BAB II
LANDASAN TEORI

2.1 Kartu Nama

Kartu nama adalah sebuah kartu yang menyampaikan informasi tentang sebuah perusahaan ataupun individu yang disampaikan hanya sebagai pengingat dalam sebuah perkenalan formal. Sebuah Kartu nama biasanya berisi tentang nama perusahaan termasuk logo perusahaan, alamat pos, nomer telepon, nomor fax dan *email*. Desain kartu nama harus sesuai dengan karakter usaha, dalam hal ini perorangan maupun perusahaan sehingga bisa meningkatkan nilai kualitas pada diri seseorang. Kartu nama harus menunjukkan siapa pemiliknya dan memberikan informasi mengenai segala hal yang bisa dilakukan oleh pemilik dan juga layanan yang bisa diberikan di masa mendatang.

Fungsi dari kartu nama adalah:

1. Sebagai kenang-kenangan data pribadi untuk teman-teman, relasi perusahaan dan keluarga.
2. Sebagai sarana promosi usaha untuk pemilik usaha.
3. Memberikan informasi berupa nama, telepon, *email*, dan *website* tentang produk usaha yang ditawarkan.
4. Sebagai tanda keseriusan dan alat ukur keprofesionalan dalam bisnis.

Kartu nama bagi seorang pebisnis, baik itu owner maupun karyawan terutama marketing merupakan sarana paling tepat, efektif dan efisien untuk merangkai dan memuat berbagai informasi tentang bisnis atau usaha di dalam satu lembar kertas kecil berupa kartu nama, untuk itu hampir seluruh pebisnis yang profesional pasti memiliki kartu nama. Bahkan biasanya ada diantaranya yang memiliki beberapa jenis desain kartu nama yang berdasarkan fungsinya. Kartu nama juga tidak hanya digunakan oleh kalangan pebisnis tapi juga bagi semua orang bahkan ibu rumah tangga pun membutuhkannya.

2.2 *Android*

Android adalah sistem operasi yang berbasis *Linux* untuk telepon seluler seperti telepon pintar (*Smartphone*) dan komputer tablet. *Android* menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri untuk digunakan oleh bermacam peranti begerak. Awalnya, *Google Inc.* membeli *Android Inc.*, pendatang baru yang membuat peranti lunak untuk ponsel. Kemudian untuk mengembangkan *Android*, dibentuklah *Open Handset Alliance*, konsorsium dari 34 perusahaan piranti keras, piranti lunak dan telekomunikasi, termasuk *Google, HTC, Intel, Motorola, Qualcomm, T-Mobile* dan *Nvidia.*

Beberapa pengertian lain dari *Android*, yaitu :

1. Merupakan *platform* terbuka (*Open Source*) bagi para programmer untuk membuat atau mengembangkan aplikasi.
2. Merupakan sistem operasi yang dibeli *Google Inc* dari *Android Inc*.

Bukan bahasa pemrograman, akan tetapi hanya menyediakan lingkungan hidup atau *runtime environment* yang disebut DVM (*Dalvik Virtual Machine*) yang telah dioptimasi untuk *device* dengan sistem memori yang kecil.

Sudah banyak *platform* untuk perangkat selular saat ini, termasuk didalamnya *Symbian*, *iPhone*, *Windows Mobile*, *BlackBerry*, *Java Mobile Edition*, *Linux Mobile* (LiM), dan banyak lagi. Namun ada beberapa hal yang menjadi kelebihan *Android*. Walaupun beberapa fitur-fitur yang ada telah muncul sebelumnya pada *platform* lain, *Android* adalah yang pertama menggabungkan hal seperti berikut [6]:

1. Keterbukaan, Bebas pengembangan tanpa dikenakan biaya terhadap sistem karena berbasiskan *Linux* dan *open source*. Pembuat perangkat menyukai hal ini karena dapat membangun *platform* yang sesuai yang diinginkan tanpa harus membayar *royality*. Sementara pengembang *software* menyukai karena *Android* dapat digunakan diperangkat manapun dan tanpa terikat oleh *vendor* manapun.
2. Arsitektur komponen dasar *Android* terinspirasi dari teknologi *internet Mashup*. Bagian dalam sebuah aplikasi dapat digunakan oleh aplikasi lainnya,

bahkan dapat diganti dengan komponen lain yang sesuai dengan aplikasi yang dikembangkan.

3. Banyak dukungan *service*, kemudahan dalam menggunakan berbagai macam layanan pada aplikasi seperti penggunaan layanan pencarian lokasi, *database* SQL, *browser* dan penggunaan peta. Semua itu sudah tertanam pada *Android* sehingga memudahkan dalam pengembangan aplikasi.
4. Siklus hidup aplikasi diatur secara otomatis, setiap program terjaga antara satu sama lain oleh berbagai lapisan keamanan, sehingga kerja sistem menjadi lebih stabil. Pengguna tak perlu khawatir dalam menggunakan aplikasi pada perangkat yang memorinya terbatas.
5. Dukungan grafis dan suara terbaik, dengan adanya dukungan 2D grafis dan animasi yang diilhami oleh Flash menyatu dalam 3D menggunakan *OpenGL* memungkinkan membuat aplikasi maupun *game* yang berbeda.

Portabilitas aplikasi, aplikasi dapat digunakan pada perangkat yang ada saat ini maupun yang akan datang. Semua program ditulis dengan menggunakan bahas pemrograman Java dan dieksekusi oleh mesin *virtual Dalvik*, sehingga kode program *portable* antara ARM, X86, dan arsitektur lainnya. Sama halnya dengan dukungan masukan seperti penggunaan *Keyboard*, layar sentuh, *trackball* dan resolusi lyr semua dapat disesuaikan dengan program [6].

2.2.1 Sejarah *Android*

Pada Juli 2000, *Google* bekerjasama dengan *Android Inc*., perusahaan yang berada di Palo Alto, California, Amerika Serikat. Para pendiri *Android Inc*. bekerja pada *Google*, diantaranya Andy Rubin, Rich Miner, Nick Sears dan Chris White. Saat itu banyak yang menganggap fungsi *Android Inc.* hanyalah sebagai perangkat lunak pada telepon seluler. Sejak saat itu muncul rumor bahwa *Google* hendak memasuki pasar telepon seluler. Di perusahaan *Google*, tim yang dipimpin Rubin bertugas mengembangkan program perangkat seluler yang didukung oleh *Kernel Linux*. Hal ini menunjukan indikasi bahwa *Google* sedang bersiap menghadapi persaingan dalam pasara telepon seluler. Versi *android* terbaru yaitu

versi 5.0. (*KitKat*). *Android* juga sudah bergabung dengan beberapa *smart mobile* seperti *LG, Samsung, Sony Ericsson, HTC, Motorola* dan lainnya.

Sekitar September 2007 sebuah studi melaporkan bahwa *Google* mengajukan hak paten aplikasi telepon seluler (akhirnya *Google* mengenalkan *Nexus One,* salah satu jenis telepon pintar GSM yang menggunakan *Android* pada sistem operasinya. Telepon seluler ini diproduksi oleh *HTC Corporotaion* dan tersedia di pasaran pada 5 Januari 2010).

Pada Desember 2008, diumumkan anggota baru yang bergabung dalam program kerja *Android ARM Holdings, Atheros Communications,* diproduksi oleh *Asustek Computer Inc. Garmin Ltd, Softbank, Sony Ericsson Toshiba Corp* dan *Vodafone Group Plc.* Seiring pembentukan *Open Handset Alliance,* OHA mengumumkan produk perdana mereka, *Android,* perangkat bergerak (*Mobile*) yang merupakan modifikasi *kernel Linux* 2.6. sejak *Android* dirilis telah dilakukan berbagai pembaruan berupa perbaikan *bug* dan penmbahan fitur baru [6].

2.2.2 Versi *Android*

Banyak *smartphone* dan PC Tablet menggunakan sistem operasi dengan versi yang berbeda. Semakin tinggi versi, fiturnya semakin canggih dan banyak. Telepon pertama yang memakai sistem operasi *Android* adalah HTC Dream yang dirilis pada tanggal 22 Oktober 2008. Beberapa uraian versi *android* seperti dibawah ini :

1. *Android* versi 1.1

Pada 9 Maret 2009, *Google* merilis *Android* versi 1.1. *Android* versi ini dilengkapi dengan pembaruan estetis pada aplikasi, jam alarm, *voice search* (pencarian suara), pengiriman pesan dengan Gmail, dan pemberitahuan *Email*.

2. *Android* versi 1.5 (*Cupcake*)

Pada pertengahan Mei 2009, *Google* kembali merilis telepon seluler dengan menggunakan *Android* dan SDK (*Software Development Kit*) dengan versi 1.5 (*Cupcake*). Terdapat beberapa pembaruan termasuk juga penambahan beberapa fitur dalam seluler versi ini yakni kemampuan merekam dan menonton video dengan modus kamera, mengunggah video ke *Youtube* dan gambar ke Picasa

langsung dari telepon, dukungan *Bluetooth* A2DP, kemampuan terhubung secara otomatis ke *headset bluetooth*, animasi layar dan *keyboard* pada layar yang dapat disesuaikan dengan sistem.

3. *Android* versi 1.6 (*Donut*)

Donut (versi 1.6) dirilis pada September dengan menampilkan proses pencarian yang lebih baik dibanding sebelumnya, penggunaan baterai indikator dan kontrol *applet* VPN. Fitur lainnya adalah galeri yang memungkinkan pengguna untuk memilih foto yang akan dihapus pada kamera, *camcorder* dan galeri yang diintegrasikan pada CDMA / EVDO, 802.1x, VPN, *Gestures*, dan *Text-to-spech engine*. Kemampuan dial kontak teknologi *text to change speech* (tidak tersedia pada semua ponsel).

4. *Android* versi 2.0/2.1 (*Eclair*)

Pada 3 Desember 2009 kembali diluncurkan ponsel *Android* dengan versi 2.0/2.1 (*Eclair*), perubahan yang dilakukan adalah mengoptimalkan *hardware*, peningkatan Google Maps 3.1.2, perubahan UI dengan browser baru dan dukungan HTML 5, daftar kontak yang baru, dukungan *flash* untuk kamera 3.2 MP, *digital zoom*, dan *Bluetooth* 2.1. Untuk bergerak cepat dalam persaingan perangkat generasi berikut, Google melakukan investasi dengan mengadakan kompetisi aplikasi *mobile* terbaik (*killer apps*-aplikasi unggulan). Kompetisi ini berhadiah $25,000 bagi setiap pengembang aplikasi terpilih. Kompetisi diadakan selama dua tahap yang tiap tahapnya dipilih 50 aplikasi terbaik. Dengan semakin berkembangnya dan semakin bertambahnya jumlah *handset Android*, semakin banyak pihak ketiga yang berminat untuk menyalurkan aplikasi mereka kepada sistem operasi *Android*. Aplikasi terkenal yang diubah ke dalam sistem operasi *Android* adalah Shazam, *Backgrounds* dan *WeatherBug*. Sistem operasi *Android* dalam situs internet juga dianggap penting untuk menciptakan aplikasi *Android* asli, contohnya oleh *MySpace* dan *Facebook.*

5. *Android* versi 2.2 (*Froyo* : *Frozen Yoghurt*)

Pada 20 Mei 2010, *Android* versi 2.2 (Froyo) diluncurkan. Perubahan-perubahan umumnya terhadap versi-versi sebelumnya antara lain dukungan Adobe Flash 10.1, kecepatan kinerja dan aplikasi 2 sampai 5 kali lebih cepat, integrasi

dengan V8 JavaScript *engine* yang dipakai Google Chrome yang mempercepat kemampuan rendering pada browser, pemasangan aplikasi dalam SD Card, kemampuan WiFi Hotspot portabel dan kemampuan *auto update* dalam aplikasi *Android* Market.

6. *Android* versi 2.3 (*Gingerbread*)

Pada 6 Desember 2010, *Android* versi 2.3 (*Gingerbread*) diluncurkan. Perubahan-perubahan umum yang didapat dari versi ini antara lain peningkatan kemampuan permainan (*gaming*), peningkatan fungsi *copy paste*, layar antarmuka (*User Interface*) didesain ulang, dukungan format video VP8 dan WebM. efek audio baru (*reverb, equalization, headphone virtualization,* dan *bass boost*), dukungan kemampuan *Near Field Communication* (NFC) dan dukungan jumlah kamera yang lebih dari satu.

7. *Android* versi 3.1 (*Honeycomb*)

Android Honeycomb dirancang khusus untuk tablet. *Android* versi ini mendukung ukuran layar yang lebih besar. *User Interface* pada *Honeycomb* juga berbeda karena sudah didesain untuk tablet. *Honeycomb* juga mendukung multi prosesor dan juga akselerasi perangkat keras (*hardware*) untuk grafis. Tablet peratam yang dibuat dengan menjalankan *Honeycomb* adalah Motorola Xoom.

8. *Android* versi 4.0 (*Ice Cream Sandwich*)

Diumumkan pada tanggal 19 Oktober 2011, membawa fitur *Honeycomb* untuk *smartphone* dan menambahkan fitur baru termasuk membuka kunci dengan pengenalan wajah, jaringan data pemantauan penggunaan dan kontrol, terpadu kontak jaringan sosial, perangkat tambahan fotografi, mencari *email* secara *offline* dan berbagi informasi dengan menggunakan NFC.

9. *Android* versi 4.1 (*Jelly Bean*)

Pada tanggal 27 Juni 2012 Google mengumumkan *Android* versi 4.1 (*Jelly Bean*) dalam konferensi Google I/O berdasarkan kernel Linux 3.0.31. *Jelly Bean* adalah pembaruan penting yang bertujuan untuk meningkatkan fungsi dan kinerja antarmuak pengguna (*User Interface*). Pembaruan ini diwujudkan dalam proyek *Butter*, perbaikan ini termasuk antisipasi sentuh, *triple buffering*, perpanjangan

waktu vsync dan peningkatan *frame rate* hingga 60fps untuk menciptakan UI yang lebih halus. Perangkat pertama yang menggunakan sistem operasi ini adalah tablet Nexus 7, yang dirilis pada 13 Juli 2012.

10. *Android* versi 4.4 (KitKat)

Android versi 4.4 (KitKat) direncanakan akan dirilis pada bulan Oktober 2013. Setelah sebelumnya beredar rumor bahwa *Android* versi berikutnya setelah *Jelly Bean* diperkirakan akan diberi nomor 5.0 dan dinamai *Key Lime Pe* [6].

2.2.3 Komponen Aplikasi *Android*

Fitur penting *Android* adalah bahwa satu aplikasi dapat menggunakan elemen dari aplikasi lain (untuk aplikasi yang memungkinkan). Sebagai contoh, sebuah aplikasi memerlukan fitur *scroller* dan aplikasi lain telah mengembangkan fitur *scroller* yang baik dan memungkinkan aplikasi lain menggunakannya. Maka pengembang tidak perlu lagi mengembangkan hal serupa untuk aplikasinya, cukup menggunakan *scroller* yang telah ada. Agar fitur tersebut dapat bekerja, sistem harus dapat menjalankan aplikasi ketika setiap bagian aplikasi itu dibutuhkan, dan pemanggilan objek java untuk bagian itu. Oleh karenanya *Android* berbeda dari sistem-sistem lain, *Android* tidak memiliki satu tampilan utama program seperti fungsi main() pada aplikasi lain. Sebaliknya, aplikasi memiliki komponen penting yang memungkinkan sistem untuk memanggil dan menjalankan ketika dibutuhkan [6].

2.2.4 Tipe Aplikasi *Android*

Terdapat tiga kategori aplikasi pada *Android* :

1. *Foreground Activity*

Aplikasi yang hanya dapat dijalankan jika tampil pada layar dan tetap efektif walaupun tidak terlihat. Aplikasi dengan tipe ini mempertimbangkan siklus hidup *activity*, sehingga perpindahan antar *activity* dapat berlangsung dengan lancar [12].

2. *Background Service*

Aplikasi yang memiliki interaksi terbatas dengan user, selain dari pengaturan konfigurasi, semua dari prosesnya tidak tidak tampak pada layar. Contohnya aplikasi penyaringan panggilan atau sms *auto respon* [6].

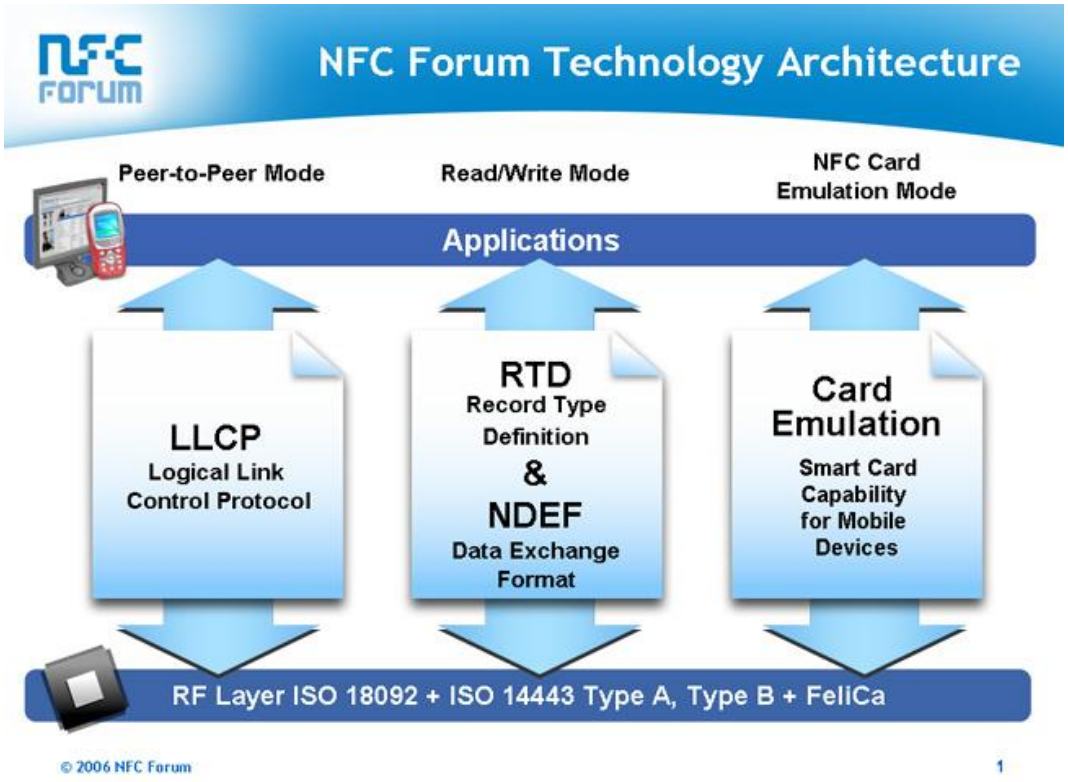
3. *Intermittent Activity*

Aplikasi yang masih membutuhkan beberapa masukkan dari pengguna, namun sebagian sangat efektif jika dijalankan di *background* dan jika diperlukan akan memberi tahu pengguna tentang kondisi tertentu. Contohnya pemutar musik. Untuk aplikasi yang kompleks akan sulit untuk menentukan kategori aplikasi tersebut apalagi aplikasi memiliki ciri-ciri dari semua kategori. Oleh karenanya perlu pertimbangan bagaimana aplikasi tersebut digunakan dan menentukan kategori aplikasi yang sesuai [6].

2.3 Near Field Communication

Near Field Communication atau NFC merupakan teknologi komunikasi baru dengan menggunakan induksi magnet berbasis teknologi *Radio Frequency Identification* (RFID). NFC mulai dikembangkan pada tahun 2002 oleh NXP dan Sony. Mereka berhasil membangun Forum NFC pertama bersama dengan Nokia. Sejak saat itu Forum NFC memiliki lebih dari 150 anggota dan peneliti yang terlibat dari beberapa perusahaan dan organisasi. Pada forum ini juga NFC dipromosikan dan secara resmi mengeluarkan sertifikasi untuk perangkat-perangkat yang mengadopsi teknologi NFC. NFC bisa juga disebut sebagai generasi kedua dari teknologi RFID.

NFC beroperasi pada frekuensi 13,56 MHz dengan kecepatan transmisi pengiriman mencapai 424 kbit/s. Jarak transmisi NFC sekitar 4-10 cm. Perbedaan antara NFC dan teknologi komunikasi contactless lainnya yaitu perangkat NFC dapat bersifat aktif – aktif (*peer to peer*) dan aktif – pasif. Oleh karena itu NFC. selalu melibatkan inisiator *(reader*) dan target. Inisiator aktif menghasilkan medan RF (*Radio Frequency*) yang dapat memberikan kekuatan ke target yang pasif (tidak memiliki sumber daya). Hal ini memungkinkan target NFC untuk memiliki bentuk yang sangat sederhana seperti stiker, gantungan kunci, atau kartu yang tidak memerlukan energy khusus [7].



Sumber gambar : NFC Forum [7]

Gambar 2.1 Arsitektur NFC

Banyak manfaat kegiatan-kegiatan yang diperoleh dengan memanfaatkan teknologi NFC. Seiring perkembangan jaman, invoasi-inovasi terhadap pemanfaatan NFC ini juga ikut berkembang. Perkembangan inovasi-inovasi tersebut terbagi ke dalam 3 kategori:

1. *Service Initiation*

Pada kategori ini NFC berguna untuk melakukan layanan sebagai media untuk mendapatkan informasi tertentu atau dapat juga untuk membuka layanan yang mengharuskan autentifikasi data terlebih dahulu. Hal ini melibatkan NFC *Tag* sebagai media penyimpanan maupun penyampaian sementara. Pada *Tag* tersebut dapat berisi informasi teks, alamat sebuah *website* (url), nomor telepon atau data lain yang ukuran yang kecil.

Salah satu contoh penggunaannya yaitu pada poster pintar (*smart* poster). Dengan mendekatkan ponsel ada kepada NFC *Tag* pada poster, maka pengguna

akan langsung mendapatkan informasi lebih detail pada yang belum tertera pada poster tersebut tanpa harus menginputkan apapun pada ponselnya.

2. Peer-to-peer

Pada kategori ini NFC berperan dalam mengkomunikasikan antara dua perangkat aktif dengan ukuran data yang relatif kecil. Namun untuk kebutuhan data yang berukuran besar, NFC dapat digunaka sebagai sambungan nirkabel seperti Bluetooth dan WiFi.

Contoh dari penggunaannya yaitu pada proses pencetakan foto langsung dari ponsel atau kamera. Pengguna tinggal memilih foto yang akan dicetak, kemudia mendekatakn ponselnya kepada printer yang sudah terintegrasi dengan teknologi NFC untuk membuat sambungan bluetooth untuk mengirimkan gambar dan mencetaknya.

3. Payment

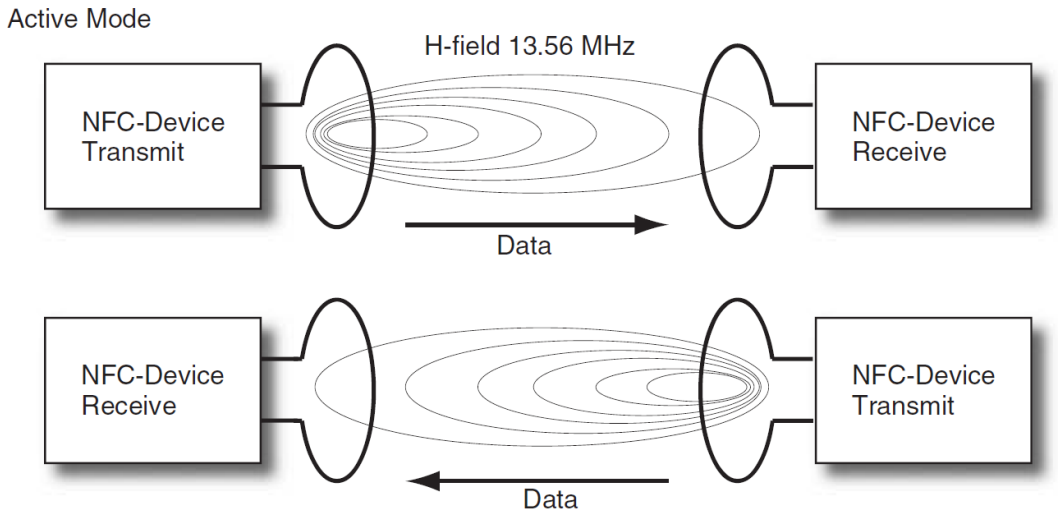
Pada kategori ini NFC berperan dalam proses pembayaran suatu barang yang sudah terintegrasi sebelumnya dengan kartu kredit, bank dan operator jaringan lainnya. Dengan memanfaatkan teknologi NFC dalam pembayaran ini dapat mempermudah dan mempercepat proses transaksi pembayaran karena tidak menggunakan uang tunai maupun metode pembayaran konvensional lainnya.

Contoh dari penggunaanya yaitu pada proses transaksi pembayaran di restoran, pusat perbelanjaan dan area parkir.

Komunikasi NFC dapat dilakukan dalam 2 mode yaitu mode aktif dan mode pasif.

1. Mode Aktif

Pada mode aktif, NFC yang berperan sebagai initiator mengaktifkan transmiter yang dimiliki sehingga arus dengan frekuensi tinggi yang mengalir pada antena menimbulkan medan magnetik H yang menyebar ke sekitarnya seperti terlihat pada gambar di bawah ini



Sumber gambar : NFC Forum [7]

Gambar 2.2 NFC Mode Aktif

Sebagian akan bergerak dan tertangkap oleh antena *interface* NFC yang terletak pada lokasi berdekatan. Kemudian tegangan U dihasilkan di antena yang dapat dideteksi dan diterima oleh receiver pada NFC target.

Cara kerja tersebut mirip dengan konsep RFID *Reader* dan transponder. Namun, terdapat perbedaan dimana pada transponder RFID *reader* harus memberikan suplai energi kepada transponder untuk mengoperasikan microchip untuk komunikasi data. Sementara pada NFC, perangkat elektronik yang memiliki fitur NFC yang memberikan suplai energi.

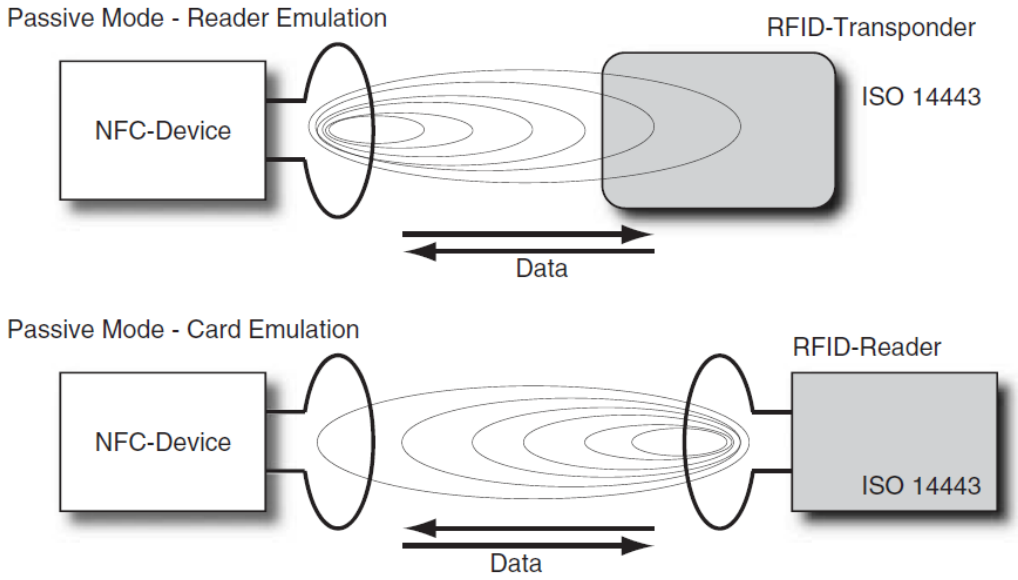
Pada mode aktif, pengiriman data dilakukan dari NFC target kepada NFC initiator dengan NFC target mengaktifkan transmitter dan NFC initiator berpindah mengaktifkan receiver. Sehingga pada mode ini kedua *interface* NFC bergantian menghasilkan medan magnetik.

2. Mode Pasif

Pada mode pasif, NFC inisiator juga menghasilkan medan magnetik untuk memulai komunikasi dengan NFC target. Namun setelah paket data sampai ke target, NFC inisiator tidak mengganti mode menjadi penerima seperti NFC mode aktif, melainkan tetap mengaktifkan transmitter. Sehingga trasmitter hanya diaktifkan pada NFC inisiator.

Kemudian data pada NFC target akan “ditumpangkan” pada gelombang tersebut dengan menghasilkan yang disebut sebagai load modulation. Hal ini memberikan keuntungan dalam pengoperasiannya pada kondisi daya yang dimiliki oleh NFC target sedang lemah.

Hal ini juga memungkinkan *interface* NFC berfungsi seperti RFID *Reader* untuk berkomunikasi dengan transponder pasif dengan memberikan suplai daya pada transponder dan menggunakan load modulation data ditransimisikan pada *interface* NFC. Misalnya, smartphone dengan fitur NFC dapat digunakan untuk membaca dan menulis pada transponder seperti smart label atau e-ticket. Aktivitas *interface* NFC yang menyerupai RFID *Reader* disebut juga ‘*reader*-emulation mode’[7].

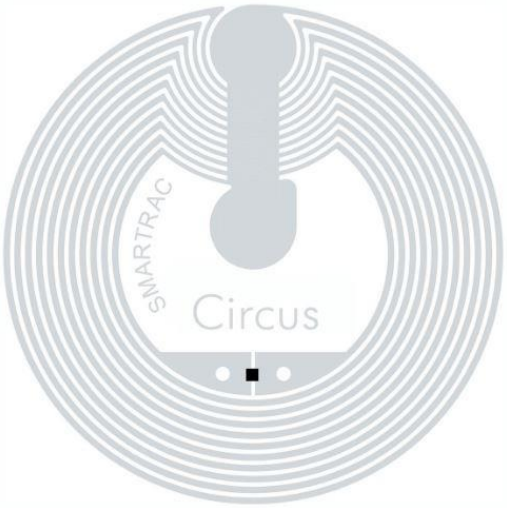


Sumber gambar : NFC Forum [7]

Gambar 2.3 NFC Mode Pasif

2.3.1 NFC *Tag*

NFC *Tag* merupakan media pendukung teknologi NFC. NFC *Tag* biasanya bersifat pasif, sehingga membutuhkan inisiator (NFC bersifat aktif) untuk proses komunikasi. NFC *Tag* dapat berisi data dengan ukuran kecil. . NFC *Tag* saat ini memiliki memory berkisar antara 96 dan 4.096 byte. NFC *Tag* berisfat read-only tetapi memungkinkan juga bersifat re-writeable.



Sumber gambar : Igoe T, Coleman D, Jepson D [8]

Gambar 2.4 NFC *Tag*

Produsen NFC telah menentukan standar dari NFC *Tag* untuk dapat dioperasikan dengan perangkat NFC. Spesifikasi yang ditentukan terbagi menjadi 4 tipe. Keempat tipe *Tag* NFC tersebut didasarkan pada produk dan ketersediaan komersil. Berikut adalah 4 tipe dari NFC *Tag* :

1. Tipe 1 : didasari pada ISO/IEC 14443A. *Tag*s mampu membaca dan menulis ulang. Pengguna dapat mengatur *tag* menjadi read-only. Memori yang tersedia adalah 96 bytes dan dapat di-upgrade hingga 2 Kbyte.
2. Tipe 2 : didasari pada ISO/IEC 14443A. *Tag*s mampu membaca dan menulis ulang. Pengguna dapat mengatur *tag* menjadi read-only. Memori yang tersedia adalah 48 bytes dan dapat di upgrade hingga 2 Kbyte.
3. Tipe 3 : didasari pada standar industri Jepang atau Japanese industrial standard (JIS) X 6319-4, atau disebut Felica. *Tag* diatur pada pembuatannya untuk mendukung baik dalam read dan re-writeable, ataupun read-only. Memori yang

tersedia dapat diubah, secara teori batas memorinya adalah 1 Mbyte per service.

4. Tipe 4 : kompatibel sepenuhnya dengan ISO/IEC 14443 standard series, memiliki memori 32 KB dan lebih cepat daripada tipe *tag* lainnya, memiliki proteksi terhadap data collision, dapat menggunakan komunikasi NFC-A maupun NFC-B. Tipe 4 dapat ditulis ulang (rewriteable) atau read-only. Tetapi pengaturan tersebut dilakukan saat tipe 4 dalam tahap pembuatan. Saat sudah selesai dibuat, pengaturannya tidak dapat diubah [8].

2.3.2 NFC *Reader*

NFC *reader* adalah perangkat yang menerima atau mengirimkan perintah dari NFC. NFC *reader* akan sangat berguna bagi pedagang yang ingin menyediakan layanan pembayaran menggunakan NFC [9].



Sumber gambar : M. Krisnanda [9]

Gambar 2.5 NFC *Reader*

2.4 Qr Code

QR Code adalah image berupa matriks dua dimensi yang memiliki kemampuan untuk menyimpan data di dalamnya. QR Code merupakan evolusi dari kode batang. Barcode merupakan sebuah simbol penandaan objek nyata yang terbuat dari pola batang-batang berwarna hitam dan putih agar mudah untuk dikenali oleh komputer. Contoh QR Code bisa dilihat pada gambar berikut.



Sumber gambar : Rahayu and Y. D [10]

Gambar 2.6 *Qr Code*

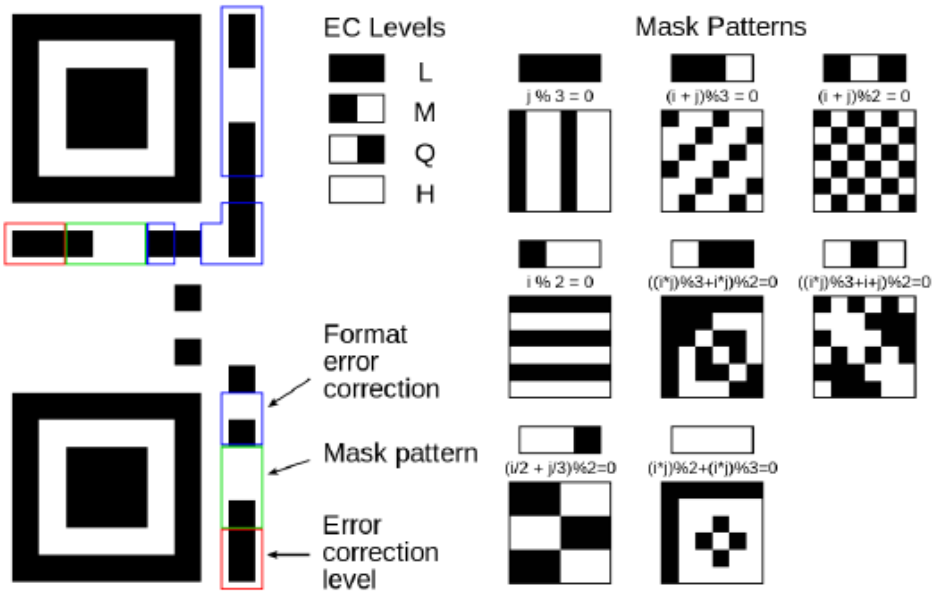
QR Code merupakan singkatan dari Quick Response Code, atau dapat diterjemahkan menjadi kode respon cepat. QR Code dikembangkan oleh Denso Corporation, sebuah perusahaan Jepang yang banyak bergerak di bidang otomotif. QR Code ini dipublikasikan pada tahun 1994 dengan tujuan untuk pelacakan kendaraan di bagian manufaktur dengan cepat dan mendapatkan respon dengan cepat pula [10].

Seiring berkembangnya QR Code, semakin banyak penelitian yang dilakukan mengenai kode simbol ini. Beberapa penelitian yang telah dilakukan diantaranya adalah :

1. Pembuatan aplikasi pembacaan QR Code menggunakan perangkat mobile berbasis J2ME.
2. QR Code untuk tandatangan digital.
3. QR Code untuk autentikasi novel user.
4. QR Code untuk edukasi [10].

2.4.1 *Encoding* Qr Code

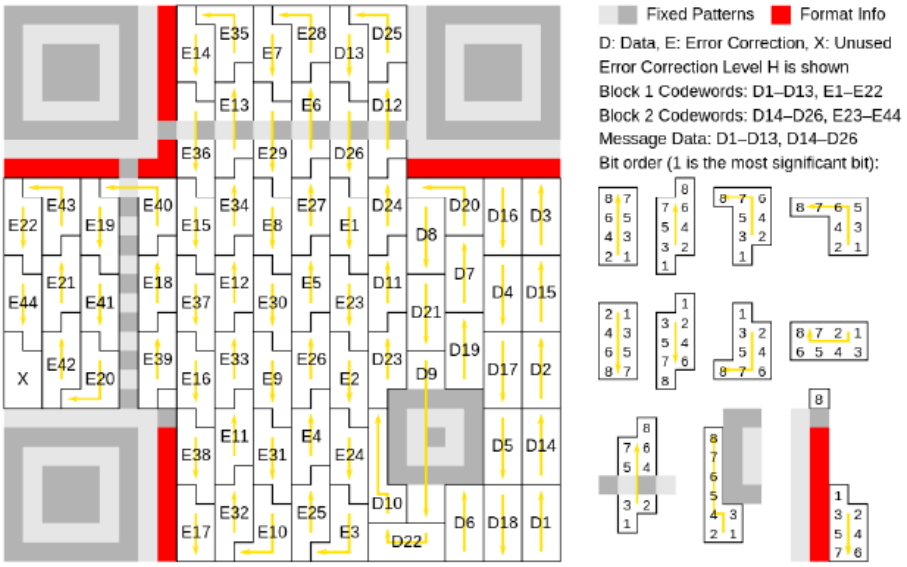
Informasi format mencatat dua hal: tingkat koreksi kesalahan dan mask pattern yang digunakan untuk simbol. Masking digunakan untuk memecah pola di area data yang mungkin membingungkan scanner, seperti daerah kosong besar atau fitur menyesatkan yang terlihat seperti tanda locator. Mask pattern didefinisikan pada grid yang diulang sediperlukannya untuk menutupi seluruh simbol. Modul yang sesuai dengan daerah gelap mask terbalik. Informasi format dilindungi dari kesalahan dengan kode BCH, dan dua salinan lengkap termasuk dalam setiap simbol QR.



Sumber gambar : Rahayu and Y. D [10]

Gambar 2.7 Informasi Format

Data pesan ditempatkan dari kanan ke kiri dalam pola zigzag. Dalam simbol yang lebih besar, cukup rumit karena adanya pola keselarasan dan penggunaan beberapa blok *interleaved* koreksi kesalahan.



Sumber gambar : Rahayu and Y. D [10]

Gambar 2.8 Ilustrasi Blok Interleaved Koreksi Kesalahan

Indikator 4 bit digunakan untuk memilih mode *encoding* dan menyampaikan informasi lainnya. Mode pengkodean dapat dicampur sesuai kebutuhan dalam simbol QR.

Tabel 2.1 Menu *Encoding*

Indikator	Keterangan
0001	Numeric *encoding* (10 bits per 3 digits)
0010	Alphanumeric *encoding* (11 bits per 2 characters)
0100	Byte *encoding* (8 bits per character)
1000	Kanji *encoding* (13 bits per character)
0011	Structured append (used to split a message across multiple QR symbols)
0111	Extended Channel Interpretation (select alternate character set or *encoding*)
0101	FNC1 in first position (see Code 128 for more information)
1001	FNC1 in second position
0000	End of message

Setelah setiap indikator yang memilih mode *encoding*, panjang *field* lah yang memberitahu berapa banyak karakter dikodekan dalam mode tersebut. Jumlah bit pada panjang *field* tergantung pada *encoding* dan versi simbol.

Tabel 2.2 Jumlah Bit Per Panjang Field

Encoding	Ver. 1-9	Ver.10-26	Ver. 27-40
Numerik	10	12	14
Alfanumerik	9	11	13
Byte	8	16	16
Kanji	8	10	12

Mode *encoding* alfanumerik menyimpan pesan lebih kompak daripada yang mode byte, tetetapi mode alfanumerik tidak bisa menyimpan huruf-huruf non-kapital dan memiliki pilihan tanda baca terbatas yang cukup untuk alamat *web* yang belum sempurna [10].

2.4.2 *Decoding* Qr Code

Tahapan *decoding* QR code dari pembacaan sampai menjadi output data merupakan kebalikan dari prosedur *encoding*, tahapan – tahapan *decoding* adalah sebagai berikut:

1. Cari dan temukan citra simbol, kenali modul gelap dan terang sebagai array “0” dan "1”.
2. Baca informasi format.
3. Baca version information dan tentukan version dari QR code
4. Lepaskan masking dengan cara menaplikasi operasi XOR pada daerah yang di masking dengan pattern masking yang di dapatkan dari ekstraksi informasi format.
5. Baca simbol karakter sesuai dengan aturan penempatan dan kembalikan data dan error correction codeword dari pesan.
6. Deteksi error menggunakan error correction codeword, jika di temukan error , perbaiki.
7. Pisahkan data codeword menjadi segment sesuai dengan mode indicator dan character count indicator.
8. Decode karakter sesuai dengan mode yang digunakan dan keluarkan hasil [10].

2.5 Instant Messaging

Instant Messaging atau yang biasa disebut *chatting* merupakan sebuah teknologi internet yang memungkinkan para pengguna dalam jaringan internet untuk mengirimkan pesan-pesan singkat secara langsung pada saat yang bersamaan (*real time*) dengan menggunakan teks kepada pengguna lainnya yang sedang terhubung ke jaringan yang sama.

Konsep yang digunakan oleh teknologi ini muncul pada awal-awal pengembangan sistem operasi UNIX dan jaringan internet. para pengguna yang sudah masuk log dapat mengirimkan perintah berupa *talk, write,* dan *finger* untuk melihat siapa saja yang sudah masuk log dan akhirnya mengirimkan pesan singkat kepada mereka.

2.5.1 Sejarah Instant Messaging

Perkembangan *Instant Messaging* diawali ketika mulai maraknya orang menggunakan teknologi secara online pada awal tahun 1990 ketika orang - orang mulai meluangkan banyak waktu untuk mengakses internet.

Para pengembang piranti lunak menciptakan sebuah *software* chat room, dimana suatu grup atau perorangan dapat melihat serta mengirimkan pesan kepada sertiap orang yang ada pada "*room*" tersebut. Penyedia jasa online Quantum Link menawarkan fitur untuk dapat saling berkirim pesan dengan sesama pengguna komputer yang sedang online. Penerapan *Instant Messaging* ini memiliki dasar yang tidak jauh berbeda, yaitu sebuah chat room yang ditujukan untuk dua orang.

Instant Messaging ini mulai meledak di internet pada November 1996, yaitu ketika Mirabilis memperkenalkan ICQ yang merupakan sebuah *Instant Messaging* yang dapat dipergunakan secara gratis bagi setiap orang. Pada tahun 1997, AOL menjadi pioneer dalam komunitas online. Hal ini dikarenakan AOL memberikan kemampuan bagi pengguna untuk dapat melakukan komunikasi dalam waktu yang sama dalam sebuah chat room dan *Instant Messaging*. Model ICQ inilah yang kemudian menjadi dasar yang penting dalam penggunaan *Instant Messaging* yang beredar sekarang ini.

Saat ini, perkembangan *Instant Messaging* telah mengalami kemajuan yang amat pesat. Penggunaan *Instant Messaging* yang awalnya hanya dapat diakses

oleh para penggunanya melalui komputer, sekarang ini telah dapat diakses melalui telepon genggam. Dengan kemampuan yang dimiliki telepon genggam untuk mengakses internet, maka para pengguna *Instant Messaging* dapat mengakses dunia maya kapan saja dan dimana saja.

Di Indonesia sendiri, pengguna *Instant Messaging* cukup banyak. Umumnya mayoritas pengguna berasal dari kaum pelajar dan mahasiswa. Menjamurnya piranti lunak seperti mig33, eBuddy, dan lain-lain memungkinkan telepon genggam untuk ber"IM" ria, membuat para penggemar Intsant Messaging semakin dimanja.

2.5.2 Fitur *Instant Messaging*

Beberapa fitur-fitut yang biasanya terdapat dalam sebuah aplikasi chatting, diantaranya :

1. Pesan singkat (*Instant messages*): Mengirimkan pesan kepada orang yang sedang online secara waktu nyata.
2. Obrolan (*Chat*): Menciptakan ruang obrolan (*chat room*) atau ruang diskusi dengan beberapa orang.
3. Tautan web (*Web links*): Berbagi pranala mengenai situs web favorit.
4. Video: Berbagi video.
5. Gambar (*Images*): Berbagi gambar.
6. Berkas (*Files*): Berbagi berkas komputer.
7. Bicara (*Talk*): Berfungsi seperti layaknya telepon.
8. Fasilitas dalam genggaman (*Mobile capabilities*): Mengirimkan pesan instan melalui telepon genggam.

2.6 Google Cloud Messaging

Google Cloud Messaging atau GCM merupakan layanan yang disediakan oleh google yang memungkinkan untuk mengirim data dari server ke client (*Android*-powered) perngkat mobile, dan menerima pesan dari perangkat yang terhubung dengan server yang sama. Layanan GCM menangani semua aspek antrian pesan dan pengiriman ke target aplikasi *Android* yang berjalan pada

perangkat target. GCM disediakan oleh google dengan cuma-cuma untuk devloper *Android* dan dalam penyimpanannya pun disediakan quota dalam jumlah yang besar oleh google untuk kebutuhan yang lebih luas.

Google Cloud Messaging yang disediakan hanya untuk *platform Android* sebagai layanan gratis yang membantu pengembang dalam mengirim dan menerima data dari server ke aplikasi *Android* yang akan dibangun. Kinerja yang ditawarkan layanan GCM ini ialah menyampaikan informasi baru yang masuk ke penyimpanan cloud, seperti halnya ketika e-mail mendapatkan pesan baru yang masuk ke mailbox [11].

2.7 Aplikasi *Mobile*

Menurut Buyens, aplikasi mobile berasal dari kata *application* dan *mobile*. *Application* yang artinya penerapan, lamaran, penggunaan. Secara istilah aplikasi adalah program siap pakai yang direka untuk melaksanakan suatu fungsi bagi pengguna atau aplikasi yang lain dan dapat digunakan oleh sasaran yang dituju sedangkan *mobile* dapat di artikan sebagai perpindahan dari suatu tempat ke tempat yang lain.

Maka aplikasi *mobile* dapat di artikan sebuah program aplikasi yang dapat dijalankan atau digunakan walaupun pengguna berpindah – pindah dari satu tempat ke tempat yang lain serta mempunyai ukuran yang kecil. Aplikasi *mobile* ini dapat di akses melalui perangkat nirkabel, pager, PDA, telepon seluler, *smartphone*, dan perangkat sejenisnya [12].

2.7.1 Karakteristik Perangkat *Mobile*

Perangkat mobile memiliki banyak jenis dalam hal ukuran, desain dan layout, tetapi mereka memiliki karakteristik yang sangat berbeda dari sistem desktop. Berikut karakteristik perangkat mobile, diantaranya :

1. Ukuran yang kecil

   Perangkat mobile memiliki ukuran yang kecil. Konsumen menginginkan perangkat yang terkecil untuk kenyamanan dan mobilitas mereka.

2. Memory yang terbatas

   Perangkat mobile juga memiliki memory yang kecil, yaitu *primary* (RAM) dan *secondary* (*disk*). Pembatasan ini adalah salah satu faktor yang mempengaruhi

penulisan program untuk berbagai jenis dari perangkat ini. Dengan pembatasan jumlah dari memory, pertimbangan-pertimbangan khusus harus diambil untuk memelihara pemakaian dari sumber daya yang mahal ini.

3. Daya proses yang terbatas

Sistem mobile tidaklah setangguh rekan mereka yaitu desktop. Ukuran, teknologi dan biaya adalah beberapa faktor yang mempengaruhi status dari sumber daya ini. Seperti harddisk dan RAM, Anda dapat menemukan mereka dalam ukuran yang pas dengan sebuah kemasan kecil.

4. Mengkonsumsi daya yang rendah

Perangkat mobile menghabiskan sedikit daya dibandingkan dengan mesin desktop. Perangkat ini harus menghemat daya karena mereka berjalan pada keadaan dimana daya yang disediakan dibatasi oleh baterai-baterai.

5. Kuat dan dapat diandalkan

Karena perangkat mobile selalu dibawa kemana saja, mereka harus cukup kuat untuk menghadapi benturan-benturan, gerakan, dan sesekali tetesan-tetesan air.

6. Konektivitas yang terbatas

Perangkat mobile memiliki bandwith rendah, beberapa dari mereka bahkan tidak tersambung. Kebanyakan dari mereka menggunakan koneksi wireless.

7. Masa hidup yang pendek

Perangkat-perangkat konsumen ini menyala dalam hitungan detik kebanyakan dari mereka selalu menyala. Coba ambil kasus sebuah handphone, mereka booting dalam hitungan detik dan kebanyakan orang tidak mematikan handphone mereka bahkan ketika malam hari. PDA akan menyala jika anda menekan tombol power mereka [12].

2.7.1.1 Activities

Activity merupakan bagian yang paling penting dalam sebuah aplikasi, karena *Activity* menyajikan tampilan visual program yang sedang digunakan oleh pengguna. Setiap *Activity* dideklarasikan dalam sebuah kelas yang bertugas untuk menampilkan antarmuka pengguna yang terdiri dari *Views* dan respon terhadap

Event. Setiap aplikasi memiliki sebuah *activity* atau lebih. Biasanya pasti akan ada *activity* yang pertama kali tampil ketika aplikasi dijalankan.

Perpindahan antara *activity* dengan *activity* lainnya diatur melalui sistem, dengan memanfaatkan *activity stack*. Keadaan suatu *activity* ditentukan oleh posisinya dalam tumpukan *acitivity*, LIFO (*Last In First Out*) dari semua aplikasi yang sedang berjalan. Bila suatu *activity* baru dimulai, *activity* yang sebelumnya digunakan maka akan dipindahkan ketumpukan paling atas. Jika pengguna ingin menggunakan *activity* sebelumnya, cukup menekan tombol *Back*, atau menutup *activity* yang sedang digunakan, maka *activity* yang berada diatas akan aktif kembali. *Memory Manager Android* menggunakan tumpukkan ini untuk menentukan prioritas aplikasi berdasarkan *activity*, memutuskan untuk mengakhiri suatu aplikasi dan mengambil sumber daya dari aplikasi tersebut [13].

Ketika *activity* diambil dan disimpan dalam tumpukkan *activity* terdapat 4 kemungkinan kondisi transisi yang akan terjadi :

2. *Active*

Setiap *activity* yang berada ditumpukan paling atas, maka dia akan terlihat, terfokus, dan menerima masukkan dari pengguna. *Android* akan berusaha untuk membuat *activity* aplikasi ini untuk untuk tetap hidup dengan segala cara, bahkan akan menghentikan *activity* yang berada dibawah tumpukkannya jika diperlukan. Ketika *activity* sedang aktif, maka yang lainnya akan dihentikan sementara [13].

3. *Paused*

Dalam beberapa kasus *activity* akan terlihat tapi tidak terfokus pada kondisi inilah disebut *paused*. Keadaan ini terjadi jika *activity* transparan dan tidak *fullscreen* pada layar. Ketika *activity* dalam keadaan *paused*, dia terlihat *active* namun tidak dapat menerima masukkan dari pengguna. Dalam kasus ekstrim, *Android* akan menghentikan *activity* dalam keadaan *paused* ini, untuk menunjang sumber daya bagi *activity* yang sedang aktif [13].

4. *Stopped*

Ketika sebuah *activity* tidak terlihat, maka itulah yang disebut *stopped*. *Activity* akan tetap berada dalam memori dengan semua keadaan dan informasi yang ada. Namun akan menjadi kandidat utama untuk dieksekusi oleh

sistem ketika membutuhkan sumberdaya lebih. Oleh karenanya ketika suatu *activity* dalam kondisi *stopped* maka perlu disimpan data dan kondisi antarmuka saat itu. Karena ketika *activity* telah keluar atau ditutup, maka dia akan menjadi *inactive* [13].

5. *Inactive*

Kondisi ketika *activity* telah dihentikan dan sebelum dijalankan. *Inactive activity* telah ditiadakan dari tumpukan *activity* sehingga perlu *restart* ulang agar dapat tampil dan digunakan kembali. Kondisi transisi ini sepenuhnya ditangani oleh manajer memori *Android*. *Android* akan memulai menutup aplikasi yang mengandung *activity inactive*, kemudian *stopped activity*, dan dalam kass luar biasa *paused activity* juga akan ditutup [13].

2.7.1.2 Services

Suatu *service* tidak memiliki tampilan antarmuka, melainkan berjalan di *background* untuk waktu yang tidak terbatas. Komponen *service* diproses tidak terlihat, memperbarui sumber data dan menampilkan notifikasi. *Service* digunakan untuk melakukan pengolahan data yang perlu terus diproses, bahkan ketika *Activity* tidak aktif atau tidak tampak [13].

2.7.1.3 Intents

Intents merupakan sebuah mekanisme untuk menggambarkan tindakan tertentu, seperti memilih foto, menampilkan halaman *web*, dan lain sebagainya. *Intents* tidak selalu dimulai dengan menjalankan aplikasi, namun juga digunakan oleh sistem untuk memberitahukan ke aplikasi bila terjadi suatu hal, misal pesan masuk. *Intents* dapat eksplisit atau implisit, contohnya jika suatu aplikasi ingin menampilkan *URL*, sistem akan menentukan komponen apa yang dibutuhkan oleh *Intents* tersebut [13].

2.7.1.4 Broadcast Receiver

Broadcast Receivers merupakan komponen yang sebenarnya tidak melakukan apa-apa kecuali menerima dan bereaksi menyampaikan pemberitahuan. Sebagian besar *Broadcast* berasal dari sistem misalnya, Batre sudah hampir habis, informasi zona waktu telah berubah, atau pengguna telah merubah bahasa *default* pada perangkat. Sama halnya dengan *service*, *Broadcast Receivers* tidak menampilkan antarmuka pengguna. Namun, *Broadcast Receivers* dapat menggunakan *Notification Manager* untuk memberitahukan sesuatu kepada pengguna [13].

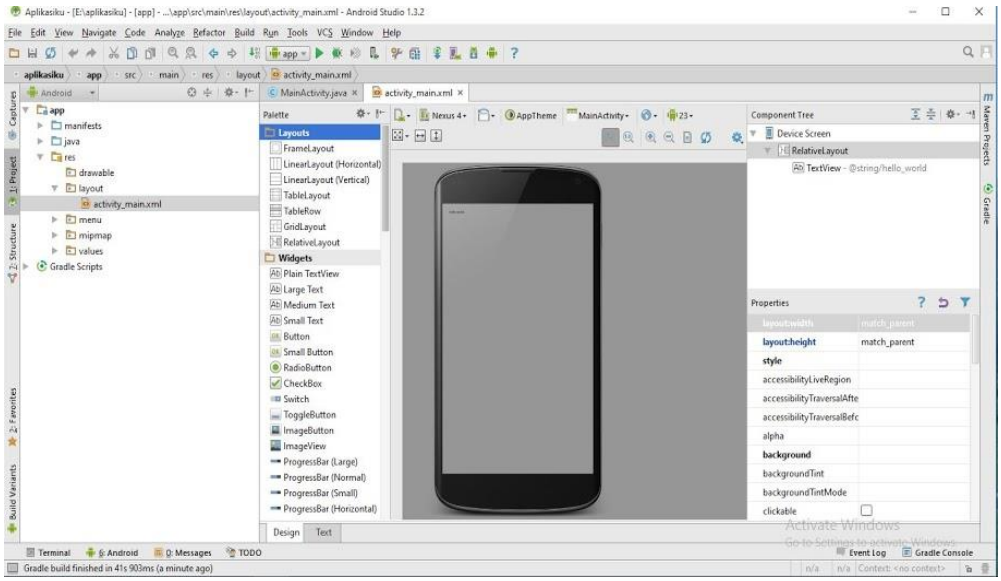
2.7.1.5 Content Providers

Content Providers digunakan untuk mengelola dan berbagi *database*. Data dapat disimpan dalam file sistem, dalam database *SQLite*, atau dengan cara lain yang pada prinsipnya sama. Dengan adanya *Content Provider* memungkinkan antar aplikasi untuk saling berbagi data. Komponen ini sangat berguna ketika sebuah aplikasi membutuhkan data dari aplikasi lain, sehingga mudah dalam penerapannya [13].

2.8 *Android* Studio

Android studio adalah lingkungan pengembangan terpadu (*IDE)* resmi untuk pengembangan *platform android*, hal itu disampaikan pada tanggal 16 mei 2013 Google I/O,*android* studio sudah tersedia secara bebas dibawah lisensi Apache 2.0.

Android studio pada awalnya tahap *preview* versi 0.1 yang dipakai pada tanggal 1 mei 2013 dan memasuki tahap beta pada bulan juni 2014 dan mulai stabil dirilis pada Desember 2014 dengan versi 1.0 ,berdasarkan *jetBrains* ‘IDEA *Intellij Software*, *android* studio dirancang khusus untuk pengembangan *android* yang tersedia untuk Windows,Mac OS X dan Linux sebagai pengganti *Eclipse* [14].



Sumber Gambar : Xavier Ducrohet [14]

Gambar 2.9 Layout *Android* Studio

2.8.1 Struktur *Android* Studio

Berikut rincian dari struktur *Android* Studio :

1. App : berfungsi untuk menyimpan file file pendukung aplikasi yang akan diolah.

   A. Manifest

   tempat dimana konfigurasi konfigurasi pendukung project serta izin (permision) untuk mengakses hardware.

   B. Java

   tempat berdiamnya *code/scripts* java untuk menjalankan aplikasi yang akan dibuat

   C. *Res* :

   1. *Drawable* : tempat untuk menyimpan gambar atau *icon* pendukung aplikasi.
   2. *Layout* : tempat berkumpulnya design tampilan yang akan tampak pada tampilan aplikasi anda.

3. *Menu* : berisi *code* berupa *file* xml yang didalamnya berisi *array* tentang menu yang akan tampil dalam *project* anda, misalkan *file, new,open,close,*dsb.
4. *Mipmap* : tempat berkumpulnya *icon launcher* (icon utama aplikasi) yang ukurannya menyesuaikan *device* yang digunakan.
5. *Values* : untuk menyimpan file .xml yang menggambarkan sumber dari berbagai jenis. Sebagai contoh folder berisi *file strings*.xml dengan definisi sumber *string*. Seperti yang akan anda lihat dari menambahkan bagian *color , folder* tata letak juga dapat berisi, misalnya, deskripsi warna. Di dalam file dimens.xml, terdapat *string-string* untuk konfigurasi dimensi aplikasi. Tidak jauh berbeda dengan dimens.xml maupun *strings*.xml, pada file styles.xml terdapat deklarasi variable untuk theme.

2. *Gradle scripts* : berfungsi sebagai *configurasi* aplikasi anda dengan *device,* serta menghubungkan *library* pendukung aplikasi yang anda buat. Penggunaan *android* studio ini untuk membuat aplikasi rumah pintar di *smartphone android* yang sebagai perangkat bantuan untuk mengontrol rumah pintar [14].

2.9 Database

Beberapa definisi basis data sebagai berikut :

1. Database adalah sekumpulan data store (bias dalam jumlah yang besar) yang tersimpan dalam *magnetic disk, optical disk, magnetic drum* atau media penyimpanan.
2. Database adalah sekumpulan program-program aplikasi umum yang bersifat "*batch*" yang mengeksekusi dan memproses data secara umum (seperti pencarian,peremajaan,penambahan, dan penghapusan terhadap data)
3. Database terdiri dari data yang akan digunakan atau diperuntukan terhadap banyak user (baik menggunakan teknik pemrosesan yang bersifat batch atau on-line) akan menggunakan data tersebut sesuai dengan tudas dan fungsinya, dan user lain dapat juga menggunakan data tersebut dalam waktu yang bersamaan.

2.9.1 SQL

SQL adalah sebuah bahasa permintaan database yang terstruktur. Bahasa SQL dibuat sebagai bahasa yang dapat merelasikan beberapa tabel dalam database maupun merelasikan antar database. Bahasa SQL ditulis langsung dalam sebuah program databases sehingga pengguna dapat melihat langsung permintaan yang diinginkan sekaligus melihat hasilnya. SQL tidak hanya mengambil informasi dari tabel-tabel *database* SQL dapat digunakan dalam membuat elemen dalam *database*, memasukan, mengubah dan menghapus data dari *database*. Dengan memanfaatkan bahasa web seperti HTML, dan PHP, SQL merupakan tool penting yang tidak dapat ditinggalkan untuk pengembangan aplikasi database berbasis web. Query dasar yang sering digunakan adalah *create, read, update* dan *delete*.

1. *Create*

Digunakan untuk menginputkan data kesebuah tabel di database, contoh querynya sebagai berikut:

[INSERT INTO anggota(nama_anggota, umur) VALUES(‘Asep’, 21)]

2. *Read*

Read digunakan untuk menampilakan data sesuai dengan ketentuan tertentu, contoh querynya sebagai berikut :

[SELECT * FROM anggota]

3. *Update*

Update digunakan untuk mengubah data sesuai dengan apa yang diinginkan, contoh querynya adalah sebagai berikut:

[UPDATE anggota SET umur = ‘21’ WHERE id_anggota = 1]

4. *Delete*

Delete digunakan untuk menghapus data yang ingin dihilangkan, contoh querynya sebagai berikut:

[DELETE FROM anggota WHERE id_anggota =5]

2.9.1.1 DDL

DDL atau *Data Definition Language* adalah sebuah metode *query* sql yang berguna untuk mendefinisikan data pada sebuah *database*, adapun *query* yang dimiliki adalah:

1. CREATE : Digunakan untuk melakukan pembuatan tabel dan *database*
   contoh querynya sebagai berikut :
   [Create : mysql > CREARE DATABASE nama_database;]
2. DROP : Digunakan untuk melakukan penghapusan tabel maupun *database*
   contoh querynya sebagai berikut :
   [Drop : mysql > DROP DATABASE nama_database;]
3. ALTER : Digunakan untuk melakukan perubahan struktur tabel yang telah dibuat, baik menambah *field* (*add*), mengganti nama *field* (*change*) ataupun menamakannya kembali (*rename*) serta menghapus (*drop*)
   contoh querynya sebagai berikut :
   [Alter : mysql > ALTER nama_tabel alter_spesifikasi kondisi;]

2.9.1.2 DML

DML atau *Data Manipulation Language* adalah sebuah metode *query* yang dapat digunakan apabila DDL telah terjadi, sehingga fungsi dari *query* ini adalah untuk melaukan pemanipulasian *database* yang telah ada atau telah dibuat sebelumnya. Adapun *query* yang termasuk di dalamnya adalah:

1. INSERT : Digunakan untuk melakukan penginputan pada tabel di *database.*
   contoh querynya sebagai berikut :
   [INSERT : mysql > INSERT INTO nama_tabel VALUES (isi_field);]
2. UPDATE : Digunakan untuk melakukan pengubahan atau peremajaan terhadap data yang ada pada tabel.
   contoh querynya sebagai berikut :
   [UPDATE : mysql > UPDATE nama_tabel SET isi_field=“data_baru“;]

3. DELETE : Digunakan untuk melakukan penghapusan data pada tabel. Penghapusan ini dapat dilakukan secara sekaligus (seluruh isi tabel) maupun hanya beberapa *Recordset.*
   contoh querynya sebagai berikut :
   [DELETE : DELETE nama_tabel WHERE kondisi;]

2.9.1.3 DCL

DCL atau *Data Control Language* merupakan bahasa yang digunakan untuk mengendalikan pengaksesan data pada *database,* adapun perintah yang dimiliki DCL adalah:

1. GRANT : Untuk memberikan hak akses atau izin oleh *administrator server* kepada pengguna.
2. REVOKE : Untuk menghilangkan atau mencabut hak akses yang telah diberikan kepada *user* oleh *administrator*.

2.9.2 MySQL

MySQL adalah sebuah program *database* server yang mampu menerima dan mengirimkan datanya dengan sangat cepat, *multi user* serta menggunakan perintah standar SQL (*Structured Query Laguage*). MySQL merupakan *database* server *multi user* dan *multi threaded* yang tangguh. Dengan memiliki banyak *feature* MySQL bisa bersaing dengan *database* komsersial sekalipun. Tidak mengejutkan MySQL menjadi *database* pilihan untuk banyak pengguna PHP. MySQL termasuk RDBMS (*Relational Database Management System*) itulah sebabnya istilah seperti tabel, baris dan kolom digunakan pada MySQL. Pada MySQL sebuah *database* mengandung satu atau sejumlah tabel. Tabel terdiri dari sejumlah baris dan setiap baris mengandung satu atau beberapa kolom.

MySQL mampu menangani data yang cukup besar. Perusahaan yang mengembangkan MySQL yaitu TcX, mengaku menyimpan data lebih dari 40 *database*, 10.000 tabel dan sekitar 7 juta baris, totalnya kurang lebih 100 *Gigabyte*

data. SQL adalah sebuah standar yang digunakan untuk mengakses *database server*. Bahasa ini pada awalnya dikembangkan oleh IBM, namun telah diadopsi dan digunakan sebagai standar industri.

Dengan menggunakan MySQL, proses akses *database* menjadi lebih *userfriendly* dibandingkan dengan menggunakan *dBASE* atau *Clipper* yang masih menggunakan perintah-perintah pemrograman. MySQL merupakan *software database* yang masih yang masih popular di lingkungan *Linux*, kepopuleran ini sangat ditunjang oleh performa *query* dari databasenya yang saat ini bisa dikatakan paling cepat dan jarang bermaslah. MySQL ini juga sudah berjalan pada lingkungan windows [15].

MySQL juga memiliki beberapa kelebihan, antara lain :

1. *Portability*

   MySQL dapat berjalan stabil pada berbagai sistem operasi seperti *windows, Linux, FreeBSD, Solaris* dan lain-lain.

2. *Open Source*

   MySQL didistribusikan secara *open source* (gratis), dibawah lisensi GPL sehingga dapat digunakan cuma-Cuma.

3. *Multi User*

   MySQL dapat digunakan oleh beberapa *user* dalam waktu yang bersamaan tanpa mengalami masalah atau konflik.

4. *Performance Tuning*

   MySQL memiliki kecepatan yang menakjubkan dalam menangani *query* sederhana, dengan kata lain dapat memproses lebih banyak SQL per satuan waktu.

5. *Coloumn Types*

   MySQL memiliki tipe kolom yang sangat kompleks, seperti *integer, double, char, text, date* dan lain-lain.

6. *Command and Function*

   MySQL memiliki operator dan fungsi secara penuh yang mendukung perintah *select* dan *where* dalam *query*.

7. *Security*

MySQL memiliki beberapa lapisan sekuritas seperti level *subnetmask*, nama *host*, dan izin akses *user* dengan sistem perizinan yang mendetail serta password terenkripsi.

8. *Scability and Limits*

MySQL mampu menangani *database* dalam skala besar, dengan jumlah *records* lebih dari 50 juta dan 60 ribu tabel serta 5 milyar baris. Selain itu batas *indeks* yang dapat ditampung mencapai 32 *indeks* pada tiap tabelnya.

9. *Connectivity*

MySQL dapat melakukan koneksi dengan *clients* menggunakan protokol TCP/IP, *Unix socket* (UNIX) atau *Named Pipes* (NT).

10. *Localisation*

MySQL dapat mendeteksi pesan kesalahan pada *client* dengan menggunakan lebih dari dua puluh bahasa. Meskipun demikian, bahasa Indonesia belum termasuk didalamnya.

11. *Interface*

MySQL memiliki *interface* (antar muka) terhadap berbagai aplikasi dan bahasa pemograman dengan menggunakan fungsi API (*Aplication Programming Interface*).

12. *Clients and Tools*

MySQL dilengkapi dengan berbagai tools yang dapat digunakan untuk administrasi *database* dan pada setiap tool yang ada disertakan petunjuk online.

13. Struktur Tabel

MySQL memiliki struktur tabel yang lebih fleksibel dalam menangani *ALTER TABLE*, dibandingkan database lainnya semacam *PosgreeSQL* ataupun *Oracle*.

Kelemahan MySQL dari dulu sampai saat ini adalah *feature-creep* artinya MySQL berusaha kompatibel dengan beberapa standar serta berusaha memenuhinya namun jika itu diungkapkan kenyataannya bahwa fitur-fitur tersebut

belum lengkap dan belum berperilaku sesuai standar. Contoh fitur *SUB-SELECT* (nesting *SELECT* dalam *SELECT*) yang tidak optimal dan sering salah parsing *query* SQL dan jalan keluarnya dengan memecah menjadi beberapa query [15].

2.10 Javascript

JavaScript adalah bahasa *scripting* yang popular di internet dan dapat bekerja di sebagian besar *browser* popular seperti *Internet Explorer* (IE), *Mozilla FireFox, Netscape*, dan *Opera*. Kode JavaScript dapat disisipkan dalam halaman web menggunakan *tag* SCRIPT.

Keunggulan javascript yang juga dikenal dengan nama ECMAScript yaitu dapat berjalan di semua *platform* dengan *browser* yang mendukung JavaScript, dan hampir semua *platform* dan *browser* saat ini mendukung JavaScript. Contoh dari aplikasi yang dibangun dengan JavaScript adalah *Google Maps* yang dapat berjalan di atas *Linux, Windows,* dan *Mac OS*. JavaScript juga semakin dan terus berkembang, seperti pertumbuhan pustaka (*library*) yang memudahkan untuk menavigasi dokumen, memlilih elemen DOM, membuat animasi, menangani *event* dan mengembangkan aplikasi *Ajax*. JavaScript adalah bahasa pemograman *client-side* yang *cross-platform* (berjalan di banyak *platform*) dan bersifat bebas (untuk dimodifikasi dan gratis) juga diadopsi secara universal [16].

2.11 JSON

JSON merupakan bagian dari bahasa pemrograman JavaScript (Standard ECMA-262 3rd *Edition – December* 1999). JSON merupakan format teks yang sepenuhnya independen tetapi menggunakan konvensi yang familiar dengan bahasa pemrograman dari keluarga-C, termasuk C, C++, C#, Java, JavaScript, Perl, Python, dan sebagainya. Kelebihan inilah yang membuat JSON menjadi sebuah bahasa *data-interchange* yang ideal.

JSON dibangun dalam dua struktur:

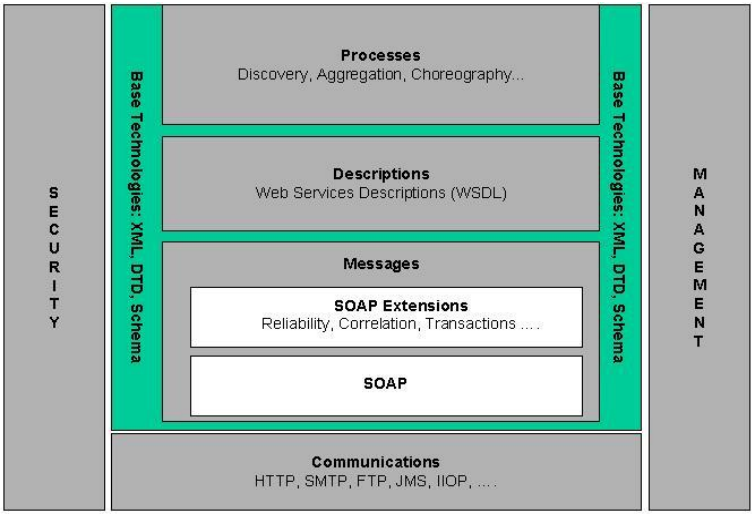
1 Beberapa pasangan dari nama/nilai. Dalam beberapa bahasa pemrograman biasa disebut dengan istilah *object, record, struct, tabel hash, key list* atau *associative array*.
2 Nilai-nilai yang terusun secara *ordered list*. Biasa disebut dengan *array, vector, list* atau daftar dalam bahasa pemrograman.

JSON adalah struktur data yang *universal*, dalam artian bisa digunakan dalam berbagai bahasa pemrograman. Hampir semua bahasa pemrograman mendukung penuh JSON dalam berbagai format. Hal ini memungkinkan format data yang dapat dipertukarkan menggunakan bahasa pemrograman juga menggunakan dasar dari struktur JSON.

2.12 Web Service

Web service adalah sebuah entitas komputasi yang dapat diakses melalui jaringan internet maupun intranet dengan standar protokol tertentu dalam *platform* dan antarmuka bahasa pemograman yang independen. Tujuan pengembangannya adalah untuk menjembatani komunikasi antar program, sehingga aplikasi yang satu dan aplikasi yang lain yang terdapat pada suatu jaringan yang sama atau pada jaringan berbeda dapat saling berkomunikasi asalkan menggunakan standart protokol yang ditetapkan oleh *web service*. Hal ini bisa terjadi, karena standart protokol itu tidaklah terikat pada suatu *platform* atau bahasa pemograman. Protokol itu sendiri dibangun oleh *Extensible Markup Language* (XML) yang memang kenyataannya telah didukung oleh banyak *paltform*, bahasa pemograman, dan oleh *developer* di seluruh dunia.

Arsitektur *web service* dibangun oleh beberapa layer dan teknologi yang saling berhubungan. Banyak cara untuk memvisualisaskan *service*, sama banyaknya dengan cara untuk membangun dan menggunakan *web service* gambar berikut hanyalah salah satu cara untuk menggambarkan arsitekturnya.



Sumber gambar : Purba, Siregar [17]

Gambar 2.10 Arsitektur Web Service

Web service disusun oleh tiga komponen standart, yaitu :

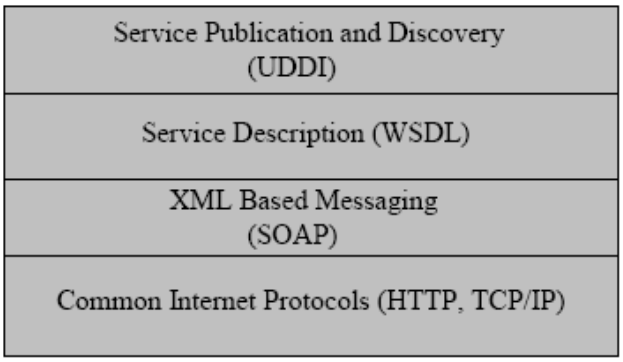
1. *Simple Object Access Protocol* (SOAP), yaitu protokol yang bertanggungjawab dalam pertukuran informasi dalam lingkungan jaringan terdistribusi.
2. *Web service definition Language* (WDSL), dokumen standart yang dituliskan pada format XML, dan mendifinikasan kehadiran web *service* dalam suatu jaringan.
3. *Universal description, Discovery and Inttegration* (UDDI), yaitu suatu lokasi direktori yang beriskan *service* (layanan) dan bersifat bebas *platform* (*platform independent*), dituliskan berbasis XML dan dapat diakses oleh entittas yang berada di dalam dan luar jaringan.

Adanya standart tersebut membuat web *service* mudah di akses melalui berbagai antarmuka dan juga memberi peluang dimungkinkannya berbagai sistem yang dibangun pada *platform* yang berbeda dan bahasa yang berbeda untuk berkolaborasi pada dalam suatu pekerjaan [17].

2.12.1 Komponen-Komponen *Web Service*

Web service secara keseluruhan memiliki empat layer komponen yaitu:

1. Layer 1: Protokol internet standar seperti HTTP, TCP/IP
2. Layer 2: *Simple Object Access Protocol* (SOAP), merupakan protokol akses objek berbasis XML yang digunakan untuk proses pertukaran data/informasi antar layanan.
3. Layer 3*: Web Service Definition Language* (WSDL), merupakan suatu standar bahasa dalam format XML yang berfungsi untuk mendeskripsikan seluruh layanan yang tersedia.



Sumber gambar : Purba, Siregar [17]

Gambar 2.11 Komponen-Komponen Web Service

2.13 Perancangan Berorientasi Objek

Teknologi objek menganalogikan sistem aplikasi seperti kehidupan nyata yang didominasi oleh objek. Didalam membangun sistem berorientasi objek akan menjadi lebih baik apabila langkah awalnya didahului dengan proses analisis dan perancangan yang berorientasi objek. Tujuannya adalah mempermudah *programmer* didalam mendesain program dalam bentuk objek-objek dan hubungan antar objek tersebut untuk kemudian dimodelkan dalam sistem nyata. Suatu perusahaan *software* yaitu *Rational Software*, telah membentuk konsarium dengan berbagai organisasi untuk meresmikan pemakaian *Unified Modelling Language* (UML) sebagai bahasa standar dalam *Object Oriented Analysist Design* (OOAD) [18].

2.13.1 UML (*Unified Modeling Language*)

UML adalah kependekan dari *Unified Modeling Language* yang merupakan suatu cara untuk menyelesaikan suatu masalah dengan mendeskripsikannya yang telah menjadi standar dalam dunia industri untuk memvisualisasikan, merancang dan mendokumentasikan sistem perangkat lunak. Dengan menggunakan UML kita dapat membangun model untuk segala bentuk dan jenis aplikasi perangkat lunak, yang mana aplikasi yang dibangun dapat berjalan pada perangkat lunak dengan sistem operasi dan jaringan apapun. Selain itu, dengan menggunakan UML, programmer dapat mengerti, memahami dengan jelas maksud, tujuan dan arah rancangan sistem, sehingga sistem perngakat lunak yang dirancang dapat dibangun dengan bahasa program apapun. Namun karena UML merupakan suatu model pengembangan sistem perangkat lunak yang berbasis *object oriented* sehingga menggnuakan bentuk *class* dan *operation* dalam konsep dasarnya, maka lebih cocok dalam pemrogramannya dengan menggunakan bahasa-bahasa pemrograman yang berbasis OOP (*Object Oriented Programming*) seperti C, C++, Java, VB dan lain sebagainya [18].

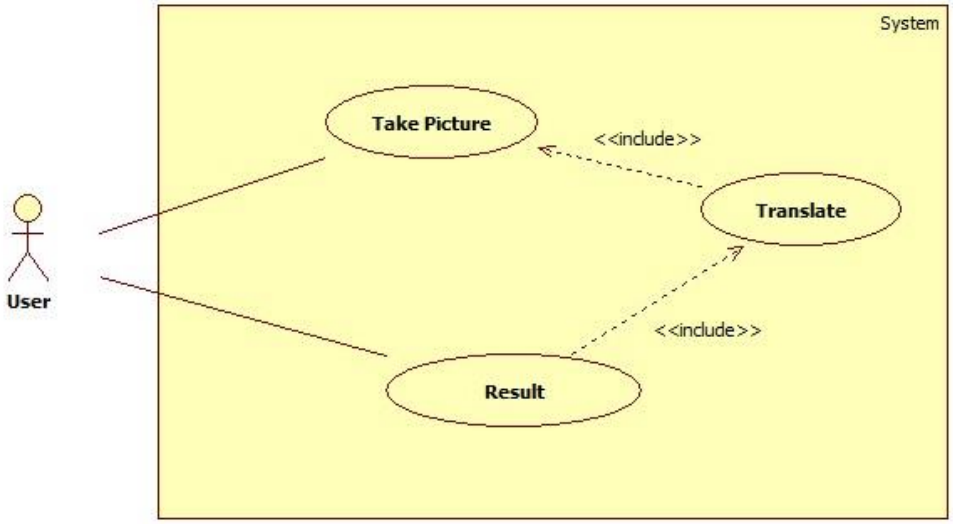
Secara lebih mendalam, UML lebih dari sekedar sebuah standard dan penemuan dari suatu notasi-notasi yang disatukan, tetapi juga berisi konsep-konsep baru yang menarik yang tidak ditemukan secara umum dalam komunitas *object oriented*. Dalam membangun suatu model perangkat lunak dengan UML, digunakan bentuk-bentuk diagram atau symbol untuk merepresentasikan elemen-elemen dalam sistem. UML ini digunakan untuk perancangan sistem yang nanti akan digunakan pada Bab 3 pada tugas akhir ini. Bentuk diagram yang digunakan untuk merepresentasikannya adalah sebagai berikut:

1. *Use-case* Diagram.
2. *Class* Diagram.
3. *Sequence* Diagram.
4. *Collaboration* Diagram.
5. *Activity* Diagram.
6. *Component* Diagram.
7. *Deployment* Diagram.

2.13.1.1 *Use-case* Diagram

Use-case diagram merupakan suatu bentuk diagram yang menggambarkan fungsi-fungsi yang diharapkan dari sebuah sistem yang dirancang. Dalam *Usecase diagram* penekanannya adalah "*apa*" yang diperbuat oleh sistem, dan bukan "*bagaimana*". Sebuah *use-case* akan merepresentasikan sebuah interaksi antara pelaku atau *actor* dengan sistem.

Use-case diagram yang digunakan dalam mercancang suatu sistem dapat sangat membantu pada saat menyusun *requirement* sebuah sistem, mengomunikasikannya dengan klien, dan merancang pengujian untuk semua fitur yang terdapat dalam sistem. Dalam suatu sistem aplikasi *database*, *use-case diagram* sangat membantu *requierement* apa saja yang diperlukan [19].



Gambar 2.12 *Use-case* Diagram

2.13.1.2 *Class* Diagram

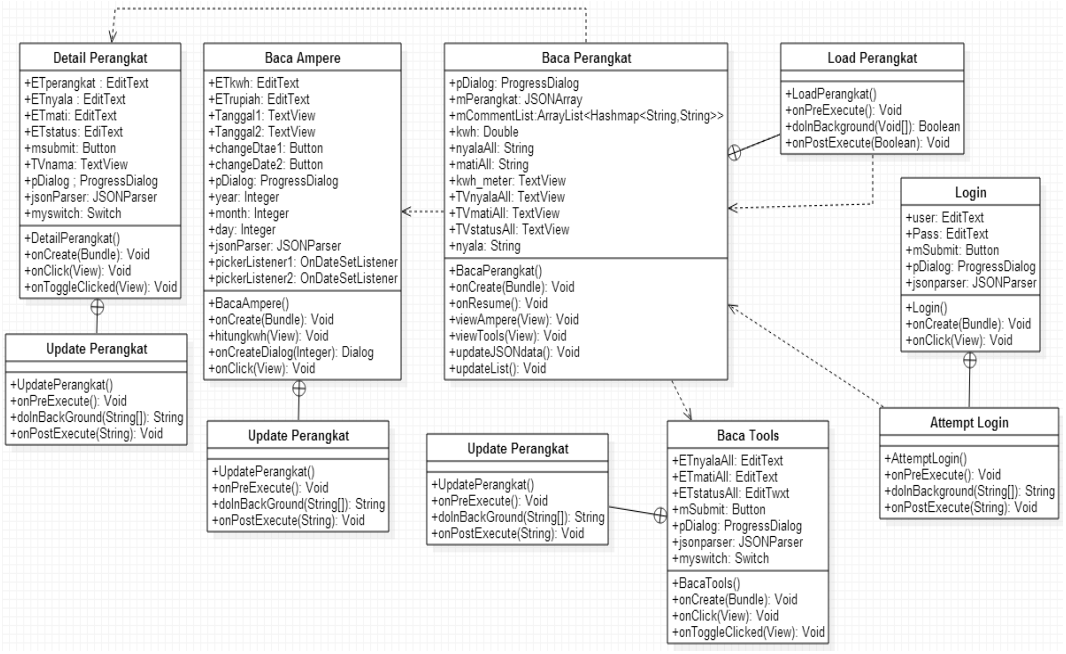
Sebuah *Class Diagram* menunjukkan struktur yang statis dari beberapa *class* dalam suatu sistem. *Class-class* merepresentasikan suatu keadaan (atribut/properti) dan yang akan dikerjakan oleh sistem (metoda/fungsi). *Class* memiliki tiga area pokok :

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

Atribut dan metoda dalam class diagram dapat memiliki salah satu sifat seperti berikut di bawah ini:

1. *Private*, hanya dapat diakses oleh *class* itu sendiri.
2. *Protected*, hanya dapat diakses oleh *class* itu sendiri dan turunan dari *class* tersebut.
3. *Public*, dapat diakses oleh *class* selain dari *class* yang bersangkutan.

Class dapat direpresentasikan dalam sebuah *interface* atau sebaliknya merupakan implementasi dari sebuah *interface* yang berupa *class* abstrak yang hanya tidak memiliki *attribute* dan hanya memiliki metoda. Berikut merupakan bentuk *class diagram* secara umum.

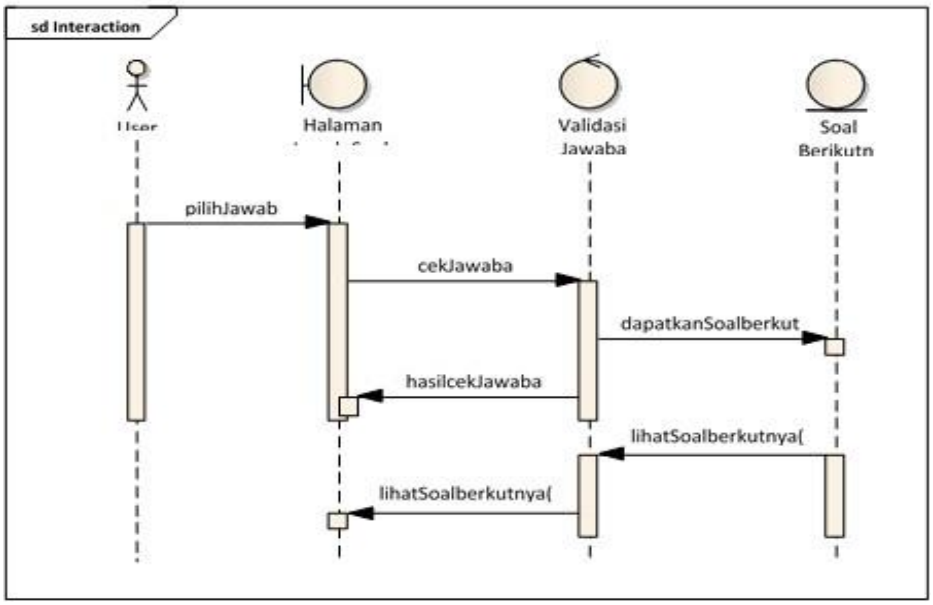


Gambar 2.13 *Class* Diagram

2.13.1.3 Sequence Diagram

Sequence Diagram merupakan diagram yang mengambarkan kolaborasi yang dinamis antara obyek satu dengan yang lain. Kolaborasi ini ditunjukkan dengan adanya interaksi antar obyek di dalam dan di sekitar sistem yang berupa pesan atau instruksi yang berurutan.

Sequence diagram umumnya digunakan untuk menggambarkan suatu skenario atau urutan langkah-langkah yang dilakukan baik oleh *actor* maupun sistem yang merupakan respon dari sebuah kejadian untuk mendapatkan hasil atau *output.*

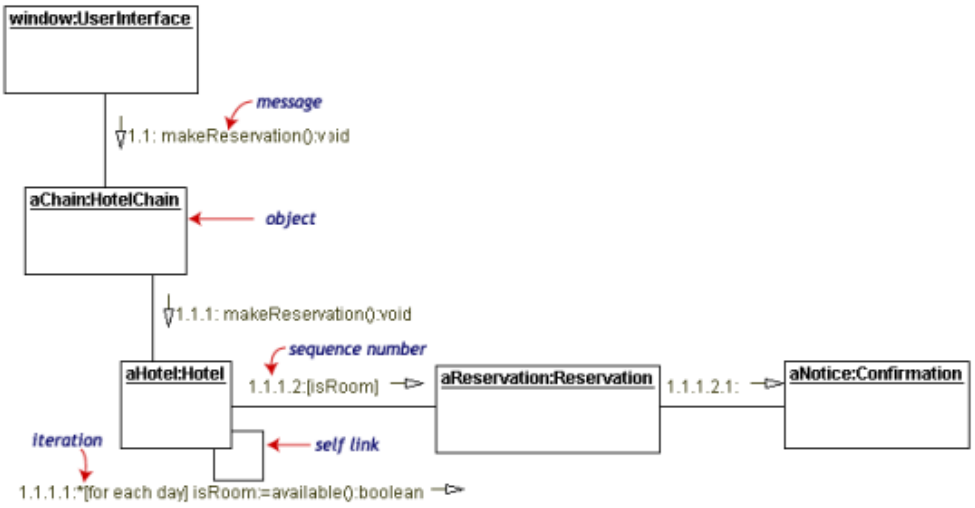


Sumber gambar :Sri Dharwiyanti dan romi Satria Wahono [19]

Gambar 2.14 *Sequence* Diagram

2.13.1.4 *Collaboration* Diagram

Diagram kolaborasi adalah diagram interaksi yang menekankan organisasi strutkural dari objek-objek yangmenerima serta mengirim pesan (*message*).Sebuah *collaboration diagram* menunjukkan kolaborasi yang dinamis yang mirip dengan *sequence diagram*. *Collaboration diagram* digambarkan sebagai sebuah *obiject* diagram dimana sejumlah obyek ditunjukkan disekitarnya dengan hubungan-hubungannya.

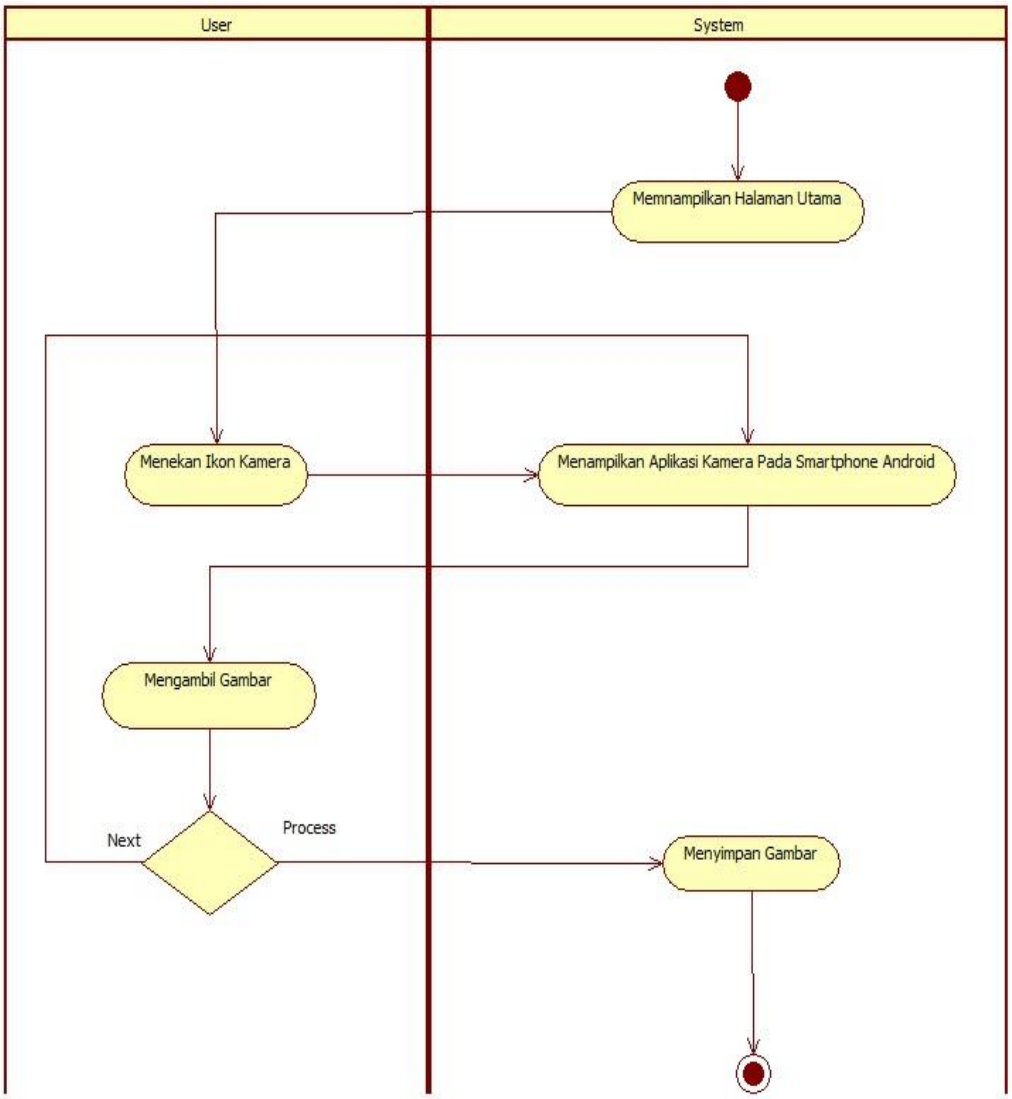


Sumber gambar :Sri Dharwiyanti dan romi Satria Wahono [19]

Gambar 2.15 *Collaboration* Diagram

2.13.1.5 *Activity* Diagram

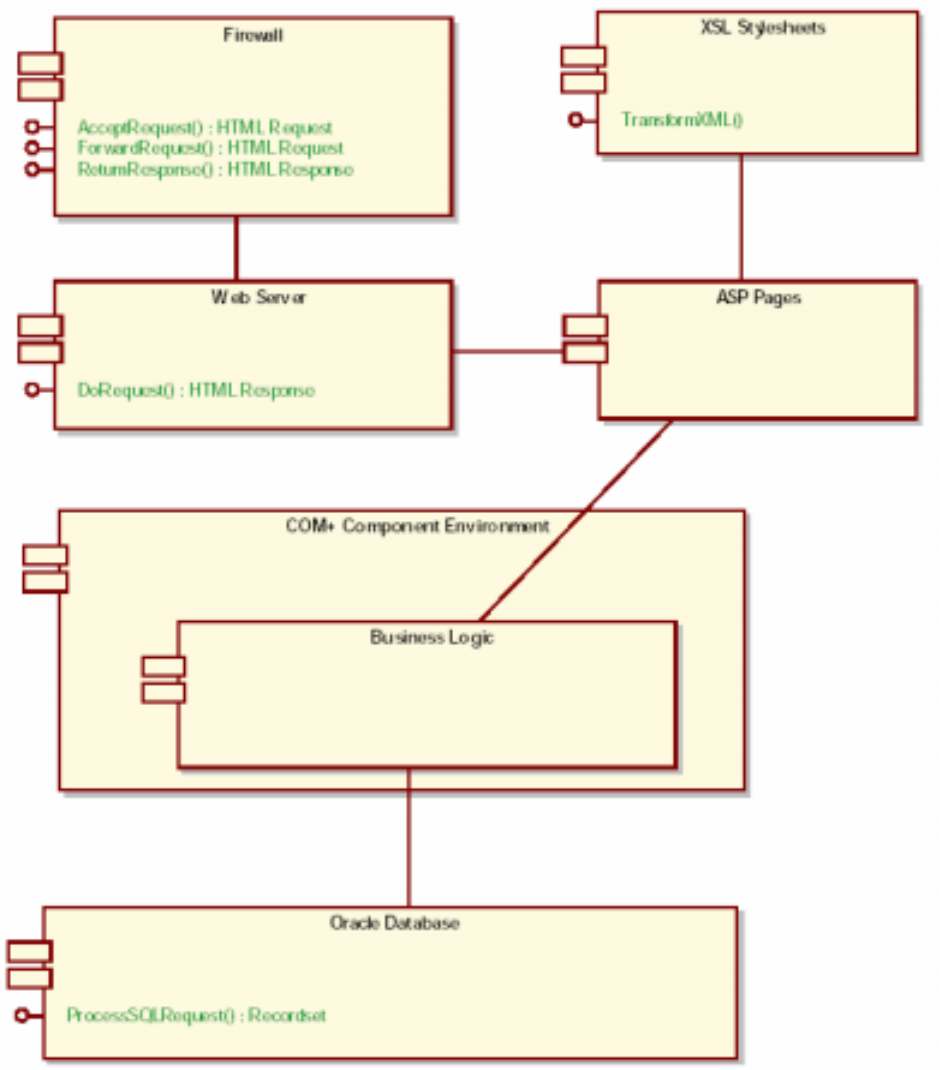
Sebuah *Activity Diagram* menunjukkan suatu alur kegiatan secara berurutan.*Activity Diagram* digunakan untuk mendiskripsikan kegiatan-kegiatan dalam sebuah operasi meskipun juga dapat digunakan untuk mendeskripsikan alur kegiatan yang lainnya seperti *use case* atau suatu interaksi.



Gambar 2.16 *Activity* Diagram

2.13.1.6 *Component* Diagram

Component Diagram menunjukkan struktur dan hubungan antar komponen *software* termasuk ketergantungan (*dependency*) diantara komponen-komponen tersebut. Komponen pada piranti lunak adalah berupa modul-modul yang berisikan *code*, baik *library* maupun *executable*. Umumnya komponen yang terbentuk dari beberapa *class* dan/atau *package*, atau juga dapat dari komponenkomponen yang lebih kecil.

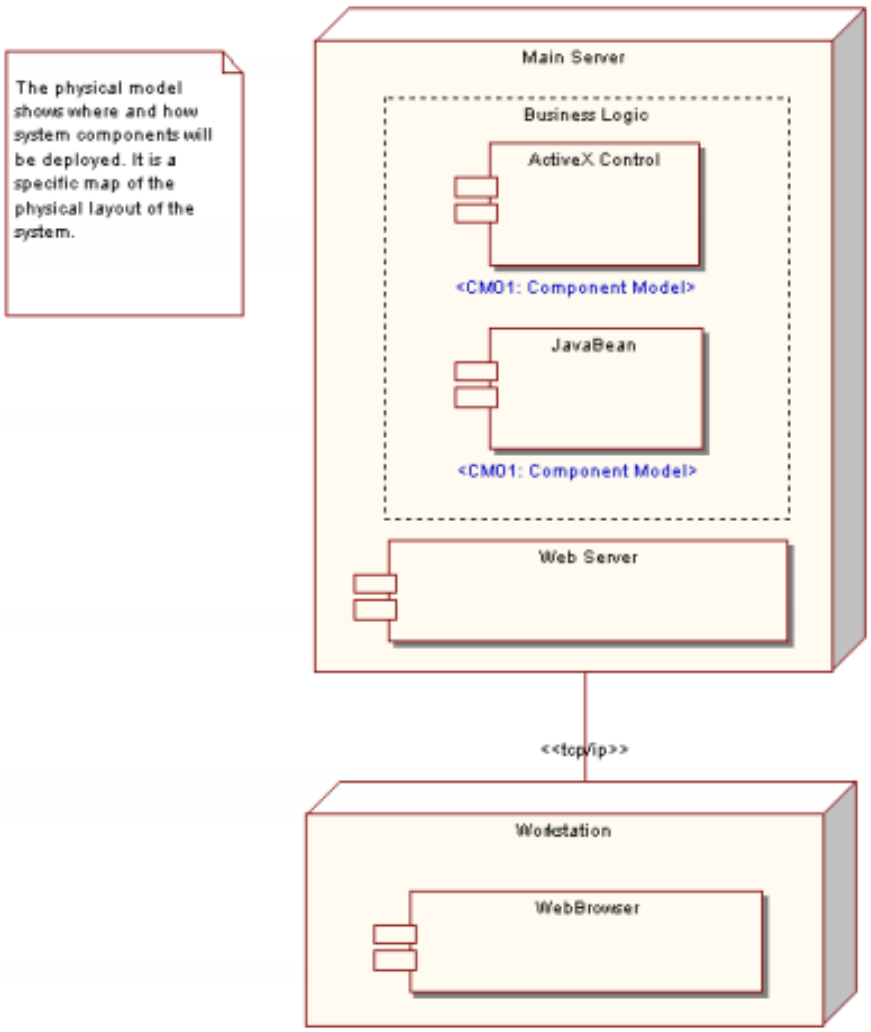


Sumber gambar :Sri Dharwiyanti dan romi Satria Wahono [19]

Gambar 2.17 *Component* Diagram

2.13.1.7 *Deployment* Diagram

Deployment Diagram menunjukkan arsitektur fisik pada *hardware* dan *software* pada suatu sistem yang dirancang. *Deployment diagram* juga dapat menunjukkan perngkat-perangkat dan *nodes* diantara hubungan yang dimilikinya antar komponen.



Sumber gambar :Sri Dharwiyanti dan romi Satria Wahono [19]

Gambar 2.18 *Deployment* Diagram

2.14 *Software Development Kit* (SDK)

Software Development Kit (SDK) adalah suatu kit atau library dari bahasa pemrograman untuk pengembangan atau pembangunan suatu perangkat lunak dan biasanya SDK terdiri dari kumpulan tools yang dibutuhkan. Misalnya bahasa pemrograman java, mempunyai SDK yang berisi suatu *library* yang dapat digunakan untuk membuat suatu aplikasi berbasis java.

2.15 *Java Development Kit* (JDK)

Java Development Kit (JDK) adalah sekumpulan perangkat lunak yang dapat kamu gunakan untuk mengembangkan perangkat lunak yang berbasis Java, Sedangkan JRE adalah sebuah implementasi dari *Java Virtual Machine* yang benar-benar digunakan untuk menjalankan program java.Biasanya, setiap JDK berisi satu atau lebih JRE dan berbagai alat pengembangan lain seperti sumber *kompiler java, bundling, debuggers, development libraries* dan lain sebagainya. Perbedaan JDK dengan SDK (*Software Development Kit*) yaitu JDK adalah sebuah SDK tetapi sebuah SDK tidak harus menjadi sebuah JDK.

2.16 Metode Pengujian *BlackBox*

Metode pengujian adalah cara atau teknik untuk menguji perangkat lunak, mempunyai mekanisme untuk menentukan data uji yang dapat menguji perangkat lunak secara lengkap dan mempunyai kemungkinan tinggi untuk menemukan kesalahan. Salah satunya yaitu metode pengujian black box.

Pengujian *black box* merupakan pendekatan komplementer dari teknik white box, karena pengujian black box diharapkan mampu mengungkap kelas kesalahan yang lebih luas dibandingkan teknik white box. Pengujian black box berfokus pada pengujian persyaratan fungsional perangkat lunak, untuk mendapatkan serangkaian kondisi input yang sesuai dengan persyaratan fungsional suatu program. Pengujian black box adalah pengujian aspek fundamental sistem tanpa memperhatikan struktur logika internal perangkat lunak. Metode ini digunakan untuk mengetahui apakah perangkat lunak berfungsi dengan benar [20].

Pengujian black box merupakan metode perancangan data uji yang didasarkan pada spesifikasi perangkat lunak. Data uji dibangkitkan, dieksekusi pada perangkat lunak dan kemudian keluaran dari perangkat lunak dicek apakah telah sesuai dengan yang diharapkan. Pengujian black box berusaha menemukan kesalahan dalam kategori [20] :

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.

Pengujian black box cenderung diaplikasikan selama tahap akhir pengujian. Pengujian black box harus dapat menjawab [20]:

1. Bagaimana validitas fungsional diuji
2. Kelas input apa yang akan membuat kasus pengujian menjadi lebih baik
3. Apakah system akan sangat sensitive terhadap harga input tertentu
4. Bagaimana batasan dari suatu data diisolasi
5. Kecepatan data apa dan volume data apa yang akan ditoleransi oleh system
6. Apa pengaruh kombinasi tertentu dari data terhadap system operasi.

2.17 Skala Likert

Skala likert digunakan untuk mengukur sikap, pendapat dan persepsi seseorang atau sekelompok orang tentang kejadian atau gejala sosial. Dengan menggunakan skala likert, maka variabel dijabarkan menurut urutan variabel, sub variabel, indikator, deskriptor. Dan deskriptor ini dapat dijadikan titik tolak untuk membuat butir instrumen berupa pernyataan atau pertanyaan yang perlu dijawab oleh responden.

Item item likert menyediakan repon dengan kategori yang berjenjang. Biasanya banyaknya jenjang adalah lima, yaitu sangat setuju, setuju, ragu ragu, tidak setuju dan sangat tidak setuju. Setiap kategori respon selanjutnya diberi skor. Untuk item positif skor terbesarnya adalah lima, skor terendah adalah satu dan sebaliknya jika item negatif. Untuk menghitung total skor tiap responden adalah

dengan cara menjumlahkan skor skor item yang diperoleh reponden. Oleh karena itu, prosedur penskalaan likert sering disebut sebagai Likert's Summeted Rating [21].

Langkah langkah menyusunnya adalah sebagai berikut [21]:

1. Tentukan secara tegas sikap terhadap topik apa yang akan diukur.
2. Tentukan secara tegas sub varibel/dimensi yang menyusun sikap tersebut, kognitif, afektif dan konatif.
3. Susun pernyataan/pertanyaan atau item yang merupakan alat pengukur dimensi yang menyusun sikap yang akan diukur sesuai dengan indikator.
4. Setiap item diberi respon yang sifatnya tertutup.
5. Untuk setiap respon, jawaban diberi skor berdasarkan kriteria sebagai berikut: apabila item positif maka angka terbesar diletakkan pada respon sangat setuju, sedangkan bila item negatif maka angka terbesar diletakkan pada respon sangat tidak setuju.
6. Untuk mengetahui posisi setiap responden tentang suatu variabel, tentukan skor maksimal dan skor minimal yang mungkin dicapai oleh responden.

Langkah langkah untuk mengetahui bagaimana sikap tiap responden terhadap kualitas produk adalah [21]:

1. Menentukan skor maksimal, yaitu skor jawaban terbesar dikali banyak item.
2. Menentukan skor minimal, yaitu skor jawaban terkecil dikali banyak item.
3. Menentukan nilai median, yaitu hasil penjumlahan skor maksimal dengan skor minimal dibagi dua.
4. Menentukan nilai kuartil 1, yaitu hasil penjumlahan skor minimal dengan median dibagi dua.
5. Menentukan nilai kuartil 3, yaitu hasil penjumlahan skor maksimal dengan median dibagi dua.
6. Membuat skala yang menggambarkan skor minimal, nilai kuartil 1, median, nilai kuartil 3 dan skor maksimal.
7. Mencari batas batas skor untuk masing masing kategori sikap.
8. Membuat tabel distribusi frekuensi sikap tiap responden terhadap kualitas produk.

Langkah langkah untuk mengetahui bagaimana sikap responden secara keseluruhan terhadap kualitas produk adalah [21]:

1. Menentukan skor maksimal, yaitu skor maksimal yang diperoleh tiap responden dikali banyaknya responden.
2. Menentukan skor minimal, yaitu skor minimal yang diperoleh tiap responden dikali banyaknya responden.
3. Menentukan nilai median, yaitu hasil penjumlahan total skor maksimal dengan total skor minimal dibagi dua.
4. Menentukan nilai kuartil 1, yaitu hasil penjumlahan total skor minimal dengan median dibagi dua.
5. Menentukan nilai kiartil 3, yaitu hasil penjumlahan total skor maksimal dengan median dibagi dua.
6. Membuat skala yang menggambarkan total skor minimal, nilai kuartil 1, median, kuartil 3 dan total skor maksimal.
7. Mencari batas batas skor untuk masing masing kategori sikap.
8. Menentukan skor total yang diperoleh seluruh responden.
9. Mengintrepretasikan skor total responden dengan skala pada point.

2.18 Definisi Kualitas Perangkat Lunak

Menurut definisi dalam Steve McConnell's Code Complete membagi perangkat lunak ke dalam dua hal yaitu internal dan external quality characteristics. Karakteristik kualitas eksternal merupakan bagian-bagian dari suatu produk yang berhubungan dengan para pemakainya, sedangkan karakteristik kualitas internal tidak secara langsung berhubungan dengan pemakai. Software Quality didefinisikan sebagai kesesuaian yang diharapkan pada semua software yang dibangun dalam hal fungsi software yang diutamakan dan unjuk kerja software, standar pembangunan software yang terdokumentasi dan karakteristik yang ditunjukkan oleh software [22].

Definisi ini menekankan pada 3 hal yaitu:

1. Kebutuhan software adalah fondasi ukuran kualitas software, jika software Tidak sesuai dengan kebutuhan yang ditentukan maka kualitas pun kurang.

2. Jika menggunakan suatu standar untuk pembangunan software maka jika software tidak memenuhi standar tersebut maka dianggap kurang berkualitas.
3. Seringkali ada kualitas yang secara langsung diutarakan (tersirat) seperti kemudahan penggunaan dan pemeliharaan yang baik. Kualitas software dipertanyakan jika tidak memenuhi kebutuhan ini.

Hal lain yang perlu diperhatikan dalam kualitas perangkat lunak adalah quality assurarance (QA) yang merupakan activity of providing revidence. Selain itu harus adanya software quality management (SQM). Tujuan dari SQM adalah untuk mengembangkan suatu pemahaman kuantitatif dari kualitas proyek produk perangkat lunak dan mencapai tujuan spesifik kualitasnya. Menurut IEEE, kualitas perangkat lunak didefinisikan sebagai derajat dari sebuah sistem, komponen atau proses yang memenuhi kebutuhan-kebutuhan yang dispesifikasikan dan kebutuhan pengguna atau harapan pengguna.

Menurut Pressman, kualitas perangkat lunak adalah kesesuaian antara kebutuhan fungsional dan kebutuhan performansi yang terdefinisi, standar pengembangan yang terdokumentasi secara ekplisit dan karakteristik-karakteristik yang tersirat yang diharapkan dari perangkat lunak yang dikembangkan secara profesional.

2.19 Model Metriks Kualitas Perangkat Lunak

Dalam keilmuan rekayasa perangkat lunak terdapat beberapa model kualitas perangkat lunak. Pada setiap model ini terdapat beberapa faktor yang menjadi poin-poin utama dalam penilaian kualitas sebuah perangkat lunak. Berikut adalah penjelasan tentang model kualitas perangkat lunak yang akan diterapkan. Model ISO-9126 dikenalkan pertama kali pada tahun 1991 sebagai standarisasi kualitas produk perangkat lunak. Standarisasi ini dibuat karena banyaknya model kualitas yang ditawarkan sebagai faktor kualitas perangkat lunak [23].

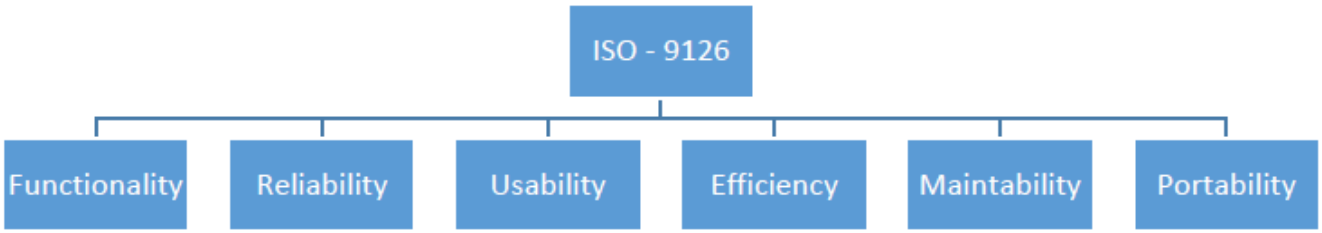
Dalam dokumen pertama model ISO-9126 terdiri dari beberapa bagian model kualitas untuk kualitas sebuah produk perangkat lunak, di antaranya:

1. Quality model
2. External quality model

3. Internal quality model
4. Quality in use model

2.19.1 Model Kualitas Eksternal

Bagian pertama dari kualitas model tersebut menentukan 6 karakteristik yang mereka bagi kedalam 21 sub karakteristik untuk kualitas internal dan kualitas eksternal. Berikut ini merupakan pengertian dari masing-masing faktor kualitas dan sub pada ISO-9126.



Gambar 2.19 Faktor ISO 9126

1. Functionality

Faktor ini berhubungan dengan seberapa jauh sebuah perangkat lunak dapat memenuhi kebutuhan penggunanya.

a. Suitability

Sub-faktor suitability berhubungan dengan tingkat kemampuan dan kelayakan dari sebuah perangkat lunak untuk dapat menyediakan fungsionalitas untuk kebutuhan yang spesifik.

Tabel 2.3 *Suitability Metriks*

Nama Metriks	Pengukuran	Intrepretasi nilai
Functional adequacy	X = 1-A/B A = jumlah fungsi yang terdapat masalah dalam evaluasi B = Jumlah fungsi yang di evaluasi	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Functional implementation completeness	X = 1-A/B A = jumlah fungsi yang hilang terdeteksi dalam evaluasi	X = A/T A = jumlah kesalahan yang dibuat oleh pengguna T = jumlah tugas dicoba

Nama Metriks	Pengukuran	Intrepretasi nilai
	B = Jumlah fungsi yang dijelaskan dalam spesifikasi kebutuhan	
Functional implementation coverage	X = 1-A/B A = jumlah yang dilakasanakan dengan benar atau fungsi yang hilang dalam evaluasi B = jumlah fungsi yang dijelaskan dalam	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Functional specification stability (volatility)	X = 1-A/B A = Jumlah fungsi perubahan setelah memasuki operasi dimulai B = Jumlah fungsi yang dijelaskan dalam spesifikasi kebutuhan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

b. Accuracy

Accuracy berhubungan dengan kemampuan sebuah perangkat lunak untuk dapat memberikan hasil yang benar dan sesuai dengan fungsi yang dijalankan.

Tabel 2.4 *Accuracy*

Nama Metriks	Pengukuran	Intrepretasi nilai
Accuracy to expectation	X = A / T A = jumlah kasus yang dihadapi oleh pengguna dengan perbedaan terhadap hasil yang diharapkan T = Waktu operasi	0 <= X Semakin dekat ke 0 semakin baik
Computational Accuracy	X = A/T A = Jumlah perhitungan akurasi yang dihadapi oleh pengguna T = Waktu operasi	0 <= X Semakin dekat ke 0 semakin baik
Precision	X = A/T A = Jumlah hasil yang dihadapi oleh pengguna dengan tingkat presisi yang berbeda dari yang	0 <= X Semakin dekat ke 0 semakin baik

Nama Metriks	Pengukuran	Intrepretasi nilai
	dibutuhkan waktu operasi	

c. *Interoperability*

Sub-faktor *interoperability* menggambarkan tentang kemampuan sebuah perangkat lunak untuk dapat berinteraksi dengan sistem yang lain atau dengan sistem tertentu.

Tabel 2.5 *Interoperability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Data exchangeability (Data format based)	X = A/B A = jumlah format yang data yang disetujui untuk ditukar dengan perangkat lunak lain atau sistem pengujian bursa data B = Total jumlah format data yang akan diperlukan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Data exchangeability (User's success attempt based)	X = 1-A/B A = Jumlah kasus dimana pengguna gagal untuk pertukaran data dengan perangkat lunak atau sistem lain B = Jumlah kasus di mana pengguna upaya	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Data exchangeability (User's success attempt based)	X = 1-A/B A = Jumlah kasus dimana pengguna gagal untuk pertukaran data dengan perangkat lunak atau sistem lain B = Jumlah kasus di mana pengguna upaya untuk pertukaran data	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

d. Security

Sub-faktor ini berhubungan dengan keamanan yang dimiliki oleh sebuah perangkat lunak. Keamanan yang dimaksud dapat berupa pemberian hak akses kepada penggunanya.

Tabel 2.6 *Security*

Nama Metriks	**Pengukuran**	**Intrepretasi nilai**
Access auditability	X = A/B A = Jumlah "pengguna akses ke sistem data" tercatat dalam database sejarah akses B = Jumlah "pengguna akses ke sistem dan data" dilakukan selama evaluasi	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Access controllability	X = A/B A = Jumlah yang terdeteksi berbagai jenis operasi ilegal B = Jumlah jenis operasi ilegal seperti dalam spesifikasi	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Data corruption prevention	X = 1 – A/N A = Jumlah kali bahwa peristiwa korupsi data terbesar yang terjadi N = Jumlah uji coba menyebabkan acara korupsi data	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

e. Functionality Compliance

Functionality compliance menggambarkan tentang kesesuaian antara fungsionalitas yang ada pada sebuah perangkat lunak dengan aturan standar yang berlaku pada perangkat lunak lainnya yang sejenis.

Tabel 2.7 *Funcionality Compliance*

Nama Metriks	Pengukuran	Intrepretasi nilai
Functional compliance	X = 1 - A/B A = Jumlah item kepatuhan fungsi tertentu yang belum dilakasankan selama pengujian B = Total jumlah item kepatuhan fungsi tertentu	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Interface standard compliance	X = A/B A = Jumlah *interface* diterapkan dengan benar sebagaimana ditentukan B = jumlah *interface* yang membutuhkan pemenuhan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

2. Reliability

Faktor *reliability* berhubungan dengan keandalan atau seberapa baik tingkat pelayanan sebuah perangkat lunak apabila digunakan dalam kondisi tertentu.

a. Maturity

Maturity berhubungan dengan kelayakan sebuah perangkat lunak dalam menangani kegagalan atau kesalahan yang terdapat didalamnya.

Tabel 2.8 *Maturity*

Nama Metriks	Pengukuran	Intrepretasi nilai
Estimated latent fault density	X = {ABS(A1 – A2)}/B (X:perkiraan yang kepadatan kesalahan laten) ABS ()= Absolut A1 = Jumlah kesalahan laten memprediksi dalam produk perangkat lunak A2 = Jumlah sebenarnya terprediksi kesalahan. B = Produk Ukuran	0 <= X Hal ini tergantung pada tahap pengujian. Pada tahap selanjutnya lebih kecil lebih baik.
Failure density against test cases	X = A1/A2 A1 = Jumlah kegagalan terdeteksi A2 = Jumlah kasus uji yang dilakukan	0 <= X Hal ini tergantung pada tahap pengujian. Pada tahap selanjutnya lebih kecil lebih baik.

Nama Metriks	Pengukuran	Intrepretasi nilai
Fault density	X = A/B A = Jumlah kesalahan yang terdeteksi B = Ukuran Produk	0 <= X Hal ini tergantung pada tahap pengujian. Pada tahap selanjutnya lebih kecil lebih baik.
Fault removal	a).X = A1/A2 A1= Jumlah kesalahan dikoreksi A2= Jumlah Sebenarnya terdeteksi kesalahan. b).Y = A1/A3 A= Jumlah kesalahan	0 <= X <= 1 Semakin dekat 1.0 semakin lebih baik sebagai kesalahan yang lebih sedikit 0 <= Y
Mean time between failures (MTBF)	a).X = T1/A b).Y = T2 / A T1=Waktu operasi T2=Jumlah interval waktu yang berturut-turut mengalami kegagalan A=Jumlah sebenarnya terdeteksi kegagalan,(kegagalan terjadi selama waktu operasi)	0 <= X, Y Semakin lama semakin baik, seperti waktu yang lebih lama dapat diharapkan anatara kegagalan
Test coverage (Specified operation scenario testing coverage)	X = A / B A=Jumlah kasus uji benerbener dilakukan mewakili skenario operasi selama pengujian. B=Jumlah kasus uji yang akan dilakukan untuk menutupi kebutuhan	0 <= X <= 1 Semakin mendekati 1.0 semakin lebih baik
Test maturity	X = A / B A=Jumlah kasus uji lulus selama pengujian atau operasi B=Jumlah kasus uji yang akan dilakukan untuk menutupi kebutuhan	0 <= X <= 1 Semakin mendekati 1.0 semakin lebih baik

b. Fault Tolerance

Fault tolerance merupakan sub-faktor *reliability* yang menggambarkan tentang seberapa jauh sebuah perangkat lunak dapat memberikan toleransi kegagalan atas kesalahan yang terjadi saat digunakan pada kondisi tertentu.

Tabel 2.9 *Fault Tolerance*

Nama Metriks	Pengukuran	Intrepretasi nilai
Breakdown avoidance	X = 1-A/B A = Jumlah kerusakan B = Jumlah kegagalan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Failure avoidance	X = A/B A = Jumlah kejadian kegagalan kritis dan serius terhadap kasus uji dari pola kesalahan B=Jumlah kasus uji dieksekusi dari pola kesalahan.	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Incorrect operation avoidance	X = A/B A = Jumlah kejadian kegagalan kritis dan serius terhadap kasus uji dari pola kesalahan B=Jumlah kasus uji dieksekusi dari pola kesalahan.	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

c. Recoverability

Sub-faktor ini berhubungan dengan kemampuan sebuah perangkat lunak untuk dapat membangun kembali kinerja dan memulihkan data baik secara langsung maupun secara tidak langsung.

Tabel 2.10 *Recoverability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Availability	a).X={To/(To+Tr)} To=Waktu untuk operasi Tr=Waktu untuk memperbaiki A1=Total kasus yang tersedia pengguna software pengguna	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik, karena pengguna dapat menggunakan perangkat lunak untuk lebih banyak waktu

Nama Metriks	Pengukuran	Intrepretasi nilai
	berhasil ketika pengguna mencoba untuk menggunakan b).Y=A1/A2 A2=Jumlah kasus upaya pengguna untuk menggunakan perangkat lunak selama waktu pengamatan	0 <= Y <= 1 Semakin dekat ke 1.0 semakin baik
Mean down time	X=T/N T= Jumlah down time N=Jumlah kerusakan yang diamati kasus terburuk atau distribusi down time harus diukur	0 < X Semakin dekat ke 1.0 semakin baik
Mean recovery time	X=Sum(T)/N T=Waktu untuk pemulihan sistem perangkat lunak pada setiap kesempatan B=Jumlah kasus yang diamati sistem software masuk recovery	0 < X Semakin dekat ke 1.0 semakin baik
Restartability	X=A/B A=Jumlah restart yang bertemu untuk waktu yang dibutuhkan selama pengujian atau operasi pengguna B=Total jumlah restart selama pengujian atau operasi pengguna	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik, karena pengguna dapat restart dengan mudah.
Restorability	X=A/B A=Jumlah kasus restorasi berhasil dilakukan B=Jumlah kasus restorasi diuji sesuai kebutuhan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik, karena produk lebih mampu untuk mengembangkan dalam kasus tertentu.
Restore effectiveness	X=A/B A=Jumlah kasus berhasil dipublikasikan memenuhi terget mengembalikan waktu	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik, karena proses restorasi di produk lebih efektif.

Nama Metriks	Pengukuran	Intrepretasi nilai
	B=Jumlah kasus yang dilakukan	

d. Reability Compliance

Reliability compliance berhubungan dengan kesesuaian antara tingkat pelayanan yang diberikan oleh perangkat lunak dengan aturan standar yang digunakan pada setiap perangkat lunak yang memiliki tipe yang sama.

Tabel 2.11 *Reability Compliance*

Nama Metriks	Pengukuran	Intrepretasi nilai
Reliability compliance	X = 1 - A / B A = Jumlah item kepuasan keandalan ditentukan yang belum dilaksanakan selama pengujian B = Total jumlah item kepatuhan keandalan ditentukan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

3. Usability

Usability berhubungan dengan seberapa baik perangkat lunak dapat dipahami, dipelajari, dan digunakan.

a. Understandability

Sub-faktor ini berhubungan dengan seberapa jauh sebuah perangkat lunak dapat dipahami oleh pengguna baik dari secara konsep logis dan penerapan penggunaan perangkat lunak tersebut.

Tabel 2.12 *Understandability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Completeness of description	X = A/B A =Jumlah fungsi (jenis fungsi) dipahami B = Total jumlah (jenis fungsi)	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Demonstration accessibility	X = A/B	0 <= X <= 1

Nama Metriks	Pengukuran	Intrepretasi nilai
	A = Jumlah demonstrasi / tutorial yang pengguna berhasil mengaksesnya B =Jumlah demonstrasi /tutorial yang tersedia	Semakin dekat ke 1.0 semakin baik.
Demonstration accessibility in use	X = A/B A =Jumlah kasus di mana pengguna berhasil melihat demonstrasi ketika pengguna mencoba untuk meliha demonstrasi B =Jumlah kasus di mana pengguna mencoba untuk melihat demonstrasi selama pengamatan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik.
Demonstration effectiveness	X = A/B A =Jumlah fungsi yang di operasikan berhasil B = Jumlah demonstrasi / tutorial yang di akses	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik.
Evident functions	X = A/B A =Jumlah fungsi (jenis fungsi) yang didentifikasi oleh pengguna B =Total jumlah fungsi	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik.
Function understand-ability	X = A/B A =Jumlah fungsi antarmuka yang tujuanya dengan benar dijalankan oleh pengguna B =Jumlah fungsi yang tersedia dari antarmuka	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik.
Understandable input and output	X = A/B A =Jumlah input dan output data item yang pengguna berhasil mengerti B =Jumlah item data input dan output yang tersedia dari antarmuka.	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik.

b. *Learnability*

Learnability menggambarkan tentang sebuah perangkat lunak dapat dipelajari dengan baik oleh penggunanya.

Tabel 2.13 *Learnability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Ease of function learning	T= Berarti waktu yang dibutuhkan untuk pembelajaran menggunakan fungsi dengan benar	0 < T Semakin pendek semakin baik
Ease of learning to perform a task in use	T = Jumlah waktu operasi pengguna sampai waktu dicapai untuk melakukan tugas tertentu dalam waktu singkat	0 < T Semakin pendek semakin baik
Effectiveness of the user documentation and/or help system	X = A/B A = Jumlah tugas yang berhasil diselesaikan setelah mengakses bantuan online atau dokumentasi B =Jumlah tugas diuji	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

c. *Operability*

Operability berhubungan dengan seberapa jauh perangkat lunak dapat dioperasikan oleh penggunanya.

Tabel 2.14 *Operability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Operational consistency in use	a).X = 1-A/B A = Jumlah pesan atau fungsi uang menemukan data yang tidak konsisten dengan ekspektasi pengguna B = Jumlah pesan atau fungsi b).Y=N/UOT N= Jumlah pesan atau fungsi yang menemukan data yang tidak konsisten	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik 0 <= Y Semakin dekat ke 0.0 semakin baik

Nama Metriks	Pengukuran	Intrepretasi nilai
	dengan ekspektasi pengguna UOT = Pengguna waktu operasi selama sistem berjalan.	

d. Attractiveness

Sub-faktor ini menggambarkan tentang bagaimana sebuah perangkat lunak dapat menarik perhatian bagi penggunanya.

Tabel 2.15 *Attractiveness*

Nama Metriks	Pengukuran	Intrepretasi nilai
Attractive interaction	Kuesioner untuk menilai daya tarik dari antarmuka pengguan, setelah pengguna mengunakan.	Tergantung pada metode kuesioner scoringnya
Interface appearance customisability	X = A/B A = Jumlah elemen antarmuka disesuaikan dalam penampilan untuk kepuasan pengguna B = Jumlah antarmuka pengguna yang ingin menyesuaikan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

e. Usability Compliance

Sub-faktor ini berhubungan dengan kesesuaian antara kegunaan perangkat lunak dengan standar yang digunakan oleh perangkat lunak sejenis lainnya.

Tabel 2.16 *Usability Compliance*

Nama Metriks	Pengukuran	Intrepretasi nilai
Usability compliance	X = 1-A/B A = Jumlah item kepuasan kegunaan tertentu yang belum dilaksanakan selama pengujian B = Total jumlah item kepuasan kegunaan tertentu	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

4. Efficiency

Faktor efficiency berhubungan dengan efisiensi dari seberapa besar sumber daya yang digunakan oleh sebuah perangkat lunak.

a. Time Behaviour

Time behaviour berhubungan dengan waktu yang dibutuhkan oleh sebuah perangkat lunak untuk menjalankan suatu fungsi atau tujuan tertentu.

Tabel 2.17 *Time Behaviour*

Nama Metriks	**Pengukuran**	**Intrepretasi nilai**
Response time	T = (Waktu untuk mendapatkan hasil)-(saat masuk perintah selesai)	0 < T Semakin cepat semakin baik.
Response time (Mean time to response)	Tmean = (Ti) Σ/ N, (untuk i = 1 sampai N) TXmean = diperlukan waktu respon rata-rata Ti = waktu respon untuk evaluasi ke-i (shot) N = jumlah evaluasi (tembakan sampel)	0 <= X Semakin dekat ke 1.0 dan kurang dari 1.0 lebih baik
Response time (Worst case response time ratio)	X = Tmax / Rmax Tmax = MAX (Ti) (untuk i = 1 sampai N) Rmax = diperlukan waktu respons maksimum MAX (Ti) = waktu respons maksimum antara evaluasi N = jumlah evaluasi (tembakan sampel) Ti = waktu respon untuk evaluasi ke-i (shot)	0 <= X Semakin ke 1 dan kurang dari 1 akan lebih baik

b. Resource Utilization

Sub-faktor ini menggambarkan tentang jumlah sumber daya yang digunakan oleh perangkat lunak untuk melakukan fungsi atau tujuan tertentu.

Tabel 2.18 Resource

Nama Metriks	**Pengukuran**	**Intrepretasi nilai**
I/O devices utilisation	X = A/B A = Waktu I/O perangkat diduduki B = Ditentukan waktu yang dirancang untuk menempati	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
I/O loading limits	X = Amax/Rmax Amax = MAX (Ai), (untuk i = 1 sampai N) Rmax = maksimum yang diperlukan I / O pesan MAX (Ai) = Jumlah maksimum pesan I / O dari 1 untuk evaluasi ke-i. N = jumlah evaluasi.	0 <= X Semakin kecil semakin baik
I/O related errors	X = A/T A= Jumlah pesan peringatan atau kegagalan sistem T= Penggunaan waktu operasi selam pengamatan pengguna	0 <= X Semakin kecil semakin baik
Mean I/O fulfillment ratio	X = Amean/Rmean Amean = Σ (Ai) / N Rmean = diperlukan rata jumlah pesan I /O Ai = jumlah pesan kesalahan I / O untuk i th evaluasi N = jumlah evaluasi	0 <= X Semakin kecil semakin baik
User waiting time of I/O devices utilisation	T= Waktu yang dihabiskan untu menugguakhir dari perangkat I/O operasi	0 < T Semakin kecil semakin baik

c. *Efficiency Compliance*

Efficiency compliance berhubungan dengan kesesuaian antara sumber daya yang digunakan oleh sebuah perangkat lunak dengan standar penggunaan sumber daya pada perangkat lunak lainnya yang sejenis.

Tabel 2.19 *Efficiency Compliance*

Nama Metriks	Pengukuran	Intrepretasi nilai
Efficiency Compliance	X = 1-A/B (X:Rasio item kepatuhan berkaitan denga efisiensi) A = Jumlah efisiensi item kepatuahan ditentukan yang belum dilaksanakan selama pengujian B = Total jumlah item kepatuhan efisiensi dintentukan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

5. *Maintability*

Maintainability menggambarkan tentang pemeliharaan sebuah perangkat lunak, seberapa baik perangkat lunak tersebut dapat dipertahankan.

a. *Analyzability*

Analyzability berhubungan dengan seberapa jauh sebuah perangkat lunak dapat di analisis, hal ini diperlukan untuk analisis kekurangan atau penyebab kegagalan agar dapat diketahui bagian mana yang perlu dimodifikasi.

Tabel 2.20 *Analyzability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Audit trail capability	X = A/B A = Jumlah data yang sebenarnya direkam selama operasi B = Jumlah data direncanakan akan cukup direkam untuk memantau status perangkat lunak selama operasi	0 <= X Semakin dekat ke 1.0 semakin baik
Diagnostic function support	X = 1-A/B A = Jumlah kegagalan yang	0 <= X <= 1 Semakin dekat ke 1.0

Nama Metriks	Pengukuran	Intrepretasi nilai
	pemelihara dapat diagnos e (menggunakan fungsi diagnostik) untuk memahami penyebab - kapal hubungan efek B = Total jumlah kegagalan terdaftar	

b. Changeability

Changeability berhubungan dengan seberapa baik perangkat lunak dapat diubah, upaya ini diperlukan untuk modifikasi, penghapusan kesalahan atau perubahan lingkungan.

Tabel 2.21 *Changeability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Change cycle efficiency	Average Time : Tav = Sum(Tu)/N Tu= Trc – Tsn Tsn = Waktu di mana pengguna selesai untuk mengirim permintaan untuk pemeliharaan untuk pemasok dengan laporan masalah Trc = Waktu di mana pengguna menerima rilis versi revisi (atau laporan status) N = Jumlah versi revisi	0 < Tav Semakin pendek lebih baik kecuali dari jumlah versi revisi itu besar
Change implementation elapse time	Average Time : Tav = Sum(Tm) / N Tm=T out – T in T out = Waktu di mana penyebab kegagalan dihapus dengan mengubah perangkat lunak (atau status dilaporkan kembali ke pengguna) Tin = Waktu di mana penyebab kegagalan yang menemukan	0 < Tav Semakin pendek lebih baik kecuali dari jumlah versi revisi itu besar

Nama Metriks	Pengukuran	Intrepretasi nilai
	N = Jumlah kegagalan terdaftar dan dihapus	
Modification complexity	T = Sum (A / B) / N A = Kerja dihabiskan untuk mengubah B = Ukuran perubahan software N = Jumlah Perubahan	0 < T Semakin dekat ke 1.0 semakin baik
Parameterised modifiability	X=1- A / B A = Jumlah kasus yang pemelihara gagal untuk mengubah perangkat lunak dengan menggunakan parameter B = Jumlah kasus yang maintainer mencoba untuk mengubah perangkat lunak dengan menggunakan parameter	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Software change control capability	X= A / B A = Jumlah data log perubahan benar-benar dicatat B = Jumlah data perubahan log rencananya akan direkam cukup untuk melacak perubahan perangkat lunak	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

c. *Stability*

Sub-faktor ini berhubungan dengan stabilitas dari sebuah perangkat lunak yang memungkinkan untuk menyimpulkan resiko efek tak terduga yang disebabkan oleh modifikasi

Tabel 2.22 *Stability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Change Success Ratio	X = Na/Ta Y={(Na/Ta)/(Nb/Nb)} Na = Jumlah kasus yang pengguna pertemuan kegagalan selama operasi setelah software berubah	0 <= X,Y Semakin dekat ke 1.0 semakin baik

Nama Metriks	Pengukuran	Intrepretasi nilai
	Nb = Jumlah kasus yang pengguna pertemuan kegagalan selama operasi sebelum perangkat lunak berubah Waktu ta = Operasi selama periode observasi ditentukan setelah perangkat lunak berubah Tb waktu = Operasi selama periode pengamatan ditentukan sebelum perangkat lunak berubah	
Modification impact localisation (Emerging failure after change)	X=A/N A = Jumlah kegagalan muncul setelah kegagalan diselesaikan oleh perubahan selama periode tertentu N = Jumlah kegagalan diselesaikan	0<=X Semakin dekat ke 0 semakin baik

d. Testability

Testability menggambarkan tentang bagaimana perangkat lunak dapat diuji, hal ini untuk menyimpulkan tentang upaya yang diperlukan untuk memvalidasi perangkat lunak dan cakupan pengujian.

Tabel 2.23 *Testability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Availability of built-in test function	X = A/B A = Jumlah kasus dimana pengelola dapat mengunakan sesuia buit-in fungsi tes B =Jumlah kasus peluang uji	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Re-test efficiency	X = Sum(T)/N T = Waktu yang dihabiskan untuk menguji untuk memastikan apakah	0 <= X Semakin dekat ke 1.0 semakin baik

Nama Metriks	Pengukuran	Intrepretasi nilai
	melaporkan kegagalan itu diselesaikan atau tidak N = Jumlah kegagalan diselesaikan	
Test restartability	X = A/B A = Jumlah kasus di mana pengelola dapat menghentikan sebentar dan restart melaksanakan uji coba pada titik-titik yang diinginkan untuk memeriksa langkah demi langkah B = Jumlah kasus jeda melaksanakan uji coba	0 <= X Semakin dekat ke 1.0 semakin baik

e. Maintability Compliance

Hal ini berhubungan dengan kesesuaian antara pemeliharaan yang dilakukan terhadap perangkat lunak dengan standarisasi yang terdapat pada pemeliharaan perangkat lunak lainnya yang sejenis.

Tabel 2.24 *Maintability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Maintainability compliance	X = 1-A/B A = Jumlah item kepatuhan rawatan ditentukan yang belum dilaksanakan selama pengujian B = Total jumlah item kepatuhan rawatan ditentukan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

6. Portability

Portability menggambarkan tentang kemampuan sebuah perangkat lunak untuk dapat berpindah dari sebuah lingkungan atau sistem ke sistem lainnya.

a. *Adaptability*

Sub-faktor ini berhubungan dengan seberapa jauh sebuah perangkat lunak dapat beradaptasi dengan perubahan lingkungan atau sistem yang berbeda.

Tabel 2.25 *Adaptability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Adaptability of data structures	X = A/B A = Jumlah data yang beroperasi dan tetapi tidak diamati karena operasi tidak lengkap disebabkan oleh keterbatasan adaptasi B = Jumlah data yang diharapkan dapat beroperasi dalam lingkungan yang perangkat lunak disesuaikan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Hardware environmental adaptability	X = 1-A/B A = Jumlah fungsional operasi tugas tidak selesai atau tidak cukup menghasilkan untuk memenuhi selama combined pengujian operasi dengan hardware lingkungan B = Total jumlah fungsi yang diuji	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

b. *Installability*

Installability menggambarkan tentang seberapa baik perangkat lunak dapat digunakan dalam lingkungan atau sistem tertentu.

Tabel 2.26 *Installability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Ease of installation	X = A/B A = Jumlah kasus yang pengguna berhasil mengubah menginstal operasi untuk / kenyamanan nya B = Total jumlah kasus yang	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

Nama Metriks	Pengukuran	Intrepretasi nilai
	pengguna mencoba untuk mengubah menginstal operasi untuk / kenyamanan nya	
Ease of Setup Retry	X = 1-A/B A = Jumlah kasus di mana pengguna gagal di coba selama set - up operasi B = Total jumlah kasus di mana pengguna upaya untuk kembali mencoba setup selama set - up operasi	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

c. *Co-existence*

Co-existence berhubungan dengan bagaimana perangkat lunak dapat berdampingan dengan produk atau perangkat lunak lain pada suatu lingkungan atau sistem yang sama untuk mengetahui tentang dependensi, perilaku, atau efek samping yang ditimbulkan.

Tabel 2.27 *Co-existence*

Nama Metriks	Pengukuran	Intrepretasi nilai
Available coexistence	X = A/T A = Jumlah hambatan (kegagalan) tak terduga yang pengguna temukan selama operasi dengan software lain T = Waktu durasi bersamaan operasi perangkat lunak lain	0 <= X Semakin dekat ke 1.0 semakin baik

d. *Replaceability*

Replaceability berhubungan dengan bagaimana sebuah perangkat lunak dapat menggantikan perangkat lunak lain apakah ada kebergantungan kepada perangkat lunak lain saat perangkat lunak tersebut digunakan.

Tabel 2.28 *Replaceability*

Nama Metriks	Pengukuran	Intrepretasi nilai
Continued use of data	X = A/B A = jumlah data yang digunakan dalam perangkat lunak lain yang akan diganti dan menegaskan bahwa mereka mampu terus menerus digunakan B = jumlah data yang digunakan dalam perangkat lunak lain untuk diganti dan rencananya akan terus digunakan kembali	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
Function inclusiveness	X = A/B A = jumlah fungsi yang menghasilkan hasil yang sama seperti sebelumnya diproduksi dan di mana perubahan tidak akan en diperlukan B = jumlah fungsi diuji yang mirip dengan fungsi yang disediakan oleh perangkat lunak lain yang akan diganti	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik
User support functional consistency	X = 1-A1/A2 A = Jumlah fungsi baru yang ditemukan pengguna , konsisten dengan ekspektasi pengguna B = Jumlah fungsi baru	0 <= X Semakin dekat ke 1.0 semakin baik

e. Portability Compliance

Portability compliance berhubungan dengan kesesuaian antara perubahan yang dapat dilakukan oleh sebuah perangkat lunak dengan standarisasi portability yang terdapat pada perangkat lunak lain yang sejenis.

Tabel 2.29 *Portability Compliance*

Nama Metriks	Pengukuran	Intrepretasi nilai
Portability compliance	X = 1-A/B A = Jumlah portabilitas item kepatuhan ditentukan yang belum dilaksanakan selama pengujian B = Total jumlah portabilitas item kepatuhan ditentukan	0 <= X <= 1 Semakin dekat ke 1.0 semakin baik

2.19.2 Model Quality In Use

Model Quality In Use mempunyai 4 Faktor yaitu Effectiveness, Productivity, Safety dan Satisfaction. Masing masing faktor juga mempunyai beberapa metriks. Berikut penjelasannya :

1. *Effectiveness* (Efektivitas)

Effectiveness berhubungan dengan seberapa efektif suatu perangkat lunak terhadap penggunanya.

Tabel 2.30 *Effectiveness*

Nama Metriks	Pengukuran	Intrepretasi nilai
Effectiveness	*Task completion*	X = A/B A = jumlah tugas diselesaikan B = jumlah tugas yang dicoba
	Error Frequency	X = A/T A = jumlah kesalahan yang dibuat oleh pengguna T = jumlah tugas dicoba

2. Productivity

Productivity berhubungan dengan seberapa produktif kah perangkat lunak terhadap penggunanya.

Tabel 2.31 *Productivity*

Nama Metriks	Pengukuran	Intrepretasi nilai
Productivity	*Task time*	X = A/B A = jumlah tugas diselesaikan B = jumlah tugas yang dicoba

Nama Metriks	Pengukuran	Intrepretasi nilai
	Task efficiency	X = A/T A = jumlah kesalahan yang dibuat oleh pengguna T = jumlah tugas dicoba
	Economic Productivity	X = M1 / C M1 = efektivitas tugas C = total biaya tugas
	Productive proportion	X = Ta / Tb Ta = waktu produktif = waktu tugas – waktu bantu - waktu error - waktu pencarian Tb = waktu tugas
	Relative user efficiency	X = A / B A = efisiensi tugas pengguna biasa ini B = efisiensi tugas ahli pengguna

3. Safety

Safety (Keamanan) berhubungan dengan seberapa aman perangkat lunak yang dianalisis terhadap penggunanya.

Tabel 2.32 *Safety*

Nama Metriks	Pengukuran	Intrepretasi nilai
Safety	*Safety of People affected by use of the system*	X = 1-A / B A = jumlah orang menaruh di hazard B = Jumlah orang yang berpotensi terkena dampak oleh sistem
	Economic damage	X = 1-A / B A = jumlah kejadian kerusakan ekonomi B = Jumlah situasi penggunaan
	Software damage	X = 1-A / B A = jumlah kejadian perangkat lunak corrupt B = Jumlah situasi penggunaan

4. Satification

Satisfaction (Kepuasan) berhubungan dengan kepuasan pelanggan terhadap perangkat lunak yang digunakan.

Tabel 2.33 *Satification*

Nama Metriks	Pengukuran	Intrepretasi nilai
Satisfaction	*Discretionary usage*	X = A / B A = berapa kali fungsi perangkat lunak tertentu / aplikasi / sistem yang digunakan B = jumlah pengguna menggunakan
	Satisfaction questionnaire	X = A / n A = menanggapi pertanyaan n = jumlah tanggapan

2.20 Metode Pengukuran Kualitas Perangkat Lunak

Pengukuran kualitas perangkat lunak merupakan hal yang terpenting dari praktek rekayasa perangkat lunak yang baik. Serta pengukuran aktivitas manusia menambah nilai dan terlibat aktif dalam proses informasi. Pengukuran ini membantu untuk membuat karakteristik khusus dari proses dan produk informasi. Pengukuran meliputi evaluasi kuantitatif yang biasanya menggunakan metriks serta ukuran dapat menggunakan secara langsung menentukan pencapaian tujuan kualitas numerik.

Metriks Internal Dapat digunakan untuk membantu memprediksi apakah perangkat lunak akan memenuhi kebutuhan keandalan organisasi selama pengembangan sistem perangkat lunak. Maturity Metrik menunjukkan satu set atribut untuk menilai kematangan suatu software. Deteksi kesalahan adalah Sebuah ukuran seberapa banyak defects terdeteksi dalam produk perangkat lunak. Metrik ini dikumpulkan dengan menghitung jumlah terdeteksi bug yang ditemukan dalam review dan membandingkan angka itu dengan jumlah yang diperkirakan. dapat ditemukan dalam tahap pengujian statis. Metriks yang dihitung oleh rumus X = A / B , dimana A adalah jumlah absolut bug terdeteksi (dari laporan review) dan B adalah perkiraan jumlah expected. Nilai tinggi untuk metriks ini menyiratkan baik

tingkat kualitas produk. Penghapusan kesalahan adalah Sebuah ukuran seberapa banyak cacat yang ditemukan selama review dihapus (dikoreksi) selama desain dan implementasi tahap. Metrik dapat dihitung dengan rumus X = A / B. dimana A adalah jumlah bug tetap selama desain dan coding dan B adalah Jumlah yang ditemukan selama review. Semakin dekat nilai adalah 1, yang lebih baik. A Nilai penghapusan kesalahan orang akan berarti bahwa setiap cacat terdeteksi telah dihapus.

Metriks Eksternal ISO 9126 mempunyai beberapa metrik eksternal. Metriks eksternal didefinisikan dalam ISO 9126-2 dan diukur selama pengujian dinamis. Kategori metrik ini sesuai dengan yang internal yaitu *maturity, fault 38 tolerance, recoverability, and compliance*. Metriks Kematangan mengukur kebebasan sistem perangkat lunak dari kegagalan disebabkan oleh bug di sistem itu sendiri. Faktor eksternal ini lebih menilai dari segi program kepada pengguna sedangkan internal lebih menilai dari segi program tersebut. Perhitungan Metriks Pada metriks 9126 terdapat 6 faktor dan beberapa subfaktor yang akan digunakan sebagai perhitungan yang tentunya akan dilihat dari karakteristik perangkat lunak tersebut. Setiap faktor karakteristik mempunyai hitungan tersendiri yang sudah ditentukan oleh metriks 9126 [23].

Tabel 2.34 Subfaktor Metriks

Functionality Metrics	*Suitability metrics* *Accuracy metrics* *Interoperability metrics* *Security metrics* *Functionality compliance metrics*
Reliability Metrics	*Maturity metrics* *Fault Tolerance metrics* *Recoverability metrics* *Reliability compliance metrics*
Usability Metrics	*Understandability metrics* *Learnability metrics* *Operability metrics* *Attractiveness metrics* *Usability compliance metrics*
Efficiency Metrics	*Time Behaviour metrics* *Resource utilisation metrics* *Efficiency compliance metrics*

Maintainability Metrics	*Analysability metrics* *Changeability metrics* *Maintainability compliance metrics*
Portability Metrics	*Adaptability metrics* *Installbility metrics* *Co-existence metrics* *Repleaceability metrics* *Portability Compliance metrics*

2.21 Pengukuran Perangkat Lunak

Pengukuran perangkat lunak adalah faktor utama untuk mengetahui seberapa baik perangkat lunak yang akan dinilai berdasarkan metode yang digunakan. Pengukuran perangkat lunak dilakukan dengan cara mencari nilai kriteria perangkat lunak tersebut dan juga mencari bobot kepentingan berdasarkan metode yang akan digunakan. Penilaian keseluruhan dapat diperoleh dari hasil nilai kriteria dan juga nilai bobot. Perhitungan bobot dan nilai kriteria bisa bervariasi tergantung dengan metode apa yang digunakan.

Metode Pembobotan *Rank Order Centroid* (ROC) didasarkan pada tingkat kepentingan atau prioritas dari kriteria. Menurut Jeffreys dan Cockfield dalam Afiefah Rahma, teknik ROC memberikan bobot pada setiap kriteria sesuai dengan ranking yang dinilai berdasarkan tingkat prioritas. Biasanya dibentuk dengan pernyataan “Kriteria 1 lebih penting dari kriteria 2, yang lebih penting dari kriteria 3” dan seterusnya hingga kriteria ke n, ditulis . Untuk menentukan bobotnya, diberikan aturan yang sama yaitu dimana merupakan bobot untuk kriteria [23]. Atau dapat dijelaskan sebagai berikut:

$$\text{Jika} \quad Cr1 \geq Cr2 \geq Cr3 \geq \ldots \geq Crn$$

$$\text{Maka} \quad W1 \geq W2 \geq W3 \geq \ldots \geq Wn$$

Selanjutnya, jika k merupakan banyaknya kriteria, maka

$$W1 = \frac{1+ 12+ 13+\cdots+1/k}{k}$$
$$W2 = \frac{0+ 12+ 13+\cdots+1/k}{k}$$
$$W3 = \frac{0+ 0+13+\cdots+1/k}{k}$$
$$W4 = \frac{0+0+0+\cdots+1/k}{k}$$

Secara umum pembobotan ROC dapat dirumuskan sebagai berikut :

$$Wk = 1/k \sum_{t=i}^{k} (1/i)$$

2.22 Metode Pengolahan Kuisioner

Kuesioner merupakan sekumpulan pertanyaan yang diajukan kepada responden dengan tujuan dan maksud tertentu untuk memperoleh data, sehingga data tersebut dapat digunakan atau diolah kembali. Untuk mengolah data dari kuesioner tersebut dapat menggunakan berbagai macam metode pengolahan data sesuai dengan rumus metriks yang digunakan pada ISO 9126 Kuisioner ini diajukan untuk memenuhi nilai A, B maupun T yang dibutuhkan untuk menghitung nilai metriks dai setiap kriteria yang sudah ada. Dari hasil jawaban yang diberikan oleh responden akan dicari nilai rata rata, hasil jawaban responden akan dijumlahkan dan totalnya akan dibagi dengan jumlah responden. Maka akan didapat nilai A, B ataupun T [23].

2.23 Metode Penelitian

Perhitungan nilai sub faktor , faktor maupun nilai keseluruhan menjadi faktor penting untuk mengetahui hasil akhir sebuah nilai. menentukan nilai kriteria per faktor nya dan nilai akhir dengan rumus perhitungan Weight Summation. Nilai w adalah nilai metriks yang diperoleh dari hasil pengolahan kuisioner dan m adalah bobot dari sub faktor atau faktor yang sudah diperoleh dari hasil perhitungan dengan menggunakan metode Rank Order Centroid (ROC) [23].

Nilai Kriteria

$$\text{Kriteria (a)} = \sum_{j=1}^{M} W_j . m_{ij} \ldots\ldots$$

$$= w_1 m_1 + w_2 m_2 + \ldots\ldots \text{wnmn}$$