BAB II

LANDASAN TEORI

2.1 Aplikasi

Perangkat lunak aplikasi adalah suatu subkelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan suatu tugas yang diinginkan pengguna atau *user*. Biasanya perangkat lunak sistem dapat mengintegrasikan berbagai kemampuan komputer, tapi tidak secara langsung menerapkan kemampuan tersebut untuk mengerjakan suatu tugas yang menguntungkan pengguna. Contoh utama perangkat lunak aplikasi adalah pengolah kata, lembar kerja, dan pemutar media dan lain sebagainya.

Beberapa aplikasi yang digabung bersama menjadi suatu paket kadang disebut sebagai suatu paket atau suite aplikasi *(application suite)*. Contohnya adalah Microsoft Office dan OpenOffice.org, yang menggabungkan suatu aplikasi pengolah kata, lembar kerja, serta beberapa aplikasi lainnya. Aplikasi-aplikasi dalam suatu paket biasanya memiliki antarmuka pengguna yang memiliki kesamaan sehingga memudahkan pengguna untuk mempelajari dan menggunakan setiap aplikasi. Sering kali, aplikasi ini memiliki kemampuan untuk saling berinteraksi satu sama lain sehingga menguntungkan pengguna. Contohnya, suatu lembar kerja dapat

dibenamkan dalam suatu dokumen pengolah kata walaupun dibuat pada aplikasi lembar kerja yang terpisah.

2.2 Objek Wisata

2.2.1 Pengertian Objek Wisata

a. Pengertian objek wisata atau *tourist attraction*, istilah ini yang lebih sering digunakan, yaitu segala sesuatu yang menjadi daya tarik bagi orang untuk mengunjungi suatu daerah tertentu.

b. Dalam dunia kepariwisataan, segala sesuatu yang menarik dan bernilai untuk dikunjungi dan dilihat, disebut atraksi atau lazim dinamakan objek wisata.

2.2.2 Syarat-syarat Objek Wisata

Sebuah Objek wisata yang baik harus dapat mendatangkan wisatawan sebanyak-banyaknya, menahan mereka ditempat atraksi dalam waktu yang cukup lama dan memberi kepuasan kepada wisatawan yang dating berkunjung. Untuk mencapau hasil itu, beberapa syarat harus dipenuhi yaitu:

a. Kegiatan (act) dan objek (artifact) merupakan atraksi itu sendir harus dalam keadaan yang baik.
b. Karena atraksi wisata itu disajikan dihadapan wisatawan, maka cara penyajiannya harus tepat.

c. Objek atau atraksi wisata adalah terminal dari mobilitas spasial suatu perjalanan. Oleh karena itu juga harus memenuhi semua determinan mobolotas spasial, yaitu akomodasi, transportasi, promosi serta pemasaran.
d. Keadaan di objek wisata harus dapat menahan wisatawan cukup lama.
e. Kesan yang diperoleh wisatawan untuk menyaksikan ataraksi wisata harus diusahakan supaya bertahan selama mungkin.

2.2.3 Karakterisitik Objek Wisata

Selain beberapa persyaratan di atas, terdapat 3 karakterisitik utama dari obek wisata yang harus diperhatikan dalam upaya pengembangan suatu objek wisata tertentu agar dapat menarik dan dikunjungi banyak wisatawan. karakterisitik terserbut antara lain:

a. Daerah itu harus mempunyai apa yang disebut sebagai "*something too see"*. Artinya di tempat tersebut harus ada objek wisata dan atraksi wisata yang berbeda dengan apa yang dimiliki oleh daerah lain. Dengan kata lain, daerah itu harus mempunyai daya tarik yang khusus dan unik.
b. Daerah tersebut harus tersedia apa yang disebut dengan istilah "*something to do"*. Artinya di tempat tersebut selain banyak yang dapat disaksikan, harus disediakan pula fasilitas rekreasi atau *amusement* yang dapat membuat wisatawan betah tinggal lebih lama di tempat itu.

c. Di daerah tersebut harus tersedia apa yang disebut dengan istilah *"something to buy"*. Artinya di tempat tersebut harus ada fasilitas untuk berbelanja, terutama barang-barang souvenir dan kerajinan tangan rakyat sebagai oleh-oleh untuk dibawa pulang.

2.2.4 Jenis Objek Wisata

Seiring dengan perkembangan industri pariwisata, munculah bermacam-macam jenis objek wisata yang kelama-lamaan mempunyai ciriinya tersendiri. Perkembangan ini bertujuan untuk memenuhi kebutuhan wisatawan yang saat ini melakukan perjalana wisata berdasarkan alasan dan tujuan yang berbeda-beda.

Di bawah ini, diuraikan mengenai beberapa jenis objek wisata yang dikelompokkan berdasarkan alas an atau motivasi serta tujuan wisatawan dalam melakukan suatu perjalanan wisata, antara lain:

a. Objek Wisata Budaya

   Perjalana ke objek wisata ini dilakukan atas dasar keinginan untuk memperluas pandangan hidup seseorang, dengan jalan mengadakan kunjungan atau peninjauan ke tempat lain, untuk mempelajari keadaan rakyat, kebiasaan dan adat-istiadat, cara hidup, budaya dan seni mereka.

b. Objek Wisata Kesehatan

Perjalanan seorang wisatawan ke objek wisata ini dilakukan dengan tujuan untuk menukar keadaan dan lunkngan tempat sehari-hari dimana ia tinggal demi kepentingan kesehataanya dan untuk beristirahat.

c. Objek Wisata Olah Raga

Wisatawan yang melakukan perjalanan ke objek wisata ini mempunyai tujuan untuk berolah raga atau memang sengaja bermaksud mengambil bagian aktif dalam pesta olah raga di suatu tempat di Negara tertentu.

d. Objek Wisata Komersial

Perjalanan yang dilakukan ke objek wisata ini dengan tujuan untuk mengunjungi pameran-pameran dan pekan raya yang bersifat komersial.

e. Objek Wisata Politik

Perjalanan ke objek wisata ini dilakukan dengan tujuan untuk mengunjungi atau mengambil bagian aktif dalam peristiwa kegiatan politik.

f. Objek Wisata Pilgrim

Perjalanan wisata ke tempat ini sering dihubungkan dengan agama, sejarah, adat istiadat, dan kepercayaan wisatawan, dan biasanya mempunyai tujuan yang dihubungkan dengan niat atau hasrat sang

wisatawan untuk memperoleh restu, kekuatan batin, keteguhan iman, dan tidak jarang pula untuk tujuan memperoleh berkah dan kekayaan melimpah.

g. Objek Wisata Bahari

Perjalanan ke objek wisata ini banyak dikaitkan dengan kegiatan olah raga di air. Seperti memancing, berlayar, menyelam, berselancar, atau berkeliling melihat taman laut dengan pemandangan indah di bawah permukaan air.

2.3 Android

Pembahasan mengenai android dan *tools* yang digunakan dalam pembuatan aplikasi, akan dijelaskan pada sub bab berikut:

2.3.1 Pengertian Android

Android adalah sekumpulan perangkat lunak yang ditujukan bagi perangkat bergerak mencakup sistem operasi, *middleware,* dan aplikasi kunci. Android *Standart Development Kid* (SDK) menyediakan perlengkapan dan *Application Programming Interface* (API) yang diperlukan untuk mengembangkan aplikasi pada platform Android menggunakan bahasa pemrograman Java.

Android dikembangkan oleh Google bersama *Open Handset Allian*ce (OHA) yaitu aliansi perngkat seluler terbuka yang terdiri dari 47 perusahaan *Hardware,*

Software dan perusaan telekomunikasi ditujukan untuk mengembangkan standar terbuka bagi perangkat selular.

2.3.2 Sejarah dan Perkembangan Android

Pada ulanya terdapat berbagai macam sistem oprasi pada perangkat selular, diantaranya sistem operasi Symbian, Microsoft Windows Mobile, Mobile Linux, iPhone, dan sistem operasi lainnya. Namun diantara sistem operasi yang ada belum mendukung standar dan penerbitan API yang dapat dimanfaatkan secara keseluruhan dan dengan biaya yang murah. Kemudian Google ikut berkecimpung di dalamnya dengan *platform* Android, yang menjanjikan keterbukaan, keterjangkauan, *open source* dan *framework* berkualitas.

Pada tahun 2005, Google mengakuisisi perusahaan Android Inc. untuk mamulai perkembangan *platform* Android. Dimana terlibat pengembangan ini adalah Andy Rubin, Rich Miner, Nick Sears, dan Chris White. Pada pertengahan 2007 sekelompok pemimpin industri bersama-sama membentuk analisis aliansi perangkat selular terbuka, *Open Handset Alliance* (OHA). Bagian dari tujuan aliansi ini adalah berinovasi dengan cepat dan menaggapi kebutuhan konsumen dengan lebih baik, dengan produk awalnya adalah *platform* Android. Dimana Android dirancang untuk melayani kebutuhan operator telekomunikasi, manufaktur *handset*, dan pengembangan aplikasi, OHA berkomitmen untuk membuat Android *open source* dengan lisensi Apache Versi 2.0.

Android pertama kali diluncurkan pada 5 November 2007, dan smartphone pertama yang menggunakan sistem operasi Android dikeluarkan oleh T-Mobile degnan sebutan G1 pada bulan September 2008. Hingga saat ini Android telah merilis beberapa versi Android untuk menyempurnakan versi sebelumnya. Selain berdsarkan penomoran, pada setiap versi Android terdapat kode nama berdasarkan nama-nama kue. Hingga saat ini sudah terdapat beberapa versi yang telah diluncurkan, diantaranya: versi 1.1 dirlis pada 9 maret 2009, versi 1.5 dirilis pada 30 April 2009 diberi nama *Cupcake,* versi 1.6 dirilis pada 15 September 2009 diberi nama *Donut,* versi 2.0 dirilis pada 26 Oktober 2009 diberi nama *Éclair*, versi 2.2 dirilis pada 20 Mei 2010 diberi nama *Froyo* (*Frozen Yoghurt*), versi 2.3 dirilis pada 6 Desember 2010 diberi nama *Gingerbread,* versi 3.0 dirilis pada Mei 2011 diberi nama *Honeycomb*, versi 4.0 dirilis pada 19 Oktober 2011 diberi nama ICS (*Ice Cream Sandwich*).

2.3.3 Kelebihan Android

Sudah banyak *platform* untuk perangkat selular saat ini, termasuk didalamnya Symbian, iPhone, Windows Mobile, BlackBerry, Java Mobile Edition, Linux Mobile (LiM0), 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:

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 kawatir dalam menggunakan aplikasi pada perangkat yang memorinya terbatas.
5. Dukungan grafis dan suarat terbaik, dengan adanya dukungan 2D grafis dan animasi yang diilhami oleh *Flash* menyatu dalam 3D menggunakan *OpenGL* memungkinkan membuat aplikasi maupun game yang berbeda.
6. 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 portabel antara ARM, X86, dan arsitektur lainnya. Sama halnya dengan dukungan masukan seperti penggunaan *Keyboard,* layar sentuh, *trackball* dan resolusi layar semua dapat disesuaikan dengan program.

2.4 Java

2.4.1 Pengertian Java

Java adalah bahasa pemrograman yang disusun oleh James Gosling yang dibantu oleh rekan-rekannya seperti Patrick Naugton, Chris Warth, Ed Frank, dan Mike Sheridan di suatu perusahaan perangkat lunak yang bernama Sun Microsystem, pada tahun 1991. Bahasa pemrograman ini mula-mula diinisialisasi dengan nama "oak", namun pada tahun 1995 diganti namanya menjadi "Java".

Alasan utama pembentukan bahasa java adalah untuk membuat aplikasi-aplikasi yang dapat diletakkan diberbagai macam perangkat elektronik, seperti *microwave oven* dan *remote control,* sehingga Java harus bersifat *portable* atau yang sering disebut dengan *platform independent* (tidak tergantung pada *platform*). Itulah yang menyebabkan dalam dunia pemrograman Java, dikenal adanya istilah '*write once, run everywhere*', yang berarti kode program hanya ditulis sekali, namun dapat dijalankan dibawah *platform* manapun, tanpa harus melakukan perubahan kode program.

2.4.2 Arsitektur Java

Secara arsitektur, Java tidak berubah sedikitpun semenjak awal mula bahasa tersebut dirilis. Kompiler Java (yang disebut dengan **Javac** atau *Java Compiler*) akan

mentransformasikan kode-kode dalam bahsa Java ke dalam suatu *bytecode*. Apa itu *bytecode*? *Bytecode* adalah sekumpulan perintah hasil kompilasi yang kemudian dapat dieksekusi melalui sebuah mesin komputer abstrak, yang disebut dengan JVM (*Java Virtual Machine*). JVM juga sering dinamakan sebagai *interpreter*, karena sifatnya yang selalu menerjemahkan kode-kode yang tersimpan dalam *bytecode* dengan cara baris demi baris.

2.4.2.1 Java Versi Lama (Java 1)

Pada awal perilisannya, versi Java masih disebut dengan JDK (*Java Development kita*). Dalam JDK, semua kebutuhan untuk pengembangan program dan eksekusi program masih tergabung jadi satu. Penamaan ini berlaku sampai Java 1.1. namun sekarang, setelah Java 1.2, Sun Microsystem menamainya dengan JSDK (*Java Software Development Kit*) dalam hal ini kebutuhan untuk pengembangan program dipisahkan dengan kebutuhan kesekusi. Bagian *software* yang digunakan untuk kebutuhan eksekusi program disebut dengan JRE (*Java-Runtime Environment*). Selanjutnya, Java 1.2 disederhanakan penamaanya menjadi “Java 2”.

2.4.2.2 Java Versi Baru (Java 2)

Sun Microsystems telah mendefinisikan tiga buah edisi dari Java 2, yaitu sebagai berikut :

a. Java 2 Standard Edition (J2SE), yang digunakan untuk mengembangkan aplikasi-aplikasi desktop dan applet (aplikasi Java yang dapat dijalankan di dalam *browser web*).

b. Java 2 Enterprise Edition (J2EE), merupakan *superset* dari J2SE yang memperbolehkan untuk mengembangkan palikasi-aplikasi berskala besar (*enterprise*), yaitu dengan melakukan pembuatan aplikasi-aplikasi di sisi *server* dengan menggunakan EJBs (*Enterprise JavaBeans*), aplikasi web dengan menggunakan *Servlet* dan JSP (*JavaServer Pages*) dan teknologi lainnya serperti CORBA (*Common Object Request Broker Architecture*) dan XML (*Extensible Markup Language*).

c. Java 2 Micro Edition (J2ME), merupakan *subset* dari J2SE yang digunakan untuk menangani pemrograman di dalam perangkat-perangkat kecil, yang tidak memungkinkan untuk mendukung implementasi J2SE secara penuh.

2.5 Tools-Tools yang digunakan

2.5.1 Konsep 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 obke. 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).

2.5.1.1 Pengertian UML

Unified Modeling Language **(UML)** merupakan sistem arsitektur yang bekerja dalam OOAD (*Object-Oriented Analysis/Design*) dengan satu bahasa yang konsisten untuk menentukan, visualisasi, mengkontruksi, dan mendokumentasikan *artifact* (sepotong informasi yang digunakan atau dihasilkan dalam suatu proses rekayasa software, dapat berupa model, deskripsi, atau software) yang terdapat dalam sistem software. UML merupakan bahasa pemodelan yang paling sukses dari tiga metode OO yang telah ada sebelumnya, yaitu Booch, OMT (*Object Modeling Technique*), dan OOSE (*Object-Oriented Software Engineering*).

2.5.1.2 Tujuan UML

Tujuan UML yaitu diantaranya:

1. Memberikan model yang siap pakai, bahasa pemodelan visual yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum.
2. Memberikan bahasa pemodelan yang bebas dari berbagai bahasa pemrograman dan proses rekayasa.
3. Menyatukan praktek-praktek terbaik yang terdapat dalam pemodelan.
4. Untuk membuat suatu model, UML memiliki diagram grafis sebagai berikut:

   a) *Business Use Case model*

   b) *Activity Diagram*

   c) *Use Case model*

d) *Behavior diagram : Sequence diagram*

e) *Implementation diagram : Component diagram, Deployment diagram*

f) *Generate Code*

Diagram diagram tersebut diberi nama berdasarkan sudut pandang yang berbeda-beda terhadap sistem dalam proses analisis atau rekayasa. Dibuatnya berbagai jenis diagram diatas karena :

1. Setiap sistem yang kompleks selalu paling baik jika didekati melalui himpunan berbagai sudut pandang yang kecil yang satu sama lain hampir saling bebas (*independent*). Sudut pandang tunggal senantiasa tidak mencukupi untuk melihat isi item yang besar dan kompleks.
2. Diagram yang berbeda-beda tersebut dapat menyatakan tingkatan yang berbeda-beda dalam proses rekayasa.
3. Diagram-diagram tersebut dibuat agar model yang dibuat semakin mendekati realitas.

Diagram-diagram ini ditambah dengan kemampuan dokumentasi perupakan *artifacts* utama UML. *Data-flow diagram* dan tipe diagram lain yang tidak terdapat dalam UML tidak termasuk dalam paradigma *object-oriented*. *Activity diagram* dan *collaboration diagram* yang terdapat dalam UML menggantikan *data-flow diagram*. *Activity diagram* juga sangat bermanfaat untuk membuat *workflow*.

2.5.1.3 Diagram-Diagram Dalam UML

Adapun diagram – diagram dalam UML mempunyai beberapa diagram antara lain yaitu:

A. Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng- *create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng- *include* dieksekusi secara normal. Sebuah *use case* dapat di- *include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common* . Sebuah *use case* juga dapat meng- *extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

B. Class Diagram

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

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok :

1. Nama (dan stereotype)

2. Atribut

3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

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

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.

a. Hubungan Antar Class

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

C. Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*). Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

D. Activity Diagram

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di- *trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih

menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state* , standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

E. Sequence Diagram

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

berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class* . *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message.

Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity*.

F. Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number* , di mana *message* dari level tertinggi memiliki nomor 1. Messages dari level yang sama memiliki prefiks yang sama.

G. Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code* , baik berisi *source code* maupun *binary code* , baik *library* maupun *executable* , baik yang muncul pada *compile time*, *link time* , maupun *run time* . Umumnya komponen terbentuk dari beberapa *class* dan/atau *package* , tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa

interface, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

H. Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di- *deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal Sebuah *node* adalah server, *workstation* , atau piranti keras lain yang digunakan untuk men- *deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

2.5.2 Pengertian StarUML

StarUML adalah sebuah proyek open source untuk mengembangkan cepat, fleksibel, extensible, featureful, dan bebas-tersedia UML / platform MDA berjalan pada platform Win32.Tujuan dari proyek StarUML adalah untuk membangun sebuah alat pemodelan perangkat lunak dan juga platform yang menarik adalah pengganti alat UML komersial seperti Rational Rose, StarUML.

2.5.3 Eclipse

2.5.3.1 Pengertian Eclipse

Eclipse adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua *platform* (*platformindependent*). Berikut ini adalah sifat dari Eclipse:

a. *Multi-platform*: Target sistem operasi Eclipse adalah Microsoft Windows, Linux, Solaris, AIX, HP-UX dan Mac OS X.

b. *Mulit-language*: Eclipse dikembangkan dengan bahasa pemrograman Java, akan tetapi Eclipse mendukung pengembangan aplikasi berbasis bahasa pemrograman lainnya, seperti C/C++, Cobol, Python, Perl, PHP, dan lain sebagainya.

c. *Multi-role*: Selain sebagai IDE untuk pengembangan aplikasi, Eclipse pun bisa digunakan untuk aktivitas dalam siklus pengembangan perangkat lunak, seperti dokumentasi, test perangkat lunak, pengembangan web, dan lain sebagainya.

Eclipse pada saat ini merupakan salah satu IDE favorit dikarenakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Selain itu, kelebihan dari Eclipse yang membuatnya populer adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen yang dinamakan *plug-in*.

2.5.3.2 Arsitektur Eclipse

Sejak versi 3.0, Eclipse pada dasarnya merupakan sebuah *kernel*, yang mengangkat *plug-in*. Apa yang dapat digunakan di dalam Eclipse sebenarnya adalah fungsi dari *plug-in* yang sudah diinstal. Ini merupakan basis dari Eclipse yang dinamakan *Rich Client Platform* (RCP). Berikut ini adalah komponen yang membentuk RCP:

a. *Core platform*

b. OSGi

c. SWT (*Standard Widget Toolkit*)

d. JFace

e. *Eclipse Workbench*

Secara standar Eclipse selalu dilengkapi dengan JDT (*Java Development Tools*), *plug-in* yang membuat Eclipse kompatibel untuk mengembangkan program Java, dan PDE (*Plug-in Development Environment*) untuk mengembangkan *plug-in* baru. Eclipse beserta *plug-in*-nya diimplementasikan dalam bahasa pemrograman Java.

Konsep Eclipse adalah IDE yang terbuka (*open*), mudah diperluas (*extensible*) untuk apa saja, dan tidak untuk sesuatu yang spesifik. Jadi, Eclipse tidak saja untuk mengembangkan program Java, akan tetapi dapat digunakan untuk berbagai macam keperluan, cukup dengan menginstal *plug-in* yang dibutuhkan. Apabila ingin mengembangkan program C/C++ terdapat *plug-in* CDT (*C/C++ Development Tools*).

Selain itu, pengembangan secara visual bukan hal yang tidak mungkin oleh Eclipse, *plug-in* UML2 tersedia untuk membuat diagram UML. Dengan menggunakan PDE setiap orang bisa membuat *plug-in* sesuai dengan keinginannya.

2.5.4 Android SDK

Android SDK adalah *tools* API (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada *platform* android menggunakan bahasa pemrograman Java. Android merupakan subset perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang di *release* oleh Google. Saat ini disediakan Android SDK (*Software Development Kit*) sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi-netral, android member anda kesempatan unutk membuat aplikasi yang kita butuhkan yang ubkan merupakan aplikasi bawaan Hadphone/*Smartphone*. Beberapa fitur-fitur android yang paling penting adalah :

a. *Framework* : aplikasi yang mendukung pengganti komponen dan reusable.

b. *Dalvik Virtual Machine* dioptimalkan untuk perangkat *mobile*

c. Integrated *Browser* verdasarkan engine *open source* WebKit.

d. Grafis yang dioptimalkan dan didukung oleh *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi opengl ES 1,0 (Opsional Ekselerasi *hardware*)

e. SQLite untuk penyimpanan data.

f. Media *Support* yang mendukung audio, video, dan gambar (MPEG4, H.264,

MP3, AAC, AMR, JPG, PING, GIF), GSM Telephony (tergantung H*ardware*)

g. *Bluetooth*, EDGE, 3G, dan WiFi (tergantung *hardware*)

h. Kamera, GPS, Kompas, dan *Accelerometer* (tergantung *hardware*)

i. Lingkungan *Development* yang lengkap dan termasuk pernagkat *emulator*, *tools* untuk *debugging*, profil dan kinerja memori, dan *plugin* untuk IDE Eclipse.

Untuk source SDK Android ini dapat dilihat dan didownload langsung di situs resmi pengembang SDK Android di http://www.developer.android.com

2.5.5 Google Maps

Google Maps merupakan sebuah layanan peta dunia virtual berbasis web yang disediakan oleh Google. Layanan ini gratis dan dapat ditemukan di http://maps.google.com. Google Maps menawarkan peta yang dapat digeser (*panned*), diperbesar (*zoom in*), diperkecil (*zoom out*), dapat diganti dalam beberapa mode (*ma*p, *satelit*, *hybrid*, dan lain-lain), fitur pencarian rute (*routing*), penunjuk arah dari satu objek peta ke objek yang lain (*direction*) dan juga pencarian tempat (*place*). Sampai saat ini, Google Maps masih berada dalam tahap beta, dan masih terus dikembangkan dengan data yang selalu diperbarui secara berkala.

Google maps merupakan hak cipta Google secara propriety, sehingga dalam

menggunakannya memerlukan adanya perjanjian, Google membuat mekanisme untuk dapat mengakses Google Maps dengan coding aplikasi dengan sebuah kunci yang dikenal dengan API *Key*.

2.5.6 SQLite

SQLite merupakan sebuah sistem manajemen basisdata relasional yang bersifat ACID-compliant dan memiliki ukuran pustaka kode yang relatif kecil, ditulis dalam bahasa C. SQLite merupakan proyek yang bersifat *public domain* yang dikerjakan oleh D. Richard Hipp.

Tidak seperti pada paradigma client-server umumnya, Inti SQLite bukanlah sebuah sistem yang mandiri yang berkomunikasi dengan sebuah program, melainkan sebagai bagian integral dari sebuah program secara keseluruhan. Sehingga protokol komunikasi utama yang digunakan adalah melalui pemanggilan API secara langsung melalui bahasa pemrograman. Mekanisme seperti ini tentunya membawa keuntungan karena dapat mereduksi *overhead*, *latency times*, dan secara keseluruhan lebih sederhana. Seluruh elemen basisdata (definisi data, tabel, indeks, dan data) disimpan sebagai sebuah file. Kesederhanaan dari sisi disain tersebut bisa diraih dengan cara mengunci keseluruhan file basis data pada saat sebuah transaksi dimulai.

2.6 Pengujian Software

Pengujian software adalah proses untuk memastikan apakah semua fungsi sistem bekerja dengan baik, dan mencari apakah masih ada kesalahan pada sistem.

Pengujian atau testing software sangat penting untuk dilakukan. Pengujian ini bertujuan untuk menjamin kualitas software, dan juga menjadi peninjauan terakhir terhadap spesifikasi, disain dan pengkodean.

Terdapat dua pendekatan dalam melakukan pengujian software , yaitu :

1. **Black-box testing**

Pendekatan ini melakukan pengujian terhadap fungsi operasional software. Pendekatan ini biasanya dilakukan oleh penguji yang tidak ikut serta dalam pengkodean software.

2. **White-box testing**

Metoda ini dilakukan oleh orang yang memahami cara kerja operasi internal software yang membentuk keseluruhan operasi software.