BAB II

TINJAUAN PUSTAKA

2.1 K.H. Zaenal Mustofa

Zaenal Mustofa mempunyai nama asli Hudaeni. Dia lahir di kampung Bageur, Singaparna, Tasikmalaya, tahun 1899. Zaenal terlahir dari keluarga yang hidup berkecukupan. Ayahnya bernama Nawapi, sedangkan ibunya bernama Ratmah. Zaenal Mustofa kecil menempuh pendidikan formal di Sekolah Rakyat. Setelah itu, dia belajar ilmu agama ke berbagai pesantren yang ada di Jawa. Oleh karena berasal dari keluarga berkecukupan, Zaenal Mustofa mendapat kesempatan untuk menunaikan ibadah haji. Disana dia juga banyak belajar ilmu agama dari beberapa ulama. Zaenal Mustofa juga mendirikan pondok pesantren dengan nama pesantren Sukamanah.

Pada tahun 1940, Zaenal Mustofa secara terang – terangan mengadakan kegiatan dakwah yang memacu semangat kebangsaan melawan penjajah. Melalui khotbah – khotbahnya dia tegas menolak sikap penjajah yang terjadi di bangsanya. 17 November 1941, K.H Zaenal Mustofa bersama para ulama lainnya ditangkap Belanda karena tuduhan telah menghasut rakyat untuk memberontak terhadap pemerintahan Hindia Belanda. Mereka ditahan di penjara Tasikmalaya dan sehari kemudian di pindahkan ke Penjara Sukamiskin, Bandung. Pada tanggal 10 Januari 1942, mereka akhirnya dibebaskan dari penjara. Walaupun demikian, Zaenal Mustofa tak menghentikan aksinya untuk berdakwah melawan penjajah. Akhirnya pada tahun 1942 dia kembali dimasukan ke penjara dengan tuduhan yang sama.

Pada tanggal 8 Maret 1942, Belanda menyerah kepada Jepang, Indonesia pun dikuasai oleh Jepang. Akhirnya Zaenal Mustofa dibebaskan oleh Jepang, dengan harapan Zaenal Mustofa dapat membantu penjajah Jepang. Namun, zaenal Mustofa tidak pernah akan pernah bekerja sama dengan para penjajah. Bahkan dia menentang dengan terang – terangan ajaran Jepang yang bertentangan dengan agama. Misalnya, ajaran memberi hormat kearah matahari terbit karena hal itu hal

itu dapat mengubah arah kiblat dan merupakan perbuatan yang merusak akidah umat Islam. Tidak hanya itu, dia juga menentang pelaksanaan Romusha (kerja paksa) yang menyengsarakan rakyat Indonesia.

Pada tanggal 25 Januari 1944, Zaenal Mustofa mengeluarkan peringatan keras kepada pemerintah Jepang. Dia membentuk pasukan Sukamanah dan melakukan persiapan untuk melakukan perlawan kepada Jepang. Namun sayangnya, aksinya itu diketahui oleh Jepang. Zaenal Mustofa pun akhirnya ditangkap dan disiksa. Pada tanggal 28 Maret 1945, akhirnya di hukum mati setelah sebelumnya disiksa oleh jepang. Jenazahnya dimakamkan di Jakarta, tetapi kemudian di pindahkan ke Taman Makam Pahlawan Tasikmalaya pada 10 Januari 1974. Untuk menghormati pengorbanannya kepada bangsa dan Negara, berdasarkan SK Keppres No. 64/TK/1972 pada tanggal 6 November 1972, beliau resmi dianugrahi gelar Pahlawan Nasional.[2]



Gambar 2.1**K.H Zaenal Mustofa**

2.2 *Game*

Game merupakan kata dalam bahasa inggris yang berarti permainan. Permainan adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius atau dengan tujuan *refreshing*. Suatu cara belajar yang digunakan dalam menganalisa interaksi antara sejumlah pemain maupun perorangan yang menunjukkan strategi strategi yang rasional. Teori permainan pertama kali ditemukan oleh sekelompok

ahli Matematika pada tahun 1944. Teori itu dikemukakan oleh John von Neumann and Oskar Morgenstern yang berisi :

"Permainan terdiri atas sekumpulan peraturan yang membangun situasi bersaing dari dua sampai beberapa orang atau kelompok dengan memilih strategi yang dibangun untuk memaksimalkan kemenangan sendiri atau pun untuk meminimalkan kemenangan lawan. Peraturan-peraturan menentukan kemungkinan tindakan untuk setiap pemain, sejumlah keterangan diterima setiap pemain sebagai kemajuan bermain, dan sejumlah kemenangan atau kekalahan dalam berbagai situasi." (*Theory of Games and Economic Behavior* (3d ed. 1953))[3].

2.2.1 Pengertian *Game*

Pengertian *game* menurut beberapa ahli [4] :

1. Menurut Agustinus Nilwan dalam bukunya Pemrograman Animasi dan *Game* Profesional terbitan Elex Media Komputindo, *game* merupakan permainan komputer yang dibuat dengan teknik dan metode animasi. Jika ingin mendalami pengunaan animasi haruslah memahami pembuatan *game*. Atau jika ingin membuat *game*, maka haruslah memahami teknik dan metode animasi, sebab keduanya saling berkaitan.
2. Menurut Clark C. Abt, Game adalah kegiatan yang melibatkan keputusan pemain, berupaya mencapai tujuan dengan dibatasi oleh konteks tertentu (misalnya, dibatasi oleh peraturan).
3. Menurut Chris Crawford, seorang computer *game* designer mengemukakan bahwa *game*, pada intinya adalah sebuah interaktif, aktivitas yang berpusat pada sebuah pencapaian, ada pelaku aktif *(player)*, ada pelaku pasif *(NPC*).
4. Menurut David Parlett, Game adalah sesuatu yang memiliki "akhir dan cara mencapainya" : artinya ada tujuan, hasil dan serangkaian peraturan untuk mencapai keduanya.
5. Menurut Roger Caillois, seorang sosiolog Perancis, dalam bukunya yang berjudul Les jeux et les hommes menyatakan game adalah aktivitas yang

mencakup karakteristik berikut: fun (bebas bermain adalah pilihan bukan kewajiban), separate (terpisah), uncertain, non-productive, governed by rules (ada aturan), fictitious (pura-pura).

6. Menurut Greg Costikyan, Game adalah "sebentuk karya seni di mana peserta, yang disebut Pemain, membuat keputusan untuk mengelola sumberdaya yang dimilikinya melalui benda di dalam *game* demi mencapai tujuan".

2.2.2 Sejarah *Game* Pertama di Indonesia

Divinekids adalah *game* yang diakui dan mendapat *award* resmi dari MURI (Museum Rekor Indonesia) sebagai *Game* Pertama Indonesia pada tahun 2004 dan banyak majalah-majalah *Game* dan Komputer (*Gamestation*, PCGAME, Bobo, INDOPOS, Game21 dan lain sebagainya). Sebenarnya sudah banyak *game-game* buatan orang-orang indonesia yang lain tetapi karena kekurangan syarat-syarat minimal tertentu seperti : Karakter menggunakan karakter asing, penyebaran yang kurang luas, bahasa yang digunakan bukan bahasa Indonesia, tidak terdaftar resmi *Copyright* ataupun Trademarknya dan lain sebagainya, karena itu tidak dipilih sebagai *Game* Pertama Indonesia Resmi.

Menurut sebagian pengamat *game* ini adalah kurangnya investigasi MURI dalam hal melihat adanya *game-game* tersebut sebelum memberikan gelar *game* pertama ke David Setiabudi. Terlebih mereka tidak meminta pendapat dari pihak yang memahami dunia *game* Indonesia itu sendiri.

Pengamat – pengamat yang berpendapat berbeda dengan MURI dan majalah-majalah PC dan *Game*, memposisikan sejarah *game* Indonesia sebagai berikut: Sejarah *Game* Indonesia menurut *Gamedevid*. Selain itu juga masih ada banyak *game-game* lain yang belum terdokumentasi dengan baik. Beberapa *game* buatan Elex Media Komputindo dengan *copyright* resmi dan penyebaran yang luas ada juga yang dibuat sebelum sebelum dirilisnya Divine Kids. *Game* ini dibuat dan dibagikan secara gratis melalui CD majalah *game*/ komputer dan melalui Internet.

Game-game Divinekids berjumlah 38 lebih dan terus bertambah (2006). Pembuatnya adalah David Setiabudi dan mahasiswa-mahasiswi bimbingannya. Pembuatan musik *game*nya dibuat oleh Rama Aurora, tunanetra berbakat membuat musik digital.

David Setiabudi membuat *game* juga tahun 1992 tetapi tidak diakui sebagai *game* pertama Indonesia karena kurang memenuhi syarat. Ada juga *game-game* buatan Korea yang terkenal diterjemahkan ke bahasa Indonesia & *game-game* yang penyebaran kurang luas & *game* yang berkaraker asing atau berbahasa asing & tidak terdaftar *Copyright* secara resmi tidak terpilih menjadi *Game* Pertama Indonesia. Ada yang berpendapat asal tahun tertulis paling tua itu patut disebut *Game* Pertama Indonesia tidak peduli dengan buatan negara mana, atau tersebar di lokal saja. Sampai sekarang masih diperdebatkan kelompok tertentu[6].

2.2.3 Klasifikasi *Game*

2.2.3.1 Berdasarkan Jenis *Platform* yang digunakan

Berikut klasifikasi *game* berdasarkan jenis *platform* yang digunakan:

1. *Arcade games*, yaitu yang sering disebut *ding-dong* di Indonesia, biasanya berada di daerah / tempat khusus dan memiliki box atau mesin yang memang khusus di *design* untuk jenis *video games* tertentu dan tidak jarang bahkan memiliki fitur yang dapat membuat pemainnya lebih merasa masuk dan menikmati, seperti pistol, kursi khusus, sensor gerakan, sensor injakkan dan stir mobil (beserta transmisinya tentunya)
2. *PC Games*, yaitu *video game* yang dimainkan menggunakan *Personal Computers* atau sering disebut Desktop.
3. *Console games*, yaitu *video games* yang dimainkan menggunakan *console* tertentu, seperti Playstation 2, Playstation 3, XBOX 360, dan Nintendo Wii.
4. *Handheld games*, yaitu yang dimainkan di *console* khusus video *game* yang dapat dibawa kemana-mana, contoh Nintendo DS dan Sony PSP.

5. *Mobile games*, yaitu yang dapat dimainkan atau khusus untuk *mobilephone* atau PDA.

2.2.3.2 Berdasarkan *Genre* Permainannya

Berikut klasifikasi *game* berdasarkan genre permainannya [4]:

1. Aksi *Shooting*, *video game* jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga timing, inti dari *game* jenis ini adalah tembak, tembak dan tembak.
   a. *First person shooting (FPS)* seperti *Counter Strike* dan *Call of Duty*
   Drive n shoot, menggunakan unsur simulasi kendaraan tetapi tetap dengan tujuan utama menembak dan menghancurkan lawan, contoh : *Spy Hunter, Rock and Roll Racing*, *Road Rash.*
   b. *Shoot em up*, seperti Raiden, 1942, dan gradius.
   c. *Beat em up* (tonjok hajar) seperti *Double Dragon* dan *Final Fight*, lalu *hack and slash* (tusuk tebas) seperti *Shinobi* dan *Legend of Kage.*
   d. *Light gun shooting*, yang menggunakan alat yang umumnya berbentuk seperti senjata, seperti *Virtua Cop* dan *Time Crisis.*
   e. *Fighting* (pertarungan) Ada yang mengelompokan *video game fighting* di bagian aksi, namun penulis berpendapat berbeda, jenis ini memang memerlukan kecepatan refleks dan koordinasi mata-tangan, tetapi inti dari *game* ini adalah penguasaan jurus (hafal caranya dan lancar mengeksekusinya).
2. Petualangan. Bedanya dengan jenis *video game* aksi-petualangan, *refleks* dan kelihaian pemain dalam bergerak, berlari, melompat hingga memecut atau menembak tidak diperlukan di sini. *Video Game* murni petualangan lebih menekankan pada jalan cerita dan kemampuan berpikir pemain dalam menganalisa tempat secara *visual*, memecahkan teka-teki maupun menyimpulkan rangkaian peristiwa dan percakapan karakter hingga penggunaan benda-benda tepat pada tempat yang tepat.

3. Simulasi. Konstruksi dan manajemen. *Video Game* jenis ini seringkali menggambarkan dunia di dalamnya sedekat mungkin dengan dunia nyata dan memperhatikan dengan detil berbagai faktor. *Video Game* jenis ini membuat pemain harus berpikir untuk mendirikan, membangun dan mengatasi masalah dengan menggunakan dana yang terbatas. Contoh: *Sim City, The Sims, Tamagotchi.*
4. *Role Playing. Video game* jenis ini sesuai dengan terjemahannya, bermain peran, memiliki penekanan pada tokoh/peran perwakilan pemain di dalam permainan, yang biasanya adalah tokoh utamanya, dimana sering kita memainkannya, karakter tersebut dapat berubah dan berkembang ke arah yang diinginkan pemain (biasanya menjadi semakin hebat, semakin kuat, semakin berpengaruh, dan lain-lain) dalam berbagai parameter yang biasanya ditentukan dengan naiknya level, baik dari status kepintaran, kecepatan dan kekuatan karakter. seperti *Final Fantasy*, *Dragon Quest* dan *Xenogears. Never Winter Nights, baldurs gate, Elder Scroll,* dan *Fallout.*
5. *Puzzle. Video game* jenis ini sesuai namanya berintikan mengenai pemecahan teka-teki, baik itu menyusun balok, menyamakan warna bola, memecahkan perhitungan matematika, melewati labirin, sampai mendorong-dorong kota masuk ke tempat yang seharusnya, itu semua termasuk dalam jenis ini. Sering pula permainan jenis ini adalah juga unsur permainan dalam *video game* petualangan maupun *game* edukasi. *Tetris, Minesweeper, Bejeweled, Sokoban* dan *Bomberman.*
6. Simulasi kendaraan. *Video Game* jenis ini memberikan pengalaman atau interaktifitas sedekat mungkin dengan kendaraan yang aslinya, muskipun terkadang kendaraan tersebut masih eksperimen atau bahkan fiktip, tapi ada penekanan khusus pada detail dan pengalaman realistik menggunakan kendaraan tersebut.
7. Olahraga. Singkat padat jelas, bermain *sport* di PC atau konsol anda. Biasanya permainannya diusahakan serealistik mungkin walau kadang ada yang menambah unsur fiksi seperti *NBA JAM*. Contohnya pun jelas,

Seri *Winning Eleven*, seri *NBA*, seri *FIFA, John Madden NFL, Lakers vs Celtics, Tony hawk pro skater,* dan lain-lain.

2.2.3.3 Sudut Pandang Permainan (*Game View Point*)

Sebuah *game* biasanya mempunyai sudut pandang permainan tersendiri disesuaikan berdasarkan *genre game* yang diambil. Berikut beberapa macam sudut pandang permainan yang biasa digunakan :

1. *Side Scrolling*

   Adalah sudut pandang permainan yang terlihat dari samping dan memungkinkan karakter utama untuk bergerak dari kiri ke kanan serta memungkinkan *background* pada *game* seolah-olah bergeser mengikuti pergerakan karakter utama.

2. *Top Down*

   Adalah sudut pandang permainan yang memungkinkan karakter utama bermanuver ke empat arah namun cara permainannya sendiri bergeser dari bawah ke atas, dan biasanya *game* yang menggunakan sudut pandang permainan jenis ini adalah *shooter game.*

3. *Isometric*

   Adalah sudut pandang permainan yang memungkinkan permainan terlihat diantara sisi *Side Scrolling* dan juga *Top Down*, dan biasanya diterapkan pada *game* dengan *genre RTS (Real Time Strategy).*

2.2.3.4 Game Berdasarkan Dimensi

Dimensi adalah angka yang berhubungan dengan sifat metrik atau topologi dari suatu objek. Berdasarkan dimensi dari objek-objeknya, game dapat dibagi menjadi beberapa jenis, diantaranya:

A. *Game* 2D

   Game dua dimensi atau 2D merupakan suatu konsep dimana semua objek berada pada satu bidang datar. Gerakan pada game 2D dibatasi

hanya horizontal dan vertikal atau secara koordinat gerakan pemain dibatasi hanya dapat bergerak pada sumbu X danY. Pada game 2D terdapat dua pergerakan kamera. Pertama adalah kamera statis dimana gambar latar (backround) dan tempat game 2D tidak bergerak sama sekali, contoh dalam jenis ini adalah tetris. Kedua adalah Side Scrolling dimana game yang kita mainkan mempunyai kamera yang dapat bergeser ke kanan atau ke kiri dengan kecepatan sesuai dengan gerakan dan kecepatan karakter yang kita mainkan/gerakkan pada game tersebut, contoh game yang termasuk pada jenis ini adalah Super Mario Bross, Sonic dan Megaman.

B. Game 3D

Game 3D menggunakan tiga dimensional representasi geometris data (X, Y, Z) yang disimpan dalam komputer untuk keperluan perhitungan dan rendering gambar 2D. Dalam sebuah game 3D, pemain bisa melihat sebuah objek dari sudut 360°. Terdapat tiga dasar dalam pembuatan sebuah objek 3D, yaitu 3D modeling, 3D rendering, dan 3D computer graphics software[14].

2.3 OOP (*Object Oriented Programming*)

OOP (*Object Oriented Programming*) atau yang dikenal dengan Pemrograman Berorientasi Objek merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut

(terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki *manager*, sekretaris, petugas administrasi data dan lainnya. Misal *manager* tersebut ingin memperoleh data dari bagian administrasi maka *manager* tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang *manager* tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi *manager* bisa mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

Pemrograman orientasi-objek menekankan konsep berikut:

a. **Kelas (*Class*)** — kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh '*class of dog*' adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan *domain* permasalahan yang ada, dan kode yang terdapat dalam sebuah *class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.
b. **Objek (*Object*)** - membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.
c. **Abstraksi (*Abstract*)** - Kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk

memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

d. Enkapsulasi (*Encapsulation*) - Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

e. Polimorfisme (*Polimorfism*) melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan; metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesan tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tungal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai *polimorfisme* melalui penggunaan fungsi kelas-pertama.

f. Inheritas (*Inheritance*). Mengatur *polimorfisme* dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis

khusus dari objek yang sudah ada objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa harus mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas)[8].

2.4 *Unified Modelling Language* (UML)

UML (*Unified Modeling Language*) adalah sebuah bahasa untuk menetukan, visualisasi, kontruksi, dan mendokumentasikan artifact (bagian dari informasi yang digunakan atau dihasilkan dalam suatu proses pembuatan perangkat lunak, terutama yang dikembangkan paradigma berbasis objek [10]. Artifact dapat berupa model, deskripsi atau perangkat lunak) dari sistem perangkat lunak, seperti pada pemodelan bisnis dan system non perangkat lunak lainnya.

UML merupakan suatu kumpulan teknik terbaik yang telah terbukti keberhasilannya dalam memodelkan sistem yang besar dan kompleks. UML tidak hanya digunakan dalam proses pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.

2.4.1 Bagian-Bagian UML

Bagian-bagian utama dari UML adalah *view*, diagram, model *element*, dan *general mechanism.*

1. *View*

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis *view* dalam UML antara lain: *use case view*, *logical view*, *component view*, *concurrency view*, dan *deployment view*.

a. *Use case view*

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. Aktor yang berinteraksi dengan sistem dapat berupa pengguna atau sistem lainnya. *View* ini digambarkan dalam

use case diagrams dan kadang-kadang dengan *activity diagrams. View* ini digunakan terutama untuk klien, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

b. Logical view

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class, object*,dan *relationship*) dan kolaborasi dinamis yang terjadi ketika objek mengirim pesan ke objek lain dalam suatu fungsi tertentu. *View* ini digambarkan dalam *class diagrams* untuk struktur statis dan dalam *state, sequence, collaboration*, dan *activity diagram* untuk model dinamisnya. *View* ini digunakan untuk perancang (*designer)* dan pengembang (*developer*).

c. Component view

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari *code module* diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi administratif lainnya. *View* ini digambarkan dalam *component view* dan digunakan untuk pengembang (*developer*).

d. Concurrency view

Membagi sistem ke dalam proses dan prosesor. *View* ini digambarkan dalam *diagram* dinamis (*state, sequence, collaboration*, dan *activity diagrams*) dan diagram implementasi (*component* dan *deployment diagrams*) serta digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

e. Deployment view

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. *View* ini digambarkan dalam *deployment diagrams* dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

2. Diagram

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem.

Sebuah *diagram* merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk *view* tertentu. Adapun jenis *diagram* antara lain :

A. Use Case Diagram

Use case adalah abstraksi dari interaksi antara sistem dan aktor. *Use case* bekerja dengan cara mendeskripsikan tipe interaksi antara pengguna sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. *Use case* merupakan konstruksi untuk mendeskripsikan bagaimana sistem akan terlihat di mata *user*. Sedangkan *use case diagram* memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan *client*.

B. Class Diagram

Class adalah dekripsi kelompok obyek-obyek dengan properti, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class diagram* dapat memberikan pandangan global atas sebuah system. Hal tersebut tercermin dari *class-class* yang ada dan relasinya satu dengan yang lainnya. Sebuah sistem biasanya mempunyai beberapa *class diagram*. *Class diagram* sangat membantu dalam visualisasi struktur kelas dari suatu sistem.

Rational Objectory Process menyarankan untuk menemukan *class-class* dalam sistem yang sedang dibangun dengan mencari *class* : *boundary*, *control* dan *entity*. Ketiga *stereotypes* ini menggambarkan sebuah sudut pandang model-view-controller sehingga membuat analis dapat membagi sistem dengan memisahkan sudut pandang dari domain dari control yang dibutuhkan oleh sistem.

a. Entity Class

Entity class memodelkan informasi dan operasi yang biasanya berumur panjang/lama. Tipe *class* ini menggambarkan entitas dunia nyata atau entitas yang dibutuhkan untuk melakukan tugas internal sistem.Mereka biasanya tidak terikat oleh komunikasi antara sistem dengan lingkungannya. Kebanyakan, mereka tidak terikat oleh aplikasi, artinya

mereka dapat digunakan lebih dari satu aplikasi. *Entity class* biasanya merupakan class yang dibutuhkan sistem untuk menyelesaikan beberapa kewajiban.

Entity class biasanya ditemukan dalam phasa eloborasi. *Entity class* sering disebut domain class karena mereka berhubungan dengan dunia nyata.

b. Boundary Class

Boundary class menangani komunikasi antara lingkungan sistem dan kedalam sistem. Mereka dapat menjamin *interface* ke pengguna atau sistem lain (misalnya, *interface* ke *actor*). Boundary class digunakan untuk memodelkan sistem interface.

Setiap pasangan actor/skenario (sebuah instance dari use case) diperiksa untuk menemukan boundary class. Boundary class yang ditemukan pada phasa elaboration biasanya pada high level. Sebagai contoh, anda sedang mendesign windows tetapi anda tidak memodelkan semua dialog box dan tombol. Anda hanya sedang mendokumentasikan kebutuhan antar muka, bukan mengimplementasikan antar mukanya. Pada saat design, class-class ini diperbaiki untuk dipertimbangkan memilih antar mukanya

c. Control Class

Control class memodelkan urutan kelakukan *(behavior)* khusus untuk satu atau lebih use case. Pada awal phasa Elaboration, sebuah control class ditambahkan untuk setiap pasangan actor atau use case. Control class bertanggung jawab untuk aliran kejadian-kejadian dalam use case.

Penambahan control class per pasangan actor atau use case hanya merupakan initial cut, pada analisa dan design, control class mungkin dihilangkan, dipecah atau digabung.

C. Component Diagram

Component software merupakan bagian fisik dari sebuah sistem, karena menetap di komputer tidak berada di benak para analis. Komponen merupakan implementasi *software* dari satu atau lebih *class*. Komponen dapat berupa *source code*, komponen biner, atau *executable component*.

Sebuah komponen berisi informasi tentang logika *class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*. Sehingga *component diagram* merepresentasikan dunia nyata yaitu *component software* yang mengandung *component, interface* dan *relationship.*

D. Deployment Diagram

Menggambarkan tata letak sebuah system secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware*, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes, executable component* dan objek yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh *node* tertentu dan ketergantungan komponen.

E. State Diagram

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu objek dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa objek lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

F. Sequence Diagram

*Sequence Diagra*m digunakan untuk menggambarkan perilaku pada sebuah skenario. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antara objek, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

G. Collaboration Diagram

Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan objek dan hubungannya. Jika penekannya pada waktu atau urutan, maka lebih baik menggunakan *sequence diagrams*, tetapi jika penekanannya pada konteks gunakan *collaboration diagram.*

H. Activity Diagram

Menggambarkan rangkaian aliran dari aktifitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

2.5 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 Susn Microsystems, pada tahun 1991 (Raharjo.B, Heryanto. I, Arif Haryano, 2009, hlm. 1-2). Bahasa pemrograman ini mula-mula diinisialisasi dengan nama "Oak", namun oada 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 *porTabel* atau yang sering disebut dengan *platform independent* (tidak bergantung 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 di bawah *platform* manapun, tanpa harus melakukan perubahan kode program.

2.5.1 *Java* Versi 1

Pada awalnya perilisannya, versi *Java* masih disebut dengan JDK (*Java Development Kit*). Dalam JDK, semua kebutuhan untuk pengembangan program dan eksekusi program masih tergabung jadi satu. Penamaan ini berlaku sampai dengan *Java* 1.1. Namun sekarang, setelah *Java* 1.2, Sun Mycrosystems menamainya dengan JSDK (*Java Software Development Kit*) dalam hal ini kebutuhan untuk pengembangan program dipisahkan dengan kebutuhan eksekusi. Bagian *software* yang digunakan untuk kebutuhan eksekusi program disebut

dengan JRE (*Java-Runtime Environment*). Selanjutnya, *Java* 1.2 disederhanakan penamaannya menjadi "*Java* 2"[11].

2.5.2 *Java Versi* 2

Sun Microsystems telah mendefinisikan tiga buah edisi dari *Java* 2 yaitu sebagai berikut [11] :

1. ***Java 2 Standard Edition* (J2SE),** yang digunakan untuk mengembangkan aplikasi-aplikasi *desktop* dan *applet* (aplikasi untuk *Java* yang dapal dijalankan di dalam *browser web*).
2. ***Java 2 Enterprise Edition* (J2EE),** merupakan *superset* dari J2SE yang memperbolehkan kita untuk mengembangkan aplikasi-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 seperti CORBA (*Common Object Request Broker Architecture*) dan XML (*Extensible Markup Language*).
3. ***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 seperti pada teknologi *mobile*.

Keunggulan *java* dibanding bahasa pemrograman lainnya, adalah sebagai berikut :

1. *Java* bersifat sederhana dan relatif mudah

   Java dimodelkan sebagian dari bahasa C++, namun dengan memperbaiki beberapa karakteristik dari C++ sendiri, seperti misalnya dalam mengurangi kompleksitas beberapa fitur, lalu penambahan fungsionalitas, serta penghilangan beberapa aspek pemicu ketidakstabilan sistem pada C++.

   Sebagai contoh, *Java* menggantikan konsep pewarisan lebih dari satu (*multiple inheritance*) dengan *interface*, menghilangkan konsep pointer yang sering membingungkan, otomatis sistem alokasi *memory*, dan sebagainya. Ini

membuat Java menjadi relatif sederhana dan mudah untuk dipelajari dibandingkan bahasa pemrograman lainnya.

2. *Java* berorientasi pada objek (*Object Oriented*)

Java adalah bahasa pemrograman yang berorientasi objek bukan seperti *Pascal, Basic*, atau C yang masih berbasis prosedural. Dalam memecahkan masalah *Java* membagi program menjadi objek-objek, kemudian memodelkan sifat dan tingkah lakunya masing-masing. Selanjutnya Java menentukan dan mengatur interaksi antara objek yang satu dengan lainnya.

3. *Java* bersifat terdistribusi

Pada dekade awal perkembangan PC (*Personal Komputer*), komputer hanya bersifat sebagai *workstation* tunggal, tidak terhubung satu sama lain. Saat ini sistem komputerisasi cenderung terdisibusi, mulai dari *workstation client, e-mail server, database server, web server, proxy server* dan sebainya.

4. *Java* bersifat *Multiplatform*

Dewasa ini kita mengenal banyak *platform Operating System*, mulai dari Windows, Apple, berbagai varian UNIX dan Linux, dan sebagainya. Pada umumnya program yang dibuat dan dikompile di suatu *platform* hanya bisa dijalankan di *platform* tersebut. *Java* bersifat multi *platform*, yakni dapat diterjemahkan oleh *Java Interpreter* pada berbagai sistem operasi.

5. *Java* bersifat *MultiThread*

Thread adalah proses yang dapat dikerjakan oleh program dalam suatu waktu. *Java* bersifat *Multithreading*, artinya dapat mengerjakan beberapa proses dalam waktu yang hampir bersamaan [10].

2.6 Greenfoot

Greenfoot merupakan *software* pembelajaran pemrograman berorientasi objek, dengan berbasis pemrograman *java* yang memungkinkan kita untuk membuat *game* 2 dimensi. *Greenfoot* diprakarsai oleh Michael Kolling pada tahun 2003. *Prototype* pertama dibangun oleh Michael Kolling dan Poul Henrikson pada

tahun 2003-2004. Pembangunan dilanjutkan pada tahun 2005 oleh kelompok Blue J lain dari University of Kent dan Deakin University. *Greenfoot* versi 1.0 pertama dirilis pada tanggal 31 Mei 2006. Pada bulan Maret 2009, proyek *Greenfoot* menjadi software *Open Source* dan berlisensi dibawah GNU GPL. Untuk menjalankan *Greenfoot* ini diharuskan sudah menginstal program *java* 5 (JDK 1.5) atau program *java* terbaru. Didalam *Greenfoot* terdapat project yang dinamakan *scenario*.

Dalam *project* tersebut, kita dapat membuat dua macam tipe *class* dari *library Greenfoot*, yaitu *class world* dan *class actor*. *World* merupakan latar dari *game* yang akan dibuat. *Class actor* adalah kelas yang berfungsi sebagai karakter pada *game* yang akan dibuat [12].

2.7 Corel Draw

Corel Draw merupakan salah satu aplikasi pengolah gambar berbasis vector yang banyak dipakai oleh pengguna PC. Karena berbagai kemudahan dan keunggulan yang dimiliki oleh corel *draw*, maka corel *draw* sering dimanfaatkan untuk *desktop publishing*, percetakan, dan bidang lain yang memerlukan pemrosesan visual. Keunggulan mengolah gambar berbasis vector adalah ukuran hasil akhir yang dapat ditekan seminimal mungkin namun dengan kualitas yang tidak kalah dengan gambar berbasis *raster* atau *bitmap*.

Sebenarnya ada banyak sekali *software* desain grafis dan pengolah gambar berbasis vector yang bisa kita gunakan, misalnya saja diantaranya adalah corel *draw*, *photoshop, freehand, adobe ilustrator* dan lain-lain. Namun corel *draw* merupakan program paling populer yang sering dimanfaatkan. Selain karena tampilannya yang *user friendly* dan mudah dipelajari, corel *draw* juga mempunyai berbagai keunggulan - keunggulan lain yang sangat bermanfaat [13].

2.8 Pengujian Perangkat Lunak

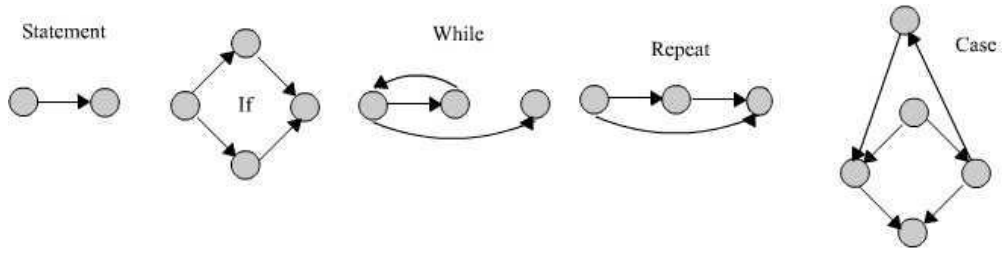
2.8.1 Alpha testing

Di dalam pengembangan software, pengujian pada umumnya sangat diperlukan sebelum perangkat lunak tersebut diberikan kepada kalayak ramai. Pengujian yang di kenal dengan alpha testing sering dilakukan di bawah suatu debugger atau dengan hardware-assisted yang debugging untuk menangkap bugs dengan cepat. Teknik ini juga dikenal sebagai white-box testing.

Selanjutnya diserahkan staff pengujian untuk pemeriksaan tambahan di dalam lingkungan yang serupa. Teknik ini dikenal juga dengan black-box testing dan sering di sebut langkah lanjutan dari white-box testing/alpha testing.

Pada umumnya terdapat dua jenis pengujian yaitu, *Black-Box Testing* dan *White-Box Testing.* Perbedaan yang mencolok diantara keduanya adalah pengujinya. Black-Box dilakukan oleh pengguna perangkat lunak yang mana hanya memperhatikan input dan outputnya saja. Apabila hasil output telah sesuai dengan input yang di uji, maka perangkat lunak talah lulus uji. Sedangkan White-Box testing biasanya dilakukan oleh tim penguji dari pembuat perangkat lunak. Sehingga yang diperhatikan bukan hanya input dan output, melainkan juga proses yang terjadi yang mengakibatkan perubahan dari input menjadi output.

Salah satu contoh dari pengujian white-box adalah basis-path, yang memungkinkan desainer test case mengukur kompleksitas logis dari desain procedural dan menggunakannya sebagai pedoman untuk menetapkan basis set dari jalur eksekusi.



Gambar 2.2 Notasi diagram aliran

1. **Notasi Diagram Air**

   Sebelum metode basis path dapat diperkenalkan, notasi sederhana untuk representasi aliran control yang disebut diagram alir harus diperkenalkan. Grafik alir itu menggambarkan aliran kontrol logika yang menggunakan notasi yang ditunjukkan pada gambar berikut. Masing-masing gagasan tersebut memiliki symbol grafik alir yang sesuai.

2. **Kompleksitas Siklomatis**

   Kompleksitas Siklomatis adalah matrik perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila matrik ini digunakan dalam konteks metode pengujian basis path, maka nilai yang terhitung untuk kompleksitas Siklomatis menentukan jumlah jalur independent dalam basis set suatu program dan memberi batas atas bagi jumlah pengujian yang harus dilakukan untuk memastikan bahwa semua statement telah dieksekusi sedikitnya satu kali.

   Fondasi kompleksitas Siklomatis adalah teori grafik, dan memberi kita matrik perangkat lunak yang sangat berguna. Kompleksitas dihitung dengan salah satu dari 3 cara berikut :

   1. Jumlah region grafik alir sesuai dengan kompleksitas Siklomatis.
   2. Kompleksitas Siklomatis, V(G), untuk grafik alir G ditentukan sebagai V(G) = E – N + 2 dimana E adalah jumlah edge grafik alir dan N adalah jumlah simpul grafik alir.
   3. Kompleksitas Siklomatis, V(G), untuk grafik alir G juga ditentukan sebagai V(G) = P + 1 dimana P adalah jumlah simpul predikat yang diisikan dalam grafik alir G.

3. **Matrik Grafis**

   Matrik Grafis adalah matrik bujur sangkar yang ukurannya (jumlah baris dan koilom sama) dengan jumlah simpul pada grafik alir. Masing-masing baris dan kolom sesuai dengan simpul yang diidentifikasikan, dan entri matrik sesuai dengan edge diantara simpul. Penggunaan matrik Grafis snagat membantu pengujian basis path, struktur data pada perangkat lunak.

2.8.2 Betha testing

Mengeluarkan pruduk betha-version kepada kelompok-kelompok tertentu untuk dilakukan pengujian sehingga dapat mengetahui bugs yang ada pada perangkat lunak. Biasanya, beta-versions dibuat terbatas dan tersedia untuk publik terbuka yang bertujuan untuk meningkatkan jumlah para pemakai di masa yang akan datang[15].

2.9 Skala Likert

Skala likert digunakan untuk mengatur sikap, pendapat, dan persepsi seseorang atau sekelompok orang tentang fenomena sosial. Dalam penelitian, fenomena sosial ini telah ditetapkan secara spesifik oleh peneliti, yang selanjutnya disebut sebagai variabel penelitian.

Dengan Skala Likert, untuk variabel yang akan diukur dijabarkan menjadi indikatot variabel. Kemudian indikator tersebut dijadikan sebagai titik tolak untuk menyusun item-item instrumen yang dapat berupa pernyataan atau pertanyaan.

Jawaban setiap item instrumen yang menggunakan Skala *Likert* mempunyai gradasi dari sangat positif sampai sangat negatif, yang dapat berupa kata-kata antara lain:

a. Sangat Setuju	a. Selalu
b. Setuju	b. Sering
c. Ragu-Ragu	c. Kadang-kadang
d. Tidak Setuju	d. Tidak Pernah
e. Sangat Tidak Setuju	

a. Sangat positif	a. Sangat baik
b. Positif	b. Baik
c. Negatif	c. Tidak baik
d. Sangat negatif	d. Sangat tidak baik

Untuk keperluan analisis kuantitatif, maka jawaban itu dapat diberi skor, misalnya:

1. Setuju/ selalu/ sangat positif diberi skor 5
2. Setuju/ sering/ positif diberi skor 4
3. Ragu-ragu/ kadang-kadang/netral diberi skor 3
4. Tidak setuju/ hampir tidak pernah/ negatif diberi skor 2
5. Sangat tidak setuju/tidak pernah diberi skor 1

Instrument penelitian yang menggunakan skala likert dapat dibuat dalam bentuk ceklis atau pilihan ganda.

Contoh bentuk pilihan ganda :

Berilah salah satu jawaban terhadap pertanyaan berikut sesuai dengan pendapat anda, dengan cara memberi tanda lingkaran pada jawaban yang tersedia.

Prosedur kerja yang baru itu akan segera diterapkan dilembaga anda?

a. Sangat setuju
b. Setuju
c. Ragu-ragu/netral
d. Tidak setuju
e. Sangat tidak setuju

SS = Sangat setuju	diberi skor	5
ST = Setuju	diberi skor	4
RG = Ragu-ragu	diberi skor	3
TS = Tidak setuju	diberi skor	2
STS = Sangat tidak setuju	diberi skor	1

Kemudian dengan teknik pengumpulan data angket, maka instrument tersebut misalnya diberikan kepada 100 orang karyawan yang diambil secara random. Dari 100 orang pegawai setelah dilakukan analisis misalnya :

25 orang menjawab SS

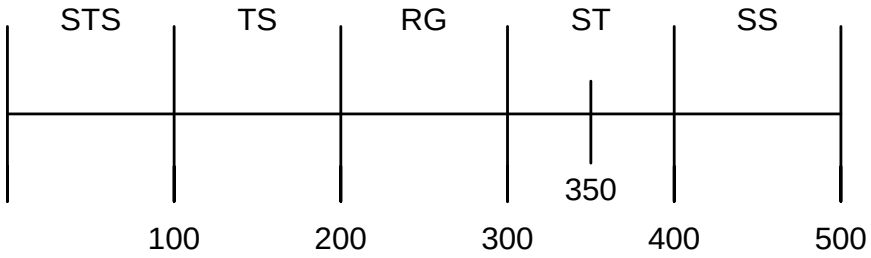
40	orang menjawab	ST
5	orang menjawab	RG
20	orang menjawab	TS
10	orang menjawab	STS

Berdasarkan skor yang telah ditetapkan dapat dihitung sebagai berikut :

Jumlah skor untuk 25 orang yang menjawab SS	=	25 x 5	=	125
Jumlah skor untuk 40 orang yang menjawab ST	=	40 x 4	=	160
Jumlah skor untuk 5 orang yang menjawab ST	=	5 x 3	=	15
Jumlah skor untuk 20 orang yang menjawab TS	=	20 x 2	=	40
Jumlah skor untuk 10 orang yang menjawab STS	=	10 x 1	=	10
Jumlah total			=	350

Jumlah skor ideal (kriterium) untuk seluruh item = 5 x 100 = 500 (seandainya semua menjawab SS). Jumlah skor yang diperoleh dari penelitian = 350. Jadi berdasarkan data itu maka tingkat persetujuan terhadap metode kerja baru itu = (350 : 500) x 100% = 70% dari yang diharapkan (100%).

Secara kontinum dapat digambarkan sebagai berikut :



Gambar 2. 1 Gambar kontinum dari contoh skala *likert*

Jadi berdasarkan data yang diperoleh dari 100 responden maka rata-rata 350 terletak pada daerah setuju.