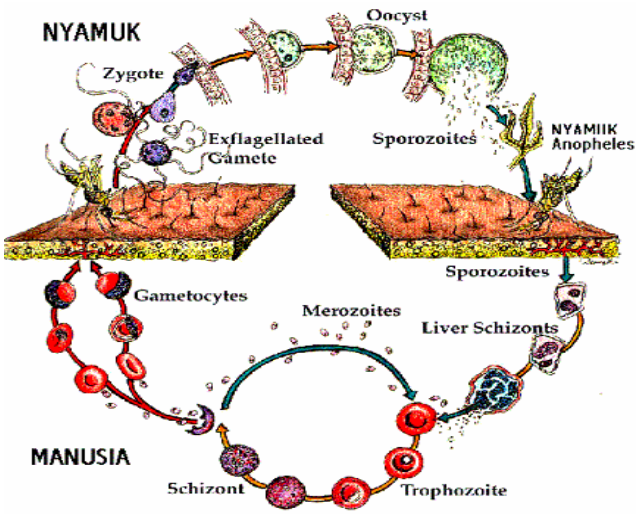
BAB II
LANDASAN TEORI

2.1 Malaria

Malaria adalah penyakit infeksi yang disebabkan oleh *protozoa* parasit kelompok *Plasmodium* yang penularannya terjadi melalui gigitan nyamuk *Anopheles*. Ciri utama *genus* ini adalah siklus hidup terjadi dalam dua inang yang berbeda, dapat dilihat pada Gambar 2.1. Siklus seksual terjadi dalam tubuh nyamuk *Anopheles* betina, yang bertindak sebagai vektor perantara penyebaran parasit. Siklus aseksual terjadi dalam tubuh manusia. Gejala awal yang sering terjadi adalah demam, sakit kepala, mual dan muntah, biasanya muncul 10 sampai 15 hari setelah terinfeksi. Bila tidak mendapatkan pengobatan yang tepat, malaria dapat menyebabkan keseriusan dan sering berakhir dengan kematian.



Gambar 2.1. Siklus Hidup Parasit Penyebab Malaria [7]

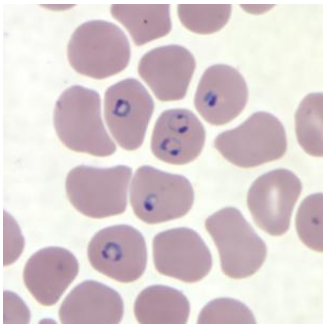
Pada tanggal 25 April 2007 dalam satu sidang *World Health Assemby*, seluruh negara anggota WHO menyatakan komitmennya untuk memberantas malaria sampai titik eliminasi [8]. Oleh karena itu, tanggal tersebutlah dijadikan tonggak sejarah dan dijadikan tanggal peringatan Hari Malaria Sedunia.

2.1.1 Jenis-jenis *Plasmodium* Malaria

Malaria memiliki 5 jenis *plasmodium* yang bisa terinfeksi pada makhluk hidup, yaitu sebagai berikut :

a. *Plasmodium Falciparum*

Plasmodium yang ditemukan di seluruh dunia di daerah tropis dan subtropis. Diperkirakan bahwa setiap tahun sekitar 1 juta orang dibunuh oleh *P. falciparum*, terutama di Afrika di mana spesies ini mendominasi. *P. falciparum* dapat menyebabkan malaria berat karena reprdouksi kelipatan yang cepat dalam darah, dan dengan demikian dapat menyebabkan kehilangan darah yang parah (anemia). Selain itu, parasit yang terinfeksi bisa menyumbat pembuluh darah kecil. Ketika ini terjadi di otak, hasilnya malaria serebral, komplikasi yang bisa berakibat fatal.

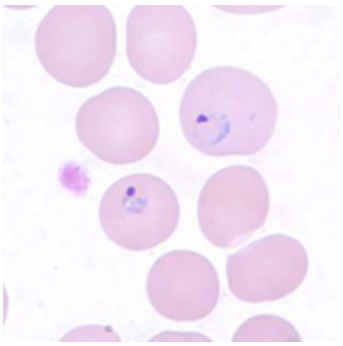


Gambar 2.2. Bentuk *Plasmodium Falciparum* [9]

b. *Plasmodium Vivax*

P. vivax kebanyakan ditemukan di Asia, Amerika Latin, dan di beberapa bagian Afrika. Karena kepadatan penduduknya, terutama di Asia parasit malaria ini termasuk yang paling lazim dijumpai manusia. P. vivax memiliki tahap hati dorman (*hypnozoites*) yang dapat mengaktifkan dan menyerang darah beberapa bulan atau tahun setelah gigitan nyamuk menginfeksi. Parasit memasuki sel-sel hati, di mana membelah untuk membentuk *schizonts* terdiri dari banyak *merozoit*. Setelah 48 jam atau

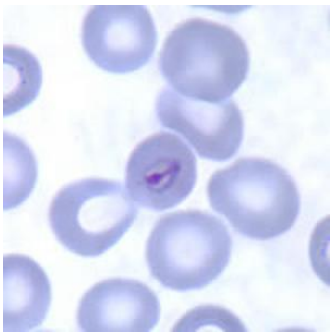
lebih, cukup bagi merozoit untuk bereproduksi dan menyebar ke sel darah merah, yang mengakibatkan demam dan menggigil.



Gambar 2.3. Bentuk *Plasmodium Vivax* [9]

c. *Plasmodium Malariae*

P. malariae ditemukan di seluruh dunia, adalah satu-satunya spesies parasit malaria yang memiliki siklus *quartan* (siklus tiga hari). (Tiga spesies lain memiliki siklus dua hari.) Jika tidak diobati, *P. malariae* menyebabkan infeksi penyakit malaria yang tahan lama, infeksi kronis yang dalam beberapa kasus dapat berlangsung seumur hidup. Pada beberapa pasien kronis terinfeksi *P. malariae* dapat menyebabkan komplikasi serius seperti sindrom nefrotik.

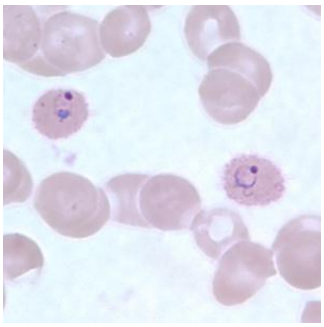


Gambar 2.4. Bentuk *Plasmodium Malariae* [9]

d. *Plasmodium Ovale*

P. ovale kebanyakan ditemukan di Afrika (terutama Afrika Barat) dan kepulauan Pasifik Barat. Secara biologis dan morfologis sangat mirip dengan *P. vivax*. Namun berbeda dari *P. vivax*, *plasmodium* ini dapat

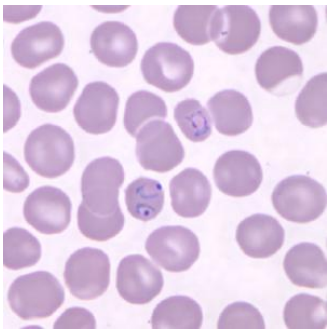
menginfeksi individu yang negatif untuk golongan darah Duffy, yang merupakan kasus bagi banyak penduduk Afrika.



Gambar 2.5. Bentuk *Plasmodium Ovale* [9]

e. *Plasmodium Knowlesi*

P. knowlesi ini ditemukan di seluruh Asia Tenggara sebagai patogen alami kera ekor panjang dan babi. Baru-baru ini terbukti menjadi penyebab signifikan malaria *zoonosis* di wilayah itu, terutama di Malaysia. *P. knowlesi* memiliki siklus replikasi 24 jam dan begitu cepat dapat berkembang menyebabkan infeksi yang parah.



Gambar 2.6. Bentuk *Plasmodium Knowlesi* [9]

Plasmodium Falciparum dan *Plasmodium Vivax* merupakan jenis yang paling sering dijumpa, namun yang paling mematikan adalah jenis *Plasmodium Falciparum*.

2.1.2 Pengendalian dan Pencegahan Malaria

Penanggulangan malaria dilakukan secara komprehensif dengan upaya promotif, preventif, dan kuratif. Hal ini bertujuan untuk menurunkan angka kesakitan dan kematian serta mencegah KLB (kejadian luar biasa). Untuk mencapai hasil yang optimal upaya preventif dan kuratif tersebut harus dilakukan dengan berkualitas dan terintegrasi dengan program lainnya.

Beberapa cara mengendalikan dan mencegah penyebaran penyakit malaria adalah sebagai berikut :

1. Mencegah kontaminasi dari lingkungan misalnya dengan menggunakan obat antimalaria seperti semprotan DDT. Penderita yang sedang dalam pengobatan harus dihindari dari *re-infection*. Perlu diperhatikan pula sanitasi lingkungan.
2. Memutuskan siklus hidup parasit. *Protozoa* dan larva yang infektif biasanya resisten terhadap terhadap senyawa kimia yang toksik. Biasanya dengan memggunakam program 3M yaitu menguras, menutup dan mengubur.
3. Mengendalikan perkembangan vektor perantara dengan menggunakan insektisida atau mengubah kondisi lingkungan agar nyamuk Anopheles sebagai spesies target tidak dapat bertahan hidup dalam habitat baru.
4. Mencegah terjadinya infeksi dengan menggunakan kelambu untuk mengindari gigitan nyamuk.
5. Mencegah pematangan parasit melalui kemoprofilaksis dan vaksinasi.

2.2 *Game*

Game adalah adalah kegiatan interaktif secara sukarela, di mana satu atau lebih pemain mengikuti aturan yang membatasi perilaku mereka, memberlakukan konflik buatan yang berakhir dengan hasil yang terukur [10]. Awal dari analisis teori *game* secara formal adalah pembelajaran duopoly dari Antoine Cournot pada tahun 1838. Matematikawan Emile Borel menyarankan teori formal dari game pada tahun

1921, yang ditindaklanjuti oleh matematikawan John Von Neumann pada tahun 1928 dalam "*Theory of Parlor Games*". Pada tahun 1944 dipublikasikan *volume monumental "Theory of Games and Economic Behavior"* oleh Von Neumann dan Oskar Morgenstern, teori ini memberikan banyak terminologi dan masalah konfigurasi dasar yang masih digunakan sampai sekarang [11].

2.2.1 Klasifikasi *Game*

2.2.1.1 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 game*s 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 komputer atau sering disebut desktop.
3. *Console games*, yaitu *video game*s 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.

2.2.1.2 Berdasarkan *Genre* Permainanya

1. Aksi *Shooting*, (tembak-tembakan, atau hajar-hajaran bisa juga tusuk-tusukan, tergantung cerita dan tokoh di dalamnya), *video game* jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga *timing*, inti dari *game* jenis ini adalah menembaki musuh.
2. *Fighting* (pertarungan) Ada yang mengelompokan *video game fighting* di bagian Aksi, namun penulis berpendapat berbeda, jenis ini memang

memerlukan kecepatan reflek dan koordinasi mata-tangan, tetapi inti dari *game* ini adalah penguasaan jurus (hafal caranya dan lancar mengeksekusinya), pengenalan karakter dan timing sangatlah penting, *combo*-pun menjadi esensial untuk mengalahkan lawan secepat mungkin. Berbeda seperti *game action* pada umumnya yang hanya melawan komputer saja, pemain jenis *fighting game* ini baru teruji kemampuan sesungguhnya dengan melawan pemain lainnya. *Seri Street Fighter, Tekken, Mortal Combat, Soul Calibur* dan *King of Fighter* adalah contohnya.

3. Petualangan. Bedanya dengan jenis *video game* aksi-petualangan, reflek dan kelihaian pemain dalam bergerak, berlari 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.
4. Strategi. Kebalikan dari *video game* jenis action yang berjalan cepat dan perlu refleks secepat kilat, *video game* jenis strategi, layaknya bermain catur, justru lebih memerlukan keahlian berpikir dan memutuskan setiap gerakan secara hati-hati dan terencana. *Video game* strategi biasanya memberikan pemain atas kendali tidak hanya satu orang tapi minimal sekelompok orang dengan berbagai jenis tipe kemampuan, sampai kendaraan, bahkan hingga pembangunan berbagai bangunan, pabrik dan pusal pelatihan tempur, tergantung dari tema ceritanya.
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 *game* jenis ini adalah juga unsur

game dalam *video game* petualangan maupun *game* edukasi. Tetris, Minesweeper, Bejeweled, Sokoban dan Bomberman.

6. Olahraga. *Game* yang diadopsi dari kegiatan olahraga yang sesungguhnya. Biasanya *game*-nya 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, dll.
7. Edukasi. *Game* ini dirancang untuk mengambungkan sisi interanktif dan menyenangkan dengan pendidikan atau pengajaran tentang keterampilan yang baru baik di bidang kesehatan, tekhnologi, pelajaran umum, bahasa, dll.

2.2.1.3 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*).

2.2.2 *Game* Edukasi

Game edukasi merupakan permainan digital yang dapat memberikan kesempatan untuk bermain melalui lingkungan simulasi dan dapat menjadi bagian integral dari pembelajaran dan pengembangan intelektual [12]. Sampai akhir abad 19 game diasosiasikan dengan hiburan, tetapi setelah mendapatkan pengaruh dari John Dewey pada tahun 1944, game mulai memegang peranan dalam teknologi pengajaran.

2.2.2.1 Tujuan *Game* Edukasi

Game edukasi mampu membantu masyarakat dalam pengembangan akhlak, intelektual, motivasi, keahlian, kecakapan. Agar tujuan pembelajaran menggunakan *game* edukasi ini tepat sasaran, pada tahun 1997 Calvo mengemukakan teori bahwa *game* harus mampu meningkatkan fungsi-fungsi berikut [3]:

1. Pengembangan Motorik

   Game harus banyak melibatkan gerakan untuk meransang keakuratan gerakan, koordinasi gerakan dan kecepatan gerakan.

2. Pengembangan Intelektual

   Game harus melibatkan pemahaman bagaimana sesuatu bekerja, pemecahan masalah, pengambilan strategi, pengambilan keputusan dan lain-lain.

3. Pengembangan Afektif

   Game harus mampu merangsang siswa untuk memahami pengalam kehidupan mereka dan membantu mereka menjadi dewasa.

4. Pengembangan Sosial

   Game harus mampu mengakomodasi interaksi antar pemain dalam bentuk-bentuk simbol dan aturan-aturan sehingga siswa yang bermain.

2.3 Artificial Intelligence

Menurut Simon, A*rtificial Intelligence* (AI) atau kecerdasan buatan merupakan kawasan penelitian, aplikasi dan instruksi yang terkait dengan pemrograman komputer untuk melakukan sesuatu hal yang dalam pandangan manusia adalah cerdas [13]. Kecerdasan diciptakan dan diterapkan ke dalam suatu mesin komputer agar dapat melakukan pekerjaan seperti yang dapat dilakukan manusia. Dari beberapa perspektif, AI dapat dikategorikan sebagai berikut :

1. Dari perspektif kecerdasan, AI berfungsi untuk membuat mesin yang cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dapat dilakukan manusia
2. Dari perspektif bisnis, AI adalah sekelompok alat bantu (*tools*) yang berdayaguna dan metodologi yang menggunakan alat-alat bantu tersebut untuk menyelesaikan masalah masalah bisnis
3. Dari perspektif pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah, dan proses pencarian (*search*).

2.3.1 Teknik-Teknik Dasar Pencarian

Pencarian merupakan salah satu teknik dari AI untuk menyelesaikan beberapa permasalahan. Keberhasilan suatu sistem salah satunya ditentukan oleh kesuksesan dalam pencarian dan pencocokan. Berikut adalah contoh aplikasi yang menggunakan teknik pencarian ini, yaitu :

1. *Game* papan dan puzzle, contoh: tic-tac-toe, catur, dan menara hanoi
2. Penjadwalan dan masalah *routing* pada *travelling salesman problem*
3. Parsing bahasa dan inteprestasinya pada masalah pencarian struktur dan arti kata
4. Logika pemrograman dalam hal pencarian fakta dan implikasinya
5. *Computer vision* dan pengenalan pola
6. Sistem pakar bebasis kaidah (*rule based expert system*)

Pencarian adalah proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan (*state space*). Ruang keadaan merupakan suatu ruang yang terdapat semua keadaan yang mungkin. Kondisi suatu pencarian meliputi :

1. Keadaan sekarang atau awal
2. Keadaan tujuan atau solusi yang dijangkau dan perlu diperiksa apakah telah mencapai sasaran
3. Biaya (*cost*) yang diperoleh dari solusi.

Solusi merupakan suatu lintasan dari keadaan awal sampai keadaan tujuan. Secara umum, proses pencarian dapat dilakukan seperti berikut :

1. Memeriksa keadaan sekarang atau awal
2. Mengeksekusi aksi yang dibolehkan untuk memindahkan ke keadaan berikutnya
3. Memeriksa jika keadaan baru merupakan solusinya. Jika tidak, keadaan baru tersebut menjadi keadaan sekarang dan proses ini diulangi sampai solusi ditemukan atau ruang keadaan habis terpakai.

2.3.2 Algoritma Pencarian (*Search Algorithm*)

Berbagai algoritma untuk pencarian (*search algorithm*) yang ada berbeda satu dengan yang lain dalam hal pengembangan kumpulan node untuk mencapai *goal state*. Perbedaan ini terutama dalam hal cara dan urutan pengembangan node, dan

sangat berpengaruh pada kinerja masing-masing algoritma. Empat kriteria yang menjadi ukuran algoritma pencarian adalah :

1. *Completeness*, apakah algoritma pasti dapat menemukan solusi?
2. *Time Comlexity*, berapa lama waktu yang dibutuhkan untuk menemukan sebuah solusi?
3. *Space Complexity*, berapa memori atau *resource* yang diperlukan untuk melakukan pencarian?
4. *Optimality*, apakah algoritma tersebut dapat menemukan solusi yang terbaik jika terdapat beberapa solusi yang berbeda?

Permasalahan pencarian dapat diselesaikan dengan 2 golongan, yaitu:

1. *Uninformed Search/Blind Search,* merupakan pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state* disebut juga pencarian buta. Beberapa contoh algoritma tersebut antara lain adalah *Breadth First Search, Uniform Cost Search, Depth First Search, Depth Limited Search, Iterative Deepening Search dan Bidirectional Search.*
2. *Informed Search/Heuristic Search* merupakan pencarian solusi dengan adanya informasi tentang biaya (*cost*) yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state*. Dengan informasi tersebut, dapat melakukan pertimbangan untuk mengembangkan atau memeriksa kumpulan node yang mengarah ke *goal state*. Beberapa contohnya adalah *Best First Search, Greedy Search*, *Djikstra*, A* (A *Star*) *Search,* dan *Hill Climbing Search.*

2.3.3 Algoritma A*

Algoritma A* diperkenalkan pertama kali oleh Peter Hart, Nils Nilsson dan Bertram Raphael pada tahun 1968. Algoritma A* merupakan format pencarian heuristik untuk menghitung efisiensi solusi optimal. A* merupakan algortima *Best First Search* yang menggabungkan *Uniform Cost Search* dan *Greedy Best First*

Search [14]. Terdapat terminologi dasar yaitu *strating point,* simpul (*node*), A, *openlist*, *closedlist,* harga (*cost*), halangan (*unwalkable*).

Starting point adalah sebuah terminologi untuk posisi awal sebuah benda. Simpul adalah petak-petak kecil sebagai representasi dari area *pathfinding*. *Openlist* merupakan tempat menyimpan data simpul yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan. *Closedlist* adalah tempat menyimpan data simpul sebelum A yang juga merupakan bagian jalur terpendek yang telah berhasil didapatkan. Sedangkan A adalah simpul yang sedang dijalankan dalam algoritma pencarian jarak terpendek.

Pada algoritma A* terdapat fungsi evaluasi terhadap *node* n adalah f(n) = g(n) + h(n). f adalah fungsi evaluasi yang diperoleh dari penjumlahan nilai g (*actual cost*) dan h (*heuristic cost*). g adalah jumlah nilai setiap simpul dalam jalur terpendek dari *starting point* ke A. h adalah jumlah nilai perkiraan dari simpul ke simpul tujuan.

Pencarian jarak terpendek menggunakan algoritma A* memiliki prinsip yang sama dengan algoritma BFS, hanya saja dengan dua faktor tambahan.

1. Setiap sisi mempunyai *cost* yang berbeda-beda, seberapa besar *cost* untuk pergi dari satu simpul ke simpul yang lain.
2. *Cost* dari setiap simpul ke simpul tujuan bisa diperkirakan. Ini membantu pencarian, sehingga lebih kecil kemungkinan mencari ke arah yang salah.

Cost untuk setiap simpul tidak harus berupa jarak. *Cost* bisa saja berupa waktu bila ingin mencari jalan dengan waktu tercepat untuk dilalui.

Pada Algoritma A* memiliki langkah-langkah sebagai berikut :

1. Masukkan simpul awal ke *Open List*.
2. Ulangi langkah berikut sampai pencarian berakhir.
3. Cari *node* n dengan nilai f(n) paling rendah, dalam *Open List*. *Node* ini akan menjadi *current node*.
4. Keluarkan *current node* dari *Open List* dan masukkan ke *Closed List*.
5. Untuk setiap suksesor dari *current node* lakukan langkah berikut :

- Jika sudah terdapat dalam *Closed List,* abaikan, jika tidak lanjutkan.
- Jika belum ada pada *Open List*, masukkan ke Open List. Simpan *current node* sebagai parent dari suksesor-suksesor ini. Simpan *cost* masing-masing simpul.
- Jika belum ada dalam *Open List*, periksa jika simpul suksesor ini mempunyai nilai lebih kecil dibanding suksesor sebelumnya. Jika lebih kecil, jadikan sebagai *current node* dan ganti *parent node* ini.

6. Walaupun telah mencapai simpul tujuan, jika masih ada suksesor yang memiliki nilai yang lebih kecil, maka simpul tersebut akan terus dipilih sampai bobotnya jauh lebih besar atau mencapai simpul akhir dengan bobot yang lebih kecil dibanding dengan simpul sebelumnya yang telah mencapai simpul tujuan.
7. Pada setiap pemilihan simpul berikutnya, nilai f(n) akan dievakuasi, dan jika terdapat nilai f(n) yang sama maka akan dipilih berdasarkan nilai g(n) terbesar.

2.4 *Object Oriented Programming* (OOP)

Object oriented programming atau yang sering disebut dalam bahasa Indonesia pemrograman berorientasi objek (PBO) merupakan paradigma pemrograman yang berorientasikan kepada objek, diamana OOP memodelkan obyek yang ada di dunia nyata (*real-word objects*) ke dalam software obyek dalam pemrograman [15]. Dalam dunia nyata terdapat contoh beberapa obyek seperti mobil, manusia dan seterusnya. Obyek ini dikarakterisasi oleh atribut dan tingkah lakunya. Dengan deskripsi tersebut, obyek pada dunia nyata dapat diasumsikan sebagai obyek perangkat lunak menggunakan atribut sebagai data dan tingkah laku sebagai method.

Secara umum pemograman berorientasi objek memiliki beberapa keuntungan, yaitu sebagai berikut :

1. Dapat memberi fleksibilitas yang lebih pada *system*.
2. Kemudahan mengubah program karena dibagi-bagi permasalahannya.
3. Digunakan luas dalam teknik piranti skala besar.

4. Lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya.
5. Pendekatan OOP lebih mudah dikembangkan dan dirawat.

Pemrograman borientasi objek menekankan pada konsep-konsep berikut yang merupakan dasar atau ciri khasnya, yaitu :

1. *Class*

   Class adalah kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *"class of monkey"* adalah suatu unit yang terdiri atas definisi data dan fungsi yang menunjuk pada berbagai macam perilaku dari monyet. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan domain permasalahan yang ada.

2. *Object*

   Object adalah sebuah komponen perangkat lunak yang stukturnya mirip dengan objek pada dunia nyata. Setiap *object* dibangun dari sekumpulan data (atribut) yang disebut variabel untuk menjabarkan karakteristik khusus dari objek, dan juga terdiri dari sekumpulan method yang menjabarkan tingkah laku dari objek. Bisa dikatakan bahwa objek adalah sebuah perangkat lunak yang berisi sekumpulan variabel dan method yg berhubungan.

3. Abstraksi

   Abstraksi merupakan kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

4. Enkapsulasi

   Enkapsulasi 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. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

5. *Polymorphism*

   Polymorphism (polimorfisme) berasal dari bahasa Yunani yang berarti banyak bentuk. konsep ini memungkinkan digunakannya suatu *interface* yang sama untuk memerintah objek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda. Harus diperhatikan disini bahwa *interface* yang sama tidak berarti cara kerjanya juga sama.

6. Inheritas

   Inheritas 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 haru mengimplementasi ulang perilaku tersebut. Namun bahasa berbasis objek tidak selalu memiliki inheritas.

Standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemograman berorientasi objek, yaitu *Unified Modeling Language* (UML).

2.4.1 UML (*Unified Modelling Language*)

Unified Modeling Language (UML) merupakan bahasa pemodelan umum yang digunakan untuk melakukan spesifikasi, visualisasi, konstruksi dan dokumentasi artifak dari *software system. Unified Modeling Language* (UML) adalah keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek (OO) [16].

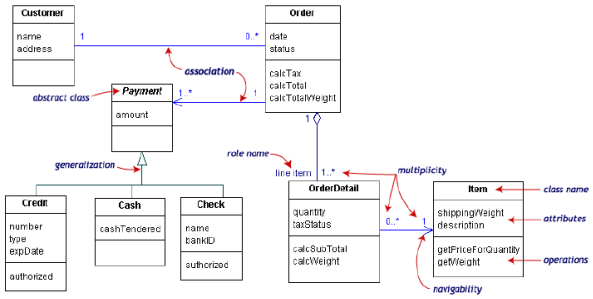
2.4.1.1 Kategori *Diagram*

UML terdiri dari diagram, notasi, konsep dan aturan yang digunakan dalam memodelkan sistem. Diagram UML dikelompokan menjadi 2, yaitu sebagai berikut :

1. Diagram yang menggambarkan struktur yang statis dari sistem, terdiri dari :

   a. *Class Diagram*

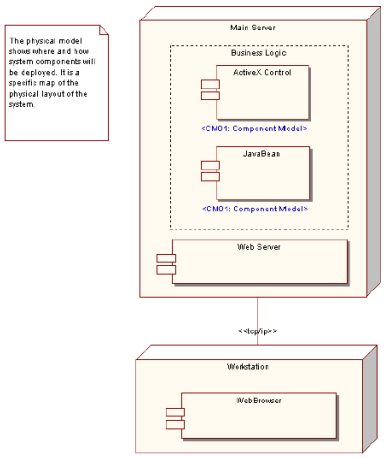
   Class diagram menggambarkan struktur dan deskripsi class, package dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.



Gambar 2.8 Contoh *Class diagram* [16]

   b. *Deployment Diagram*

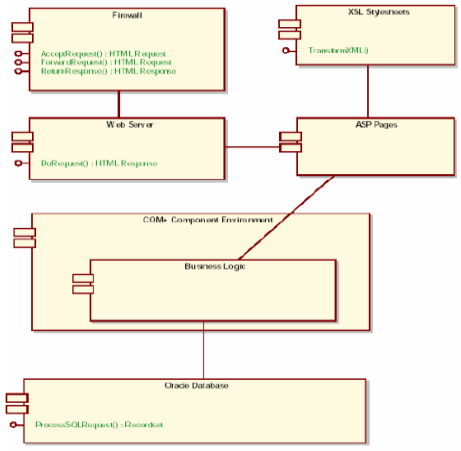
   Deployment diagram menggambarkan sumber fisik dalam sistem, termasuk node, komponen dan koneksi (model implementasi sistem yang statistik).



Gambar 2.9 Contoh *Deployment Diagram* [16]

c. *Component Diagram*

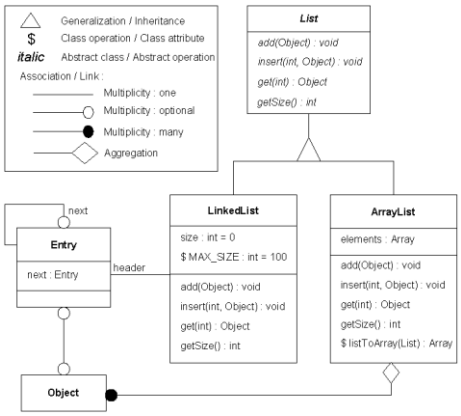
Component diagram merupakan gambaran aspek fisik sistem berbasis objek dengan menunjukkan hubungan dan ketergantungan dalam serangkaian komponen.



Gambar 2.10 Contoh *Component Diagram* [16]

d. *Object Diagram*

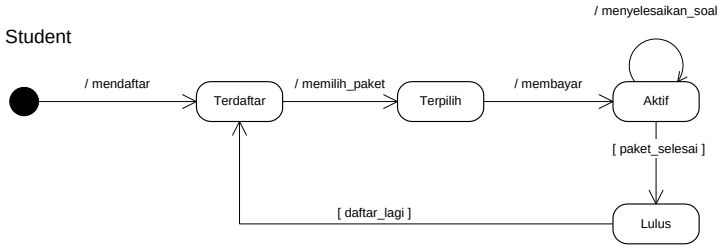
Object diagram menggambarkan hubungan antar elemen dalam model, tapi dengan memakai objeknya, bukan menggunakan class.



Gambar 2.11 Contoh *Object Diagram* [16]

e. *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.

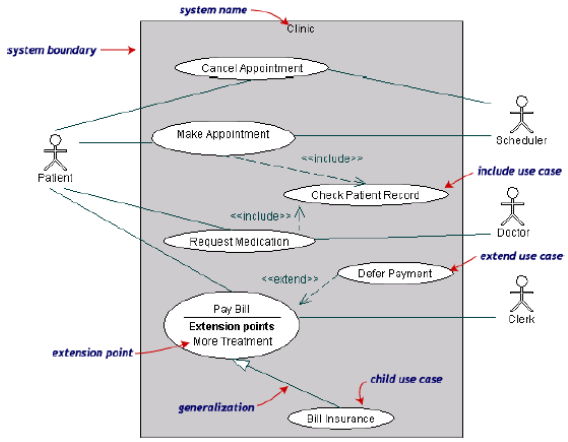


Gambar 2.12 Contoh *Statechart Diagram* [16]

2. Diagram yang menggambarkan struktur yang dinamis dari sistem, terdiri dari :

a. *Use Case Diagram*

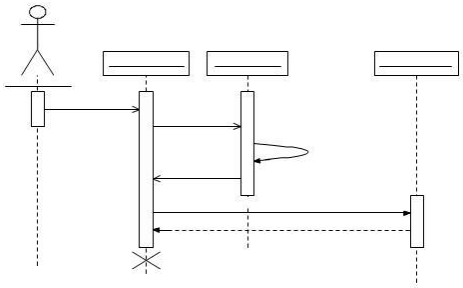
Use case diagram digunakan untuk mendaparkan persyaratan/kebutuhan *system* dan menggambarkan hubungan antara system dengan lingkungan.



Gambar 2.13 Contoh *Use Diagram* [16]

b. *Sequence Diagram*

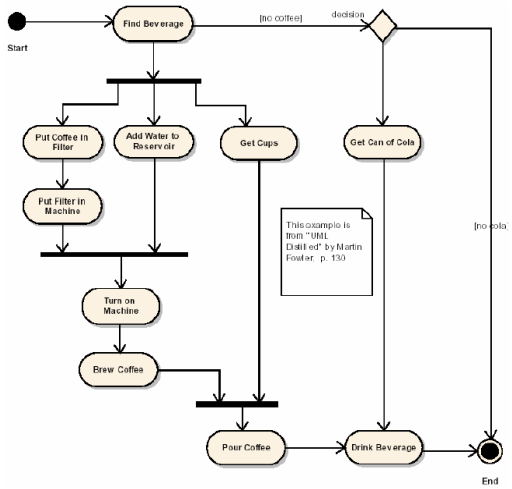
Sequence diagram merupakan diagram yang menggambarkan pola hubungan diantara sekumpulan objek yang saling mempengaruhi menurut urutan waktu.



Gambar 2.14 Contoh *Sequence Diagram* [16]

c. *Activity Diagram*

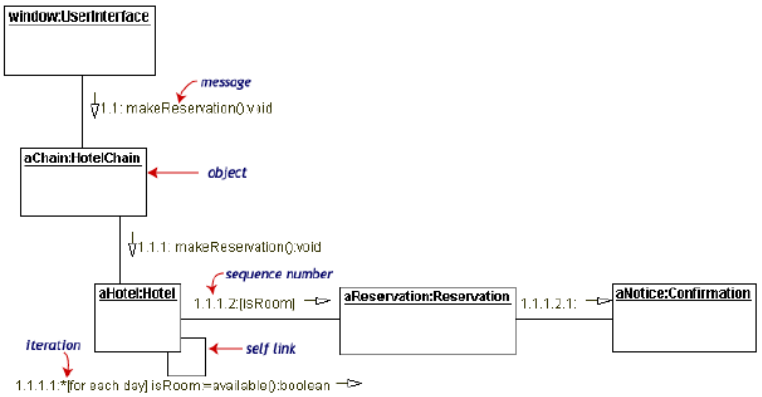
Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir.



Gambar 2.15 Contoh *Activity Diagram* [16]

d. *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*.



Gambar 2.16 Contoh *Collaboration Diagram* [16]

2.5 *Adobe Flash CS3 Professional*

Adobe Flash adalah platform multimedia dan software yang digunakan untuk *authoring* grafis vektor, animasi, *games*, dan *Rich Internet Applications* (RIA) yang dapat dilihat, diputar dan dieksekusi di *flash player* [17]. Aplikasi flash dan animasi dapat diprogram menggunakan bahasa pemograman yang disebut *ActionScript*. Bahasa tersebut merupakan penyempurnaan bahasa pemrograman *ECMAScript*, dengan model klasik *Java-style class*, daripada model prototipe *JavaScript*.

Adobe Flash CS3 Professional merupakan versi ke 9 dari flash yang *release* pada 16 April 2007, dengan kelebihan utamanya adalah *ActionScript* 3.0 yang mendukung pemograman berorientasi objek yang pada versi sebelumnya hanya terdapat *ActionScript* 1.0 dan 2.0.

2.5.1 *ActionScript 3.0*

ActionScript merupakan bahasa *scripting* yang terdapat di gunakan Flash yang bertujuan untuk mempermudah pembangunan suatu aplikasi atau animasi. Biasanya semakin kompleks animasi pada Flash, maka akan semakin banyak memakan frame. Dengan *ActionScript*, penggunaan frame tersebut dapat dikurangi, bahkan dapat membuat animasi yang kompleks hanya dengan satu frame saja.

ActionScript awalnya berasal dari *ActionScript* 1.0 yang dirilis pada tahun 2000 di Macromedia Flash 5. Bahasa scripting ini berisi semua kode dan perintah

lainnya yang berbasis web pengembang bahasa, namun kecepatan dan kekuatannya sangat pendek. Selanjutnya *ActionScript* 2.0, versi ini digunakan pada Macromedia Flash MX 2004 hingga Macromedia Flash 8. Kelebihannya ialah memiliki kemampuan *compile time checking*, *strict-typing*, dan *class-based syntax*. *ActionScript* 3.0 baru mulai digunakan pada Adobe Flash CS3 yang merupakan restrukturisasi fundamental dari model pemrograman sebelumnya. Banyak kelebihan dari AS3 seperti penggunaannya yang luas terutama dalam pengembangan *Rich Internet Application* (RIA) dan mendukung pemograman berorientasi objek.

Berikut merupakan pembaharuan dari *ActionScrpit* 3.0, yaitu :

1. Dapat memeriksa jenis-jenis informasi yang ada di *compile time* dan *runtime* prototipe.
2. Peningkatan kinerja dari sistem pewarisan berbasis kelas yang terpisah dari sistem pewarisan berbasis prototipe.
3. Dukungan untuk paket-paket, *namespaces*, dan ekspresi reguler.
4. Mengkompilasi ke tipe baru sepenuhnya dari *bytecode*, tidak kompatibel dengan *Actionscript* 1.0 dan 2.0 *bytecode*.
5. Revisi *Flash Player* API disusun menjadi paket.
6. Integrasi *ECMAScript* untuk XML (E4X) untuk keperluan pengolahan XML.

Dapat dibilang AS3 merupakan Full OOP sedangkan AS2 masih *procedural* & struktural biasa. AS3 ini dirancang untuk beberapa tujuan, yaitu sebagai berikut:

1. Keamanan, bahasa ini mendukung keamanan pengetikan sehingga para pengembang dapat menulis jelas (tidak ambigu) mudah *maintainable code*.
2. Kesederhanaan, bahasa ini cukup intuitif bagi pengembang untuk bisa membaca dan menulis program tanpa terus-menerus berkonsultasi dengan referensi manual.
3. Kinerja, bahasa memungkinkan pengembang untuk menulis program yang kompleks dengan efisien dan responsif.

4. Kompatibilitas, bahasa yang pendek menyediakan kompatibilitas ke belakang dan ke depan jalan dan tumpang tindih yang signifikan dengan standar industri.

2.6 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah jaminan kualitas dari elemen kritis perangkat lunak dan merepresentasikan review akhir dari spesifikasi, desain, dan pembuatan kode [19]. Pengujian perangkat lunak juga memberikan pandangan mengenai perangkat lunak secara obyektif dan independen, yang bermanfaat pada operasional bisnis untuk memahami tingkat risiko pada implementasinya. Teknik-teknik pengujian mencakup, tetapi tidak terbatas pada proses mengeksekusi suatu bagian program atau keseluruhan aplikasi dengan tujuan untuk menemukan kesalahan perangkat lunak. Pada umumnya terdapat 2 pengujian yaitu :

1. Pengujian Alpha

Pengujian alpha dilakukan pada sisi pengembang yang dianggap pemakai dan merekam semua kesalahan dan masalah pemakaian. Pengujian alpha dilakukan pada sebuah lingkungan yang terkendali. Tahap pertama memulai pengujian produk. Pada fase ini produk masih dalam tahap pengujian dan belum sampai pada publik.

Terdapat 2 macam pendekatan pengujian, yaitu sebagai berikut:

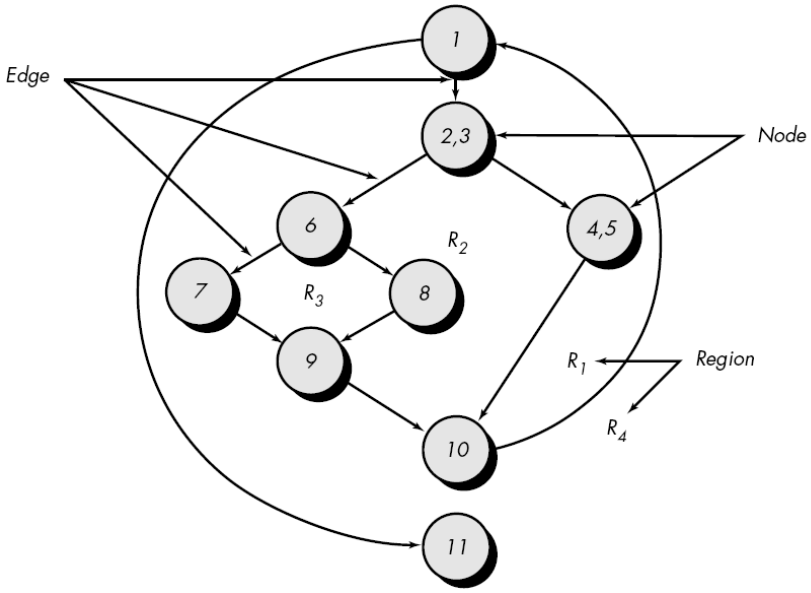
a. *White Box Testing*

Pengujian white box, kadang disebut juga pengujian glass box, merupakan metode desain uji kasus yang menggunakan struktur kontrol dari desain prosedural untuk memperoleh uji kasus. Dengan menggunakan metode pengujian white box, perekayasa sistem dapat melakukan test case untuk memberikan jaminan bahwa :

1. Semua jalur independen pada suatu modul ditelusuri minimal 1 kali
2. Semua jalur keputusan logis True/False dilalui
3. Semua loop dieksekusi pada batas yang tercantum dan batas operasionalnya
4. Struktur data internal digunakan agar validitas terjamin

Pengujian white box dapat dilakukan dengan pengujian *basis path*, metode ini merupakan salah satu teknik pengujian struktur kontrol untuk menjamin semua

statemen dalam setiap jalur independen program dieksekusi minimal 1 kali. Perhitungan jalur independen dapat dilakukan melalui metrik *Cyclomatic Complexity*. Sebelum menghitung nilai *Cyclomatic Complexity*, terlebih dahulu diterjemahkan desain prosuderal ke bagan alir (*flow chart*), kemudian dibuat *flow graph*.



Gambar 2.17 Contoh *Flow Graph* [19]

Cyclomatic complexity digunakan untuk mencari jumlah path dalam satu flow graph. Dapat dipergunakan rumusan sebagai berikut :

- Cyclomatix complexity V(G) untuk grafik alir dihitung dengan rumus:

$$V(G) = E – N + 2$$

Dimana:

E = jumlah edge pada grafik alir

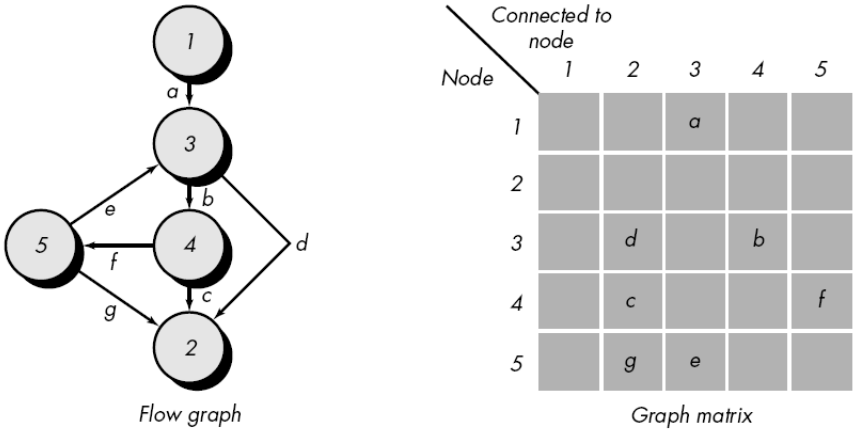
N = jumlah node pada grafik alir

- Cyclomatix complexity V(G) juga dapat dihitung dengan rumus:

$$V(G) = P + 1$$

Dimana P = jumlah predicate node pada grafik alir

Untuk mengembangkan perangkat lunak yang membantu dalam pengujian basis path, *graph matrix* dapat sangat berguna. Graph matrix adalah matriks persegi yang terdiri dari jumlah baris dan kolom yang sama dengan jumlah node pada flow graph. Setiap baris dan kolom sesuai dengan identifikasi simpul, dan entri matriks sesuai dengan koneksi tepi antara node.



Gambar 2.18 Contoh *Graph Matrix* [19]

b. *Black Box Testing*

Pengujian black box, juga disebut pengujian perilaku, berfokus pada persyaratan fungsional perangkat lunak. Pengujian black-box memungkinkan perekayasa perangkat lunak untuk mendapatkan set kondisi input yang akan sepenuhnya melaksanakan semua persyaratan fungsional untuk suatu program. Pengujian black-box bukan merupakan alternatif untuk teknik white-box. Sebaliknya, merupakan pendekatan komplementer yang kemungkinan akan mengungkap kelas yang berbeda dari kesalahan daripada metode white box. Pengujian pada Black Box berusaha menemukan kesalahan seperti:

- Fungsi-fungsi yang tidak benar atau hilang
- Kesalahan interface
- Kesalahan dalam struktur data atau akses database eksternal
- Kesalahan kinerja
- Inisialisasi dan kesalahan terminasi

2. Pengujian Beta

Pengujian beta digunakan untuk preview dalam sebuah produk perangkat lunak. pengujian beta hampir sama dengan pengujian alpha, namun pada tahap ini produk sudah didistribusikan ke publik. Pengujian beta juga bermanfaat Untuk mengetahui tanggapan dan penilaian pengguna atau publik terhadap produk perangkat lunak. Untuk mendapatkan tanggapan dari publik dapat menggunakan kuisoner, angket, poling. Setelah sampel terkumpul maka dilakukan pengukuran menggunakan skala pengukuran.

Skala pengukuran terdiri dari beberapa macam, yaitu sebagai berikut[20] :

a. Skala Nominal

Skala nominal adalah skala pengukuran yang menyatakan kategori atau kelompok dari suatu subyek. Contoh :

Jenis kelamin responden

Laki-laki = 1 Wanita = 2

b. Skala Ordinal

Skala ordinal adalah skala pengukuran yang meyatakan kategori sekaligus melakukan rangking terhadap kategori. Contoh :

Kita ingin mengukur preferensi responden terhadap empat merek produk air mineral.

Merek Air	Mineral Rangking
Aquana	1
Aquaria	2
Aquasan	3

c. Skala Interval

Skala interval merupakan skala pengukuran yang banyak digunakan untuk mengukur fenomena atau gejala sosial, dimana pihak responden diminta melakukan rangking terhadap preferensi tertentu sekaligus memberikan nilai terhadap preferensi tersebut. Jenis skala yang dapat digunakan untuk penelitian sosial, yaitu :

- Skala *Linkert*

Digunakan untuk mengukur sikap, pendapat dan persepsi seseorang atau sekelompok orang tentang fenomena sosial. Contoh :

Preferensi

1. Sangat Setuju
2. Setuju
3. Ragu-ragu
4. Tidak Setuju
5. Sangat Tidak Setuju

- Skala *Gutmann*

Pengukuran untuk memperoleh jawaban responden yang tegas. contoh :

Bagaimana pendapat anda, bila tuan Gunawan menjabat sebagai manager ?

a. Setuju

b. Tidak Setuju

- *Sematic Defferential*

Suatu skala pengukuran yang disusun dalam suatu garis dimana jawaban sangat positif terletak dibagian kanan garis, sedangkan jawaban sangat negatif terletak dibagian kiri garis atau sebaliknya.

- *Rating Scale*

Suatu skala pengukuran dimana responden menjawab salah satu jawaban kuantitatif yang disediakan.

d. Skala Rasio

Skala rasio adalah skala interval yang memiliki nilai dasar (based value) yang tidak dapat diubah. Contoh :

Umur responden memiliki nilai dasar nol.