BAB II

TINJAUAN PUSTAKA

II.1. Perancangan

Menurut Al-Bahra (2005:39), perancangan adalah suatu tahapan yang memiliki tujuan untuk mendesign sistem baru yang dapat menyelesaikan masalah – masalah yang dihadapi perusahaan yang diperoleh dari pemilihan alternatif sistem yang terbaik. Kegiatan yang dilakukan dalam tahap perancangan ini meliputi perancangan *output*, *input* dan *file*.

Sedangkan definisi perancangan menurut John Burch dan Gary Grudnitski yang telah diterjemahkan oleh Jogiyanto Hartono (2005:196) dalam buku yang berjudul *Analisis dan Desain Sistem Informasi* menyebutkan sebagai tahapan setelah analisa siklus pengembangan sistem, pendefinisian dari kebutuhan-kebutuhan fungsional dan persiapan untuk rancang bangun implementasi, serta menggambarkan bagaimana suatu sistem dibentuk.

Berdasarkan penjelasan di atas penulis dapat mengambil kesimpulan bahwa perancangan merupakan kegiatan mendesain sistem baru yang bertujuan untuk menyelesaikan masalah yang dihadapi perusahaan atau suatu kegiatan yang memiliki tujuan untuk mendesain sistem baru yang dapat menyelesaikan masalah-masalah yang dihadapi perusahaan yang diperoleh dari pemilihan alternatif sistem yang terbaik.

II.2. Aplikasi

Aplikasi berasal dari kata *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. (Sobri : 2013)

Perangkat lunak aplikasi adalah suatu sub kelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan tugas yang diinginkan pengguna. Biasanya dibandingkan dengan perangkat lunak sistem yang 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. Biasanya dibandingkan dengan perangkat lunak sistem yang mengintegrasikan berbagai kemampuan komputer, tapi tidak secara langsung menerapkan kemampuan tersebut untuk mengerjakan suatu tugas yang menguntungkan pengguna. (Prajna Deshanta Ibnugraha : 2008)

Aplikasi adalah penggunaan dalam suatu komputer, instruksi atau pernyataan yang disusun sedemikian rupa sehingga komputer dapat memproses *input* menjadi *output* . (Yandi Hotmangatur Hutajulu, 2013)

Dari definisi aplikasi di atas dapat penulis simpulkan pengertian aplikasi adalah suatu program (*software*) yang ditulis atau dirancang untuk menangani masalah tertentu.

II.3. *Game*

Menurut Kimpraswil (dalam As'adi Muhammad, 2009: 26) mengatakan bahwa definisi permainan adalah usaha olah diri (olah pikiran dan olah fisik) yang sangat bermanfaat bagi peningkatan dan pengembangan motivasi, kinerja, dan prestasi dalam melaksanakan tugas dan kepentingan organisasi dengan lebih baik. Lain halnya dengan Joan Freeman dan Utami Munandar (dalam Andang Ismail, 2009: 27) mendefinisikan prmainan sebagai suatu aktifitas yang membantu anak mencapai perkembangan yang utuh, baik fisik, intelektual, sosial, moral, dan emosional. (Yandi Hotmangatur Hutajulu, 2013: 4).

Gambar II.1. Game Smart Brain Kids

(Sumber : Arif Dwi Sutanto, 2013; 11)

Penulis menyimpulkan bahwa *game* atau permainan biasanya dilakukan untuk kesenangan dan kadang – kadang digunakan sebagai alat pendidikan. Untuk membuat sebuah *game*, terlebih dahulu pembuat *game* harus membuat deskripsi yang menceritakan *game* yang akan dibuat. Selain itu dibutuhkan juga *design game* yang sederhana untuk mempermudah pembuatan *game*. Dari *design* yang telah dibuat dapat diketahui semua elemen – elemen yang dibutuhkan dalam

pembuatan *game*, misalnya karakter *user*, karakter musuh, animasi serangan dan sebagainya.

II.3.1. Jenis-Jenis Games

Menurut Marc Saltzman (2000:1) ada beberapa jenis *game* seperti :

a. *Action Game*, secara umum lebih bergantung pada koordinasi tangan atau mata dibandingkan cerita atau strategi. Pada umunya mengandalkan ketangkasan dan refleks.

Gambar II.2. Game Timun Mas

(Sumber : Fiolin Theresia Sumaco, 2013; 5)

b. *Strategy*, *game* strategi menekankan pada pemikiran dan perencanaan logis. *Game* strategi cenderung menitikberatkan pada manajemen sumber daya dan waktu yang biasanya didahulukan sebelum respon cepat dan keterlibatan karakter. Perencanaan dan eksekusi taktis sangat penting, dan pencipta *game* biasanya menempatkan kemampuan pembuatan keputusan dan pengiriman perintah ke tangan pemain. Bertentangan dengan *game* bersifat *turn-base* seperti *Civilization* buatan *Microprose* atau He*roes of Might and Magic* buatan 3DO, *game real-time*. *Strategy* menambahkan elemen aktif dan memaksa pemain untuk merespon beberapa kejadian yang terjadi dalam waktu

yang hampir bersamaan. Contohnya Starcraft buatan Blizzard dan *Age of Empire* buatan *Ensemble* Studios.

c. *Adventure*, melibatkan perjalanan dan ekspedisi dari sebuah eksplorasi dan pemecahan teka-teki. *Game* seperti ini biasanya mempunyai jalan cerita yang linier dimana pemain sebagai protagonist harus menyelesaikan sebuah tujuan utama melewati interaksi karakter dan manipulasi inventaris.

Gambar II.3. Game Petualangan Paperu

(Sumber : Sigit Wicaksono, 2013; 5)

d. *Role Playing Games* (RPGs), mirip dengan *adventure game*, tetapi lebih bergantung pada pembangunan dan pengembangan karakter (biasanya disertai dengan statistik pemain), percakapan, dan strategi bertempur dibanding pemecahan teka-teki. Dunia fantasi yang luas dan *epic quest* dengan NPCs (*non-player characters*) merupakan sesuatu yang umum, dan jalan cerita tidak selalu linier seperti *adventure game*. *Side q*uest merupakan hal yang tidak langka bagi RPGs.

e. Sports, menstimulasi sebuah permainan perorangan atau kelompok dari sudut pandang instruksional atau pemain. Realita merupakan sesuatu yang paling penting, sama seperti ketangkasan dan strategi.

f. *Simulations* atau *Sims*, secara nyata menstimulasikan sebuah objek atau proses yang dianimasikan maupun tidak. Sebagian besar, *Sims* menempatkan pemain

pada sudut pandang 3D orang pertama, atau menciptakan kembali mesin-mesin seperti pesawat, tank, helikopter, dan kapal selam.

\

Gambar II.4. Game Finite State Machine

(Sumber : Silvia Rostianingsih, dkk, 2012; 3)

g. Puzzle atau "Classic" Games, mencakup permainan-permainan yang lebih tua, bersejarah seperti kartu, permainan papan, trivia, atau kata-kata.

Gambar II.5. Gambar Puzzle Kata

(Sumber : Katharina Candra Puspita, dkk, 2010; 4)

II.4. *Game Color Memory*

Memori atau daya ingat adalah kemampuan mengingat kembali pengalaman yang telah lampau. Secara fisiologis, ingatan adalah hasil perubahan

kemampuan penjalaran sinaptik dari satu neuron ke neuron berikutnya, sebagai akibat dari akivitas neural sebelumnya. Perubahan ini kemudian menghasilkan jaras-jaras baru atau jaras-jaras yang terfasilitasi untuk membentuk penjalaran sinyal-sinyal melalui lintasan neural otak. Anak akan belajar banyak hal, warna, bentuk, angka, huruf. Pengetahuan yang diperoleh dari cara ini biasanya mengesankan bagi anak dibandingkan yang dihafalkan. Anak dapat belajar konsep dasar, binatang, alam sekitar, buah-buahan, alfabet dan lain-lain (Danawati Safitri: 2014: 2)

Warna adalah spektrum tertentu yang terdapat di dalam suatu cahaya sempurna (berwarna putih). Identitas suatu warna ditentukan panjang gelombang cahaya tersebut. Sebagai contoh warna biru memiliki panjang gelombang 460 nanometer. (Septian Eka Dyta: 2012: 3)

Jadi, menurut Penulis permainan *Color Memory* adalah suatu permainan yang dimainkan oleh seseorang untuk mengingat warna sesuai petunjuk dari permainan yang dirancang. Hal ini bermanfaat untuk meningkatkan daya ingat seseorang dan sebagai wadah hiburan bagi pemain yang memainkan permainan ini.

II.5. Metode *Brute Force*

Kata algoritma diambil dari nama ilmuwan muslim dari Uzbekistan Abu Ja'far Muhammad bin Musa Al-Khuwārizmi (780-846M), sebagaimana tercantum pada terjemahan karyanya dalam bahasa latin dari abad ke-12 "*Algorithmi de numero Indorum*". Pada awalnya kata algorisma adalah istilah yang merujuk

kepada aturan-aturan aritmetis untuk menyelesaikan persoalan dengan menggunakan bilangan numerik arab sebenarnya dari India, seperti tertulis pada judul di atas. Pada abad ke- 18, istilah ini berkembang menjadi algoritma, yang mencakup semua prosedur atau urutan langkah yang jelas dan diperlukan untuk menyelesaikan suatu permasalahan. Pemecahan sebuah masalah pada hakekatnya adalah menemukan langkah-langkah tertentu yang jika dijalankan efeknya akan memecahkan masalah tersebut. (Yandi Hotmangatur Hutajulu: 2013: 174)

Brute Force (menurut Rinaldi Munir) adalah sebuah pendekatan langsung (*straight forward*) untuk memecahkan suatu masalah, yang biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Pada dasarnya algoritma *Brute Force* adalah alur penyelesaian suatu permasalahan dengan cara berpikir yang sederhana dan tidak membutuhkan suatu permikiran yang lama. Sebenarnya, algoritma *Brute Force* merupakan algoritma yang muncul karena pada dasarnya alur pikir manusia adalah *Brute Force* (langsung/*to the point*). (Yandi Hotmangatur Hutajulu: 2013: 174)

Algoritma *Brute Force* adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma *Brute Force* memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*). Algoritma ini memiliki moto yaitu: *Just Do It!* (Siswanto: 2014: 48)

Berikut ini adalah beberapa keunggulan algoritma *Brute Force* :

1. Algoritma *Brute Force* umumnya tidak “cerdas” dan tidak mangkus, karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya. Kadang-kadang algoritma Brute Force disebut juga algoritma naif (*naïve algorithm*).
2. Algoritma *Brute Force* seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus.
3. Untuk masalah yang ukurannya kecil, kesederhanaan *Brute Force* biasanya lebih diperhitungkan daripada ketidakmangkusannya. Algoritma *Brute Force* sering digunakan sebagai basis bila membandingkan beberapa alternatif algoritma yang mangkus.
4. Meskipun *Brute Force* bukan merupakan teknik pemecahan masalah yang mangkus, namun teknik *Brute Force* dapat diterapkan pada sebagian besar masalah. Agak sukar menunjukkan masalah yang tidak dapat dipecahkan dengan teknik *Brute Force*. Bahkan ada masalah yang hanya dapat dipecahkan secara *Brute Force*.
5. Beberapa pekerjaan mendasar di dalam komputer dilakukan secara *Brute Force*, seperti menghitung jumlah dari *n* buah bilangan, mencari elemen terbesar di dalam tabel, dan sebagainya.
6. Selain itu, algoritma *Brute Force* seringkali lebih mudah diimplementasikan daripada algoritma yang lebih canggih, dan karena kesederhanaannya, kadang-kadang algoritma *Brute Force* dapat lebih mangkus (ditinjau dari segi implementasi). (Siswanto: 2014)

Menurut Yandi Hotmangatur Hutajulu (2013: 3) , beberapa karakteristik dari algoritma *Brute Force* dapat dijelaskan sebagai berikut :

1. Membutuhkan jumlah langkah yang banyak dalam menyelesaikan suatu permasalahan sehingga jika diterapkan menjadi suatu algoritma program aplikasi akan membutuhkan banyak memori.
2. Digunakan sebagai dasar dalam menemukan suatu solusi yang lebih efektif.
3. Banyak dipilih dalam penyelesaian sebuah permasalahan yang sederhana karena kemudahan cara berpikirnya.
4. Pada banyak kasus, algoritma ini banyak dipilih karena hampir dapat dipastikan dapat menyelesaikan banyak persoalan yang ada.
5. Digunakan sebagai dasar bagi perbandingan keefektifan sebuah algoritma.

Dalam beberapa kasus tertentu algoritma *Brute Force* hampir sama dengan *Exhaustive Search*. *Exhausitve Search* yang merupakan teknik pencarian solusi secara *Brute Force* pada masalah yang melibatkan pencarian elemen dengan sifat khusus. Biasanya elemen tersebut berada di antara objek – objek kombinatorik seperti permutasi, kombinasi, atau himpunan bagian dari sebuah himpunan. Berdasarkan definisi ini, maka dapat ditarik kesimpulan bahwa *Exhaustive Search* adalah *Brute Force* juga. Oleh karena itu *Exhaustive Search* adalah salah satu implementasi dari *Brute Force* dalam kasus pencarian. Masalah–masalah dalam *Exhaustive Search* dengan penerapan algoritma *Brute Force* dapat dijelaskan sebagai berikut:

1. Enumerasi setiap solusi yang mungkin dengan cara yang sistematis.

2. Evaluasi setiap kemungkinan solusi yang ditemukan satu per satu, meskipun terdapat beberapa kemungkinan ditemukannya solusi yang tidak layak atau bahkan terdapat kemungkinan–kemungkinan solusi terbaik yang telah ditemukan dan dievaluasi.
3. Bila pencarian sudah sampai pada tujuan, maka pilih solusi yang terbaik. (Yandi Hotmangatur Hutajulu: 2013)

Brute Force merupakan algoritma pencarian string termudah. Dengan asumsi bahwa teks berada di dalam *array* T[1..n] dan *pattern* berada di dalam *array* P[1..m], maka algoritma *Brute Force* pencocokan *string* adalah sebagai berikut (Munir: 2004)

1. Mula-mula *pattern* P dicocokkan pada awal teks T.
2. Dengan bergerak dari kiri ke kanan, bandingkan setiap karakter di dalam pattern P dengan karakter yang bersesuaian di dalam teks T sampai :
    a. Semua karakter yang dibandingkan cocok atau sama (pencarian berhasil) atau
    b. Dijumpai sebuah ketidakcocokan karakter (pencarian belum berhasil)
3. Bila *pattern* P belum ditemukan kecocokannya dan teks T belum habis, geser *pattern* P satu karakter ke kanan dan ulangi langkah 2.

Persoalan pencocokan string dapat dirumuskan sebagai berikut:

1. Teks (*text*), yaitu (*long*) *string* yang panjangnya n karakter
2. *Pattern*, yaitu *string* dengan panjang m karakter (m < n) yang akan dicari di dalam teks (Nisa Hidayani: 2012)

II.6. Sekilas Tentang Microsoft Visual Studio 2010

Visual Basic mengacu pada dua hal, yang pertama Visual dan yang kedua Basic. Kata "Visual" menunjukkan cara yang digunakan untuk membuat *Graphical User Interface* (GUI). Dengan cara ini *programmer* tidak lagi menuliskan instruksi pemrograman dalam kode – kode baris, tetapi secara mudah *progammer* dapat melakukan "*drag and drop*" objek-objek yang akan digunakan. Kata "Basic" merupakan bagian bahasa *BASIC (Beginners All Purpose Symbolic Instruction Code),* yaitu sebuah bahasa pemrograman yang dalam sejarahnya sudah banyak digunakan oleh para *programmer* untuk menyusun aplikasi.

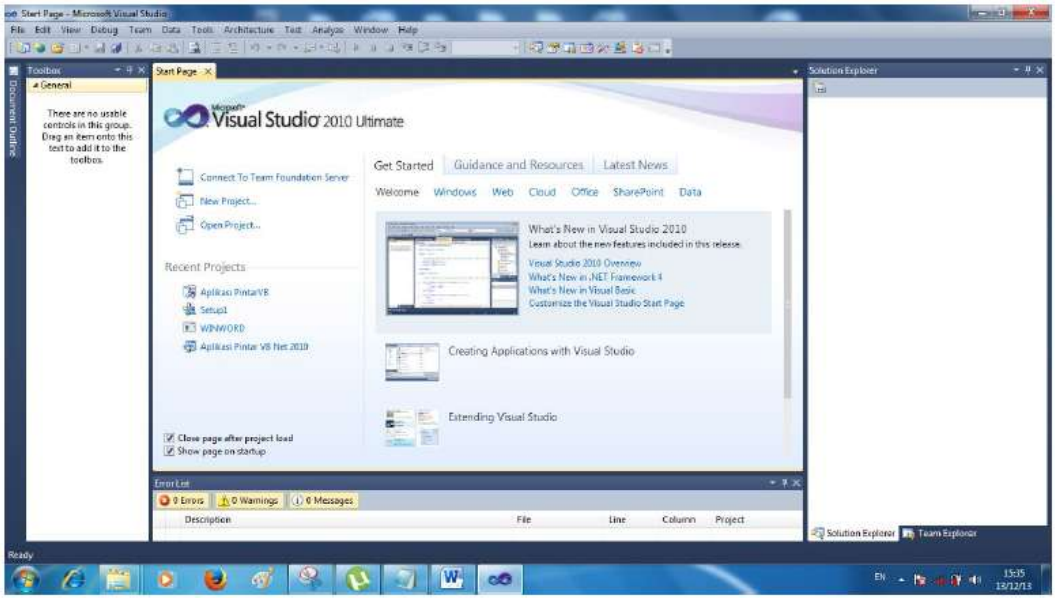
Microsoft Visual Basic 2010 adalah salah satu komponen Microsoft Visual Studio 2010. *Software* ini diluncurkan Microsoft pada tanggal 12 April 2010 dengan nama kode Dev10 dan menggunakan .Net Framework 4.0. Integrated Development Environment (IDE) pada Visual studio 2010 telah didesain ulang sehingga lebih enak dipandang dan digunakan *programmer*.

Microsoft Visual Studio 2010 merupakan sebuah *IDE (Integrated Development Environment)* yang dikembangkan oleh microsoft. *IDE* ini mencakup semua bahasa pemrograman berbasis *.NET framework* yang dikembangkan oleh *Microsoft*. Keunggulan *Microsoft Visual Studio 2010* ini antara lain adalah *support* untuk *Windows 8*, editor baru dengan *WPF (Windows Presentation Foundation)*, dan banyak peningkatan fitur lainya.

Untuk *code editor*-nya, Visual Basic 2010 telah menambah fitur *highlights reference*. Ketika satu simbol / kode dalam bahasa pemrogramannya dipilih, maka simbol / kode yang sama, meskipun penggunaannya berbeda akan terlihat

berwarna sama. Misal jika kode *math* dipilih, seluruh kode *math* akan terlihat berwarna sama.

Berikut ini gambar tampilan dari jendela Microsoft Visual Studio 2010 :



Gambar II.6. Tampilan Jendela Microsoft Visual Basic 2010

II.7. Alat Bantu Perancangan Sistem

Adapun alat bantu yang digunakan dalam perancangan atau pembangunan sistem yang digunakan dalam penelitian umumnya berupa gambaran atau diagram yang tertera dan dijelaskan di bawah ini.

II.7.1. *Unified Modelling Language (UML)*

Unified Modelling Language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, piranti lunak. UML menawarkan

sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML dapat dibuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan class dan operation dalam konsep dasarnya, maka lebih cocok untuk penulisan piranti lunak dalam bahasa berorientasi objek seperti *C++, Java, atau VB. NET* (Prastuti Sulistyorini : 2009:23).

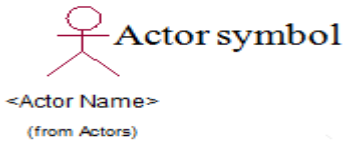
Use Case adalah deskripsi dari sebuah sistem dari perspektif pengguna. *Use Case* bekerja dengan cara mendeskripsikan *tipikal* interaksi antar *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai". *Use Case Diagram* menggambarkan fungsionalitas 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 apabila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan *client*, dan merancang *test case* untuk semua *feature* yang ada pada *system.*

Untuk menggambarkan analisa dan desain diagram, UML memiliki seperangkat notasi yang akan digunakan ke dalam tiga kategori diatas yaitu

struktur diagram, behaviour diagram dan interaction diagram. Berikut beberapa notasi dalam UML diantaranya :

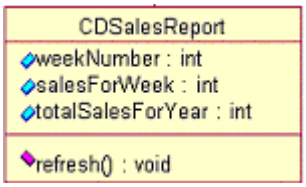
1. *Actor*, menentukan peran yang dimainkan oleh *user* atau sistem lain yang berinteraksi dengan subjek. *Actor* adalah segala sesuatu yang berinteraksi langsung dengan sistem aplikasi komputer, seperti orang, benda atau lainnya. Tugas *actor* adalah memberikan informasi kepada sistem dan dapat memerintahkan sistem untuk melakukan sesuatu tugas.



Gambar II.7. Notasi *Actor*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

2. *Class diagram*, notasi utama dan yang paling mendasar pada diagram UML adalah notasi untuk mempresentasikan suatu *class* beserta dengan atribut dan operasinya. *Class* adalah pembentuk utama dari sistem berorientasi objek.

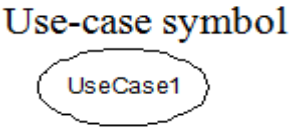


Gambar II.8. Notasi *Class*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

3. *Use Case* dan *use case specification*, *use case* adalah deskripsi fungsi dari sebuah sistem perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem

dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut skenario. *Use case* merupakan awal yang sangat baik untuk setiap fase pengembangan berbasis objek, *design*, *testing*, dan dokumentasi yang menggambarkan kebutuhan sistem dari sudut pandang di luar sistem. Perlu diingat bahwa *use case* hanya menetapkan apa yang seharusnya dikerjakan oleh sistem, yaitu kebutuhan fungsional sistem dan tidak untuk menentukan kebutuhan non-fungsional, misalnya: sasaran kinerja, bahasa pemrograman dan lain sebagainya.



Gambar II.9. Notasi *Use Case*

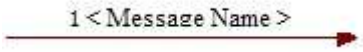
Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

4. *Realization*, menunjukkan hubungan bahwa elemen yang ada di bagian tanpa panah akan merealisasikan apa yang dinyatakan oleh elemen yang ada di bagian dengan panah.

Gambar II.10. Notasi *Relaization*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

5. *Interaction*, digunakan untuk menunjukkan baik aliran pesan atau informasi antar obyek maupun hubungan antar obyek.



Gambar II.11. Notasi *Interaction*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

6. *Dependency*, merupakan relasi yang menunjukan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. Terdapat 2 *stereotype* dari *dependency*, yaitu *include* dan *extend*. *Include* menunjukkan bahwa suatu bagian dari elemen (yang ada digaris tanpa panah) memicu eksekusi bagian dari elemen lain (yang ada di garis dengan panah). *Extend* menunjukkan bahwa suatu bagian dari elemen di garis tanpa panah bisa disisipkan ke dalam elemen yang ada di garis dengan panah.

Gambar II.12. Notasi *Dependancy*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

7. *Note*, digunakan untuk memberikan keterangan atau komentar tambahan dari suatu elemen sehingga bisa langsung terlampir dalam model. *Note* ini bisa disertakan ke semua elemen notasi yang lain.

Gambar II.13. Notasi *Interaction*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

8. *Association*, menggambarkan navigasi antar *class* (*navigation*), berapa banyak obyek lain yang bisa berhubungan dengan satu obyek (*multiplicity* antar *class*) dan apakah suatu *class* menjadi bagian dari *class* lainnya (*aggregation*).

Gambar II.14. Notasi *Association*

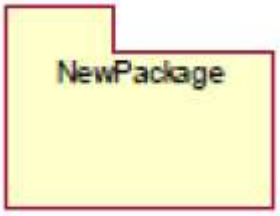
Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

9. Generalization, menunjukkan hubungan antara elemen yang lebih umum ke elemen yang lebih spesifik.

Gambar II.15. Notasi *Generalization*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

10. *Package*, adalah mekanisme pengelompokkan yang dgunakan untuk menandakan pengelompokkan elemen-elemen model.



Gambar II.16. Notasi *Package*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

11. *Interface*, merupakan kumpulan operasi berupa implementasi dari suatu *class*. Atau dengan kata lain implementasi operasi dalam *interface* dijabarkan oleh operasi di dalam *class*.

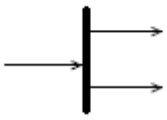
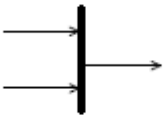
Gambar II.17. Notasi *Interface*

Sumber : Haviluddin, Jurnal Informatika Mulawarman, 2011

II.7.2. *Activity Diagram*

Activity Diagram adalah teknik untuk mendiskripsikan logika prosedural, proses bisnis dan aliran kerja dalam banyak kasus. *Activity Diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity diagram* bisa mendukung perilaku paralel sedangkan *flowchart* tidak bisa (Prastuti Sulistyorini : 2009:23).

Berikut adalah simbol-simbol yang sering digunakan pada saat pembuatan *activity diagram* yaitu :

Simbol	Keterangan
	Start Point
	End Point
	Activities
	Fork (Percabangan)
	Join (Penggabungan)
	Decision/Split Merge
Swimlane	Sebuah cara untuk mengelompokkan activity berdasarkan Actor (mengelompokkan activity dalam sebuah urutan yang sama)

Gambar II.18. Simbol *Activity Diagram*

Sumber : http://kk.mercubuana.ac.id/files/15024-5-600173869778.pdf