguna terdiri dari alat penerima GPS. Segmen kendali terdiri dari 5 stasiun yang terletak di berbagai penjuru dunia yang memantau fungsionalitas dari GPS.

GPS pertama kali dikembangkan oleh Departemen Pertahanan Amerika Serikat pada Februari 1978. GPS yang boleh digunakan penduduk sipil adalah *Standard Positioning Service* (SPS), sedangkan *Precise Positioning Service* (PPS) hanya boleh digunakan oleh agen pemerintahan.

2.3 Teori Khusus

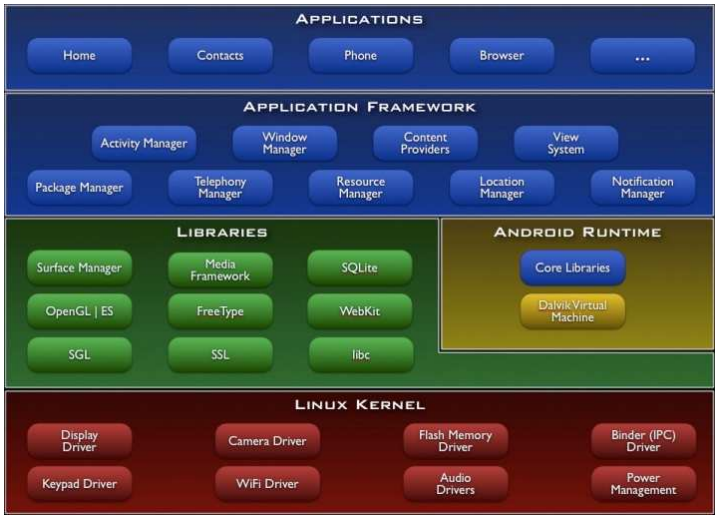
2.3.1 *Android*

Menurut Ableson, Collins, & Sen (2008), *Android* merupakan *platform open-source* pertama untuk *mobile device* yang memiliki potensi untuk membuat terobosan signifikan di banyak pasar.

Android adalah produk dari *Open Handset Alliance*, sebuah aliansi yang terdiri dari sekitar 30 organisasi yang berkomitmen untuk membawa *mobile phone* yang "*better and open*" ke pasar.

Android mencakup sistem operasi berbasis kernel *Linux*, *user interface*, aplikasi *end-user*, *code libraries*, *application frameworks*, dukungan multimedia dan fungsi telepon. Komponen dasar pada sistem ini ditulis dalam C atau C++, sedangkan aplikasi pengguna dan *built-in* ditulis dalam bahasa *Java* dengan menggunakan *Android Software Development Kit* (SDK).

Sistem operasi *Android* memiliki beberapa komponen-komponen utama yang terbagi menjadi beberapa *layer* seperti pada gambar di bawah.



Gambar 2.6 Arsitektur *Android*
(Sumber: http://developer.*Android*.com/images/system-architecture.jpg, 2012)

Secara sederhana, arsitektur *Android* merupakan sebuah *Linux kernel* dan sekumpulan *library* C/C++ dalam suatu *framework* yang menyediakan dan mengatur alur proses aplikasi.

2.3.1.1 *Linux Kernel*

Pada *Linux kernel* versi 2.6 yang digunakan *Google* untuk membangun sistem *Android*, terdapat *memory management*, *security setting*, *power management*, dan beberapa *driver hardware*. *Kernel* berperan sebagai *abstraction layer* antara *hardware* dan keseluruhan *software*.

Walaupun *Android* dibangun di atas *Linux kernel* 2.6, namun secara keseluruhan *Android* bukanlah *Linux*, karena dalam *Android* tidak terdapat paket standar yang dimiliki oleh *Linux* lainnya.

Pada *Android* hanya terdapat beberapa layanan (*memory management*, *security setting*, *power management*, dan beberapa *driver hardware*), sedangkan *Linux kernel* menyediakan *driver* layar, kamera, *keypad*, WiFi, *flash memory*, audio, dan *interprocess communication* (IPC) untuk mengatur aplikasi dan lubang keamanan.

2.3.1.2 *Android Runtime*

Android Runtime terdiri dari *Core Libraries* dan *Dalvik Virtual Machine*. *Core Libraries* mencakup serangkaian *core library Java*, satu set *library* dasar yang menyediakan sebagian besar fungsi yang ada pada dasar bahasa pemrograman *Java*.

Dalvik adalah sebuah *Java Virtual Machine* yang telah dioptimalkan untuk telepon seluler yang menjadi kekuatan pada sistem *Android*. *Dalvik Virtual Machine* menggunakan *Linux kernel* untuk menjalankan fungsi-fungsi seperti *threading* dan *low-level memory management*.

Setiap aplikasi yang berjalan pada *Android* berjalan pada prosesnya sendiri, dengan *instance* dari *Dalvik Virtual Machine*. *Dalvik* telah dibuat sehingga sebuah peranti yang memakainya dapat menjalankan *multi virtual machine* dengan efisien.

2.3.1.3 *Libraries*

Pada *layer* ini *Android* menyertakan sekumpulan *core library C/C++* yang digunakan oleh berbagai komponen pada sistem *Android*.

Library bukanlah aplikasi yang berjalan sendiri, namun hanya dapat digunakan oleh program yang berada di level atasnya, sehingga dapat diakses oleh

programmer melalui *Android application framework*. Sejak versi *Android* 1.5, pengembang dapat membuat dan menggunakan *library* sendiri menggunakan *Native Development Toolkit* (NDK).

Beberapa contoh *core library* tersebut adalah:

1. *System C library.* implementasi *standard C system library* milik BSD yang dioptimasi untuk peranti *embedded* berbasis *Linux.*
2. *Media library, library* dari *PacketVideo's OpenCORE* untuk memutar dan merekam media audio dan video.
3. *Surface manager* untuk menyediakan akses tampilan 2D dan 3D dari berbagai aplikasi.
4. *LibWebCore*, sebuah *web browser engine* modern yang mendukung *Android browser* maupun *embeddable web view.*
5. *Graphics library* termasuk *Scene Graph Library* (SGL) untuk kemampuan menampilkan grafik dua dimensi dan *Open Graphics Library* (*OpenGL*) untuk kemampuan menampilkan gambar tiga dimensi dengan *hardware acceleration* dan *software rasterizer*.
6. *SQLite* untuk dukungan dalam hal *database*.

2.3.1.4 *Application Framework*

Application framework merupakan serangkaian *tool* dasar seperti alokasi *resource*, aplikasi telepon, pergantian antarproses atau program, dan pelacakan lokasi fisik telepon.

Para pengembang aplikasi memiliki hak penuh untuk mengakses dan memanfaatkan *tool* pada *layer* ini, serta *Android Protocol Interface* (API) untuk menciptakan aplikasi yang lebih kompleks.

Arsitektur ini dirancang untuk menyederhanakan pemakaian kembali komponen-komponen, setiap aplikasi dapat menunjukkan kemampuannya dan aplikasi lain dapat memakai kemampuan tersebut. Mekanisme yang sama memungkinkan pengguna mengganti komponen-komponen yang dikehendaki.

2.3.1.5 *Applications*

Layer ini merupakan tempat di mana fungsi-fungsi dasar *smartphone* seperti menelepon dan mengirim pesan singkat, menjalankan *web browser*, mengakses daftar kontak, dan lain-lain berada.

Layer ini adalah *layer* yang paling sering diakses oleh sebagian besar pengguna melalui *user interface* yang telah disediakan.

2.3.2 *Java*

Menurut Liang (2011), *Java* adalah bahasa pemrograman serbaguna yang dapat digunakan untuk mengembangkan aplikasi pada *desktop* dan *server*. *Java* dapat digunakan untuk pemrograman web, maupun aplikasi mandiri di seluruh platform pada *server*, *desktop*, dan perangkat *mobile*.

Acuan pelaksanaan *compiler*, *virtual machine*, dan *class libraries* pada *Java* dikembangkan oleh Sun pada tahun 1995. Dengan teknik pelaksanaan ini, terdapat *bytecode* (*file class*) pada setiap aplikasi *Java* yang dapat berjalan pada

Java Virtual Machine (JVM) terlepas dari arsitektur komputer. Hal ini dimaksudkan agar pengembang aplikasi dapat "*write once, run anywhere*".

2.3.3 *Eclipse*

Menurut Liang (2011), *Eclipse* adalah sebuah *Integrated Development Environment* (IDE) untuk mengembangkan program *Java* dengan cepat, di mana fungsi-fungsi seperti *editing, compiling, building, executing*, dan *debugging* sudah diintegrasikan ke dalam satu antarmuka pengguna grafis sehingga dapat meningkatkan produktivitas pemrograman.

2.3.4 *OpenCV*

Menurut Bradski & Kaehler (2008), *OpenCV* merupakan *library computer vision open source* yang tersedia di http://sourceforge.net/projects/opencvlibrary. *Library* ini ditulis dalam bahasa *C* dan *C++* dan dapat berjalan di *Linux,* Windows dan *Mac OS X*. Ada juga pengembangan aktif antarmuka untuk *Python, Ruby, Matlab,* dan bahasa lainnya.

OpenCV sudah sangat familier dengan *image processing* pada *computer vision*. *Computer vision* sendiri adalah salah satu cabang dari bidang ilmu *image processing* yang memungkinkan komputer dapat melihat seperti manusia. Dengan dukungan ini, komputer dapat mengambil keputusan, melakukan aksi, dan mengenali suatu objek.

OpenCV didesain untuk aplikasi *real-time* dan memiliki fungsi-fungsi akuisisi yang baik untuk gambar maupun video. *OpenCV* terdiri dari 5 *library*, yaitu:

1. CV untuk algoritma *image processing* dan *vision*;
2. ML untuk *machine learning library*;
3. HighGUI untuk I/O GUI, gambar dan video;
4. CXCORE untuk struktur data, dukungan XML dan fungsi-fungsi grafis; serta
5. CvAux untuk fungsi tambahan (eksperimental).

OpenCV didesain untuk efisiensi komputasi dan dengan fokus yang kuat pada aplikasi *realtime*. *OpenCV* ditulis dalam *optimized C* dan dapat mengambil keuntungan dari prosesor *multicore*.

2.3.5 *E-Tourism*

Menurut Buhalis & Jun (2011), kemajuan teknologi yang pesat dan perkembangan pariwisata yang dinamis telah terjadi bergandengan tangan selama bertahun-tahun. Sejak tahun 1980-an, teknologi informasi komunikasi (TIK) telah mengubah pariwisata global, menciptakan aplikasi dan solusi yang sering disebut *e-tourism*.

E-tourism memungkinkan calon wisatawan mengakses informasi lebih banyak, dapat dipercaya, dan akurat yang disediakan oleh organisasi pariwisata, perusahaan swasta dan pengguna lain/konsumen. Semakin baru, berpengalaman, dan canggih menuntut wisatawan memerlukan interaksi dengan penyelenggara wisata untuk memenuhi kebutuhan spesifik mereka sendiri dan keinginan. *E-tourism* memberdayakan wisatawan untuk melakukan pemesanan di sebagian kecil dari biaya, waktu dan ketidaknyamanan yang dibutuhkan oleh metode konvensional.

2.3.6 *Image Processing*

Menurut Bradski & Kaehler (2008), *image processing* adalah penggunaan operator dengan tingkat yang lebih tinggi dibanding dengan yang terdefinisi pada struktur gambar untuk menyelesaikan tugas-tugas yang maknanya secara alami dalam konten grafis.

Beberapa contoh *image processing*:

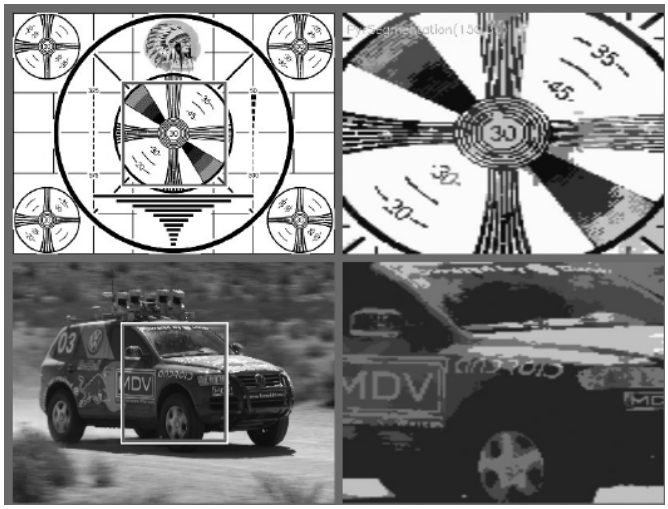
- *Smoothing/Blurring*

  Smoothing/Blurring biasanya digunakan untuk mengurangi *noise* atau *camera artifacts*. *Smoothing* juga penting ketika kita ingin mengurangi resolusi suatu gambar.

Gambar 2.7 Contoh *Smoothing/Blurring*
(Sumber: Gary Bradski dan Adrian Kaehler, 2008)

- *Resize*

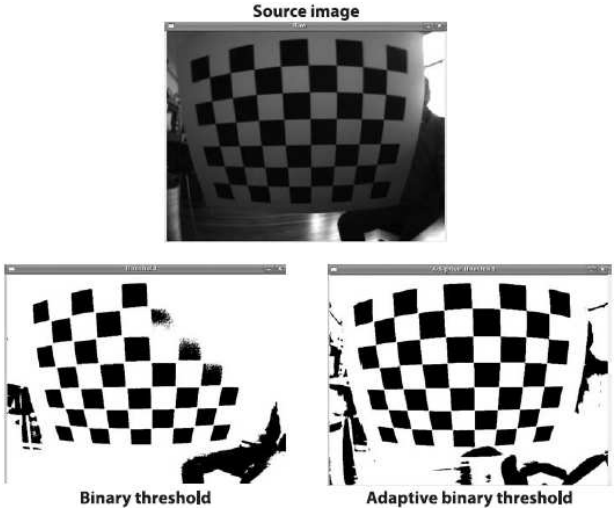
  Resize adalah mengubah ukuran suatu gambar menjadi lebih besar atau kecil.



Gambar 2.8 Contoh *Resize*
(Sumber: Gary Bradski dan Adrian Kaehler, 2008)

- *Threshold*

  Threshold digunakan untuk menghilangkan piksel yang berada di atas atau di bawah nilai tertentu dan menampilkan sisanya.



Gambar 2.9 Contoh *Threshold*
(Sumber: Gary Bradski dan Adrian Kaehler, 2008)

2.3.7 *Image Recognition*

Menurut Kozlov (2007), gambar adalah suatu sekumpulan titik terbatas tidak kosong di bidang datar. Alasan untuk definisi ini adalah bahwa setiap gambar *grayscale* dapat didekati dengan kumpulan titik dengan kepadatan yang berbeda di berbagai bagian dari gambar untuk menggambarkan tingkat abu-abu dengan tingkat keakuratan sesuai kebutuhan.

Pendekatan ini tidak menghalangi analisis gambar warna karena ini dapat direpresentasikan sebagai tiga *grayscale*. Dengan menerima definisi formal dari gambar sebagai himpunan berhingga titik di bidang datar, menjadi mungkin untuk secara seksama membuktikan semua kesimpulan berikutnya. Kesamaan antara dua gambar akan diestimasi dengan menyelaraskan mereka sehingga titik-titik yang sesuai dari gambar ini dibawa ke posisi yang mungkin paling dekat.

2.3.7.1 *Haar Classifier*

Menurut Seo (2012), *Haar classifier* adalah metode *object detection* yang telah disediakan oleh *OpenCV* dalam membangun sebuah *boosted rejection cascade* yang akan membuang data *training* negatif sehingga didapat suatu keputusan untuk menentukan data positif.

Haar classifier merupakan metode *supervised learning*, yaitu membutuhkan data *training* untuk dapat mendeteksi objek-objek tertentu. Untuk itu, *Haar classifier* membutuhkan data positif (objek yang akan dideteksi) dan data negatif (bukan objek yang akan dideteksi).