BAB II

LANDASAN TEORI

Dalam pembuatan Perangkat lunak ini. Beberapa dasar ilmu pengetahuan dibidang Informatika yang digunakan untuk mendukung penelitian ini adalah Rekayasa Perangkat Lunak (RPL), *Unified modeling language* (UML), Pembelajaran Dasar Komputer, *Macromedia Flash 8, Adobe Audition.*

2.1 Konsep Dasar Rekayasa Perangkat Lunak

2. 1. 1 Pengertian Perangkat Lunak

Roger S. Pressman berpendapat bahwa :

"Perangkat lunak adalah (1) perintah (program komputer) yang bila dieksekusi memberikan fungsi dan unjuk kerja seperti yang diinginkan, (2) struktur data yang memungkinkan program memanipulasi informasi secara proporsional dan (3) dokumen yang menggambarkan operasi dan kegunaan program". [3]

2. 1. 2 Pengertian Rekayasa Perangkat Lunak

Beberapa pengertian menurut para ahli di bidang perangkat lunak adalah sebagai berikut [1] :

Menurut Stephen R. Schach :

"Rekayasa perangkat lunak adalah sebuah disiplin dimana dalam menghasilkan perangkat lunak bebas dari kesalahan dan dalam pengiriman anggaran tepat waktu serta memuaskan keinginan pemakai."

Menurut Fritz Bauer :

"Rekayasa perangkat lunak adalah penetapan dan penggunaan prinsip rekayasa dalam rangka memperoleh perangkat lunak yang dapat dipercaya dan dapat bekerja secara efisien pada mesin nyata."

Menurut IEEE 610.12 :

"Rekayasa perangkat lunak adalah sebuah studi pendekatan dan aplikasi secara sistematis, disiplin pengembangan operasi dan pemeliharaan perangkat lunak yang kesemuanya itu merupakan aplikasi rekayasa yang berkaitan dengan perangkat lunak."

Dari ketiga pengertian tersebut, arti yang diberikan *Fritz Bauer* sangat sesuai dengan tujuan dan sasaran dari rekayasa perangkat lunak. Pernyataan *Stephen R. Schach* terbatas pada menanggulangi kekurangan yang terjadi jika tidak menerapkan rekayasa perangkat lunak. IEEE 610.12 memberikan pengertian yang paling baik dalam menyampaikan wujud dari rekayasa perangkat lunak.

Penulis menyimpulkan dari beberapa pengertian diatas bahwa rekayasa perangkat lunak adalah sebuah disiplin ilmu yang menangani perancangan, pembuatan dan pemeliharaan suatu perangkat lunak dengan memakai sistem atau prinsip aturan tertentu yang sistematis untuk menghasilkan sebuah perangkat lunak yang sesuai dengan kebutuhan nyata pemakai dengan tingkat fungsi dan efisiensi yang maksimal.

Perangkat lunak merupakan ilmu yang penting untuk diperdalam karena teknologi ini memberikan stabilitas, kontrol, dan organisasi aktifitas. Beberapa tujuan yang dilakukan rekayasa perangkat lunak, antara lain :

1. Untuk membangun *software* yang benar dan benar sebuah *software* (*Right Software and Software Right*).
2. Untuk membangun *software* yang tepat (*correct*).
3. Dikelola dengan baik untuk pemeliharaan kebenarannya (*correctness*).

2. 1. 3 Metode Perancangan Sistem

"Metode Prototyping dirancang agar dapat menerima perubahan-perubahan dalam rangka menyempurnakan prototipe yang sudah ada sehingga pada akhirnya dapat menghasilkan sistem informasi manajemen yang dapat diterima dan perubahan-perubahan yang terjadi dianggap merupakan bagian dari proses pengembangan itu sendiri." [3]

Karakteristik metode *prototyping* dibagi dalam 4 bagian yaitu pemilihan fungsi (*Function selection*), penyusunan sistem informasi (*Construction*), evaluasi (*Evaluation*) dan penggunaan selanjutnya. (*Further use*). Uraian dari langkah diatas adalah sebagai berikut :

a. Pemilihan fungsi (*Functional selection*)

Mengacu pada pemilihan fungsi yang harus ditampilkan oleh *prototyping*. Pemilihan harus selalu dilakukan berdasarkan pada tugas-tugas yang relevan.yang sesuai dengan contoh kasus yang akan diperagakan. Walaupun luasnya gambaran yang disajikan oleh prototipe tidak selamanya sama dengan hasil akhir dari sistem informasi manajemen yang akan disusun, tetapi

prototype ini harus mewakili bentuk yang akan disusun. Terdapat dua perbedaan antara cakupan fungsi prototipe dengan hasil akhir yaitu :

1) Fungsi sistem informasi yang diterapkan mendekati bentuk akhirnya, tetapi hanya fungsi yang terpilih saja yang digunakan (*Vertical prototipe*).

2) Fungsi-fungsi tidak ditampilkan secara terinci seperti yang akan digunakan pada hasil akhir, tetapi dapat diperagakan (*Horizontal prototipe*).

Pada beberapa bagian dari prototipe, seringkali kedua elemen tersebut disajikan secara bersama-sama.

b. Penyusunan sistem informasi (*Construction*)

Bertujuan untuk memenuhi permintaan akan tersedianya prototipe. Secara umum prototipe ini harus lebih kecil dari yang dibutuhkan dalam mengembangkan hasil akhir pada sebuah sistem informasi yang akan digunakan.

c. Evaluasi (*Evaluation*)

Keharusan untuk menerima *input* perbaikan atau penambahan untuk proses pengembangan selanjutnya. Seluruh unsur dalam pengembangan sistem informasi harus yakin bahwa prototipe ini dapat dievaluasi, termasuk adanya partisipasi dari *entitas-entitas* yang akan menggunakan sistem informasi ini.

d. Ada beberapa kemungkinan dalam penggunaan selanjutnya (*Futher use*) dari sebuah prototipe. Hal ini sangat bergantung pada pengalaman yang

dimiliki, dengan prototipe ini pada kondisi lingkungan tertentu. Seringkali prototipe ini digunakan sebagai alat untuk belajar dan selanjutnya tidak dipergunakan lagi, tetapi mungkin juga prototipe ini merupakan bagian dari sistem yang akan digunakan kelak. Jika prototipe ini digunakan sebagai alat untuk belajar, maka penyusunan prototipe ini harus benar-benar dilakukan dengan hati-hati. Ada beberapa aspek yang harus diperhatikan :

1) Manfaat awal, prototipe ini harus memberikan manfaat pada pengembangan perangkat lunak sehingga dapat ditawarkan pada semua unsur yang berkepentingan seperti pengembang dan pemakai.
2) Peragaan, evaluasi dan modifikasi

Prototipe ini harus dapat diperagakan kepada user. Peragaan ini harus dapat menyajikan bagaimana proses kerja user, seperti harus otentik dan tidak menimbukan masalah sehingga evalusi yang dilakukan pun relevan. Prototipe ini harus mudah dirubah baik direvisi maupun ditambah setelah dilakukan evaluasi sehingga dimungkinkan dilakukannya berbagai modifikasi.

3) Pengajaran dan pelatihan, setelah dilakukan evaluasi dan modifikasi prototipe yang baik dapat dipelajari untuk persiapan user saat bekeja dengan sistem informasi manajemen yang sudah jadi.
4) Kesepakatan, perlu diperhatikan bila selama peragaan prototipe ada masukan-masukan dari user, maka harus diarahkan untuk membuat kesepakatan pada sistem informasi yang akan dihasilkan kelak.

Jenis-jenis prototipe [3] :

1) *Feasibility prototyping* digunakan untuk menguji kelayakan dari teknologi yang akan digunakan untuk sistem informasi yang akan disusun.
2) *Requirement prototyping* juga disebut sebagai *discovery prototyping*, digunakan untuk mengetahui kebutuhan aktivitas bisnis pemakai.
3) *Design prototyping* juga sering disebut *Behaviorial prototyping*, digunakan untuk mendorong perancangan sistem informasi manajemen yang akan digunakan.
4) Implementasi prototyping atau disebut juga production prototyping adalah lanjutan dari rancangan prototipe, prototipe ini langsung disusun sebagai sistem implementasi yang akan digunakan.

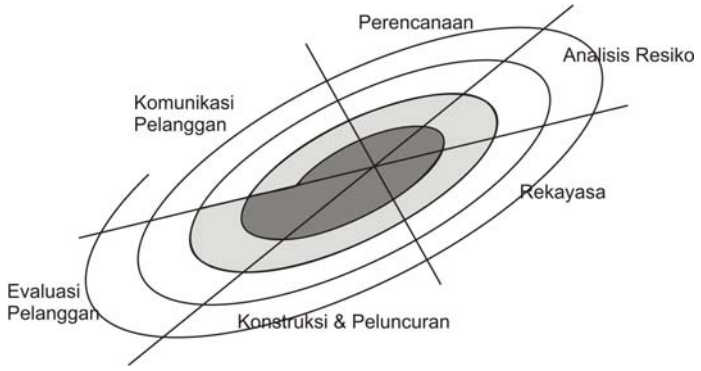
Ada beberapa teknik yang relevan yang dapat dipergunakan dalam penerapan metode *prototyping*, teknik-teknik tersebut adalah perancangan model, perancangan dialog dan simulasi :

a. Perancangan model adalah bagian yang terpenting dalam seluruh strategi *prototyping* yang digunakan sebagai alat untuk menjadikan model menjadi sistem informasi yang sebenarnya.
b. Rancangan dialog disusun agar keterlibatan user menjadi jelas dan fleksibel. Aspek dalam perancangan dialog mencakup keseluruhan unsur yang harus didialogkan.
c. Simulasi dalam *prototyping* dilakukan untuk menunjukkan bagiamana cara kerja sebuah sistem informasi yang akan diterapkan kelak.

Berdasarkan karakteristik, teknik, dan jenis diatas maka dalam membangun Perangkat lunak pembelajaran komputer untuk tunanetra dengan mode *Prototyping*.

2.1.4. Model Spiral

Pada awalnya diusulkan oleh Boehm, adalah model proses perangkat lunak yang evolusioner yang merangkai sifat iteratif dari prototipe dengan cara kontrol dan aspek sistematis dari model sekuensial linier. Model itu berpotensi untuk pengembangan versi pertambahan peranggkat lunak secara cepat. Di dalam model spiral, pernagkat lunak dikembangkan didalam suatu deretan pertambahan. Selama awal iterasi berikutnya, sedikit demi sedikit dihasilkan versi sistem rekayasa yang lebih lengkap.



Gambar .2.1 Model Spiral

1. Komunikasi pengguna → tugas-tugas yang dibutuhkan untuk membangun komunikasi yang efektif diantara pengembang dan pengguna.
2. Perencanaan → tugas-tugas yang dibutuhkan untuk mendefenisikan sumber-sumber daya, ketepatan waktu, dan proyek informasi yang diperlukan.

3. Analisis resiko → tugas-tugas dibutuhkan untuk memprediksi resiko-resiko, baik manajemen maupun teknis.
4. Rekayasa → tugas-tugas yang dibutuhkan untuk membangun satu atau lebih representasi dari aplikasi tersebut.
5. Konstruksi dan peluncuran → tugas-tugas yang dibutuhkan untuk mengkonstruksi, menguji, memasang (install) dan memberikan pelayanan kepada pengguna misalnya pelatihan dan dokumentasi.
6. Evaluasi pelanggan → tugas-tugas yang dibutuhkan untuk memperoleh umpan balik dari pengguna dengan didasarkan pada evaluasi representasi perangkat lunak, yang dibuat selama masa perekayasaan, dan implementasikan selama masa pemasangan.

2.2 Konsep UML (*Unified Modellling Language*)

Saat ini piranti lunak semakin luas dan besar lingkupnya, sehingga tidak bisa lagi dibuat asal-asalan. Piranti lunak saat ini seharusnya dirancang dengan memperhatikan hal-hal seperti *scalability*, *security*, dan eksekusi yang robust walaupun dalam kondisi yang sulit. Selain itu arsitekturnya harus didefinisikan dengan jelas, agar *bug* mudah ditemukan dan diperbaiki, bahkan oleh orang lain selain *programmer* aslinya. Keuntungan lain dari perencanaan arsitektur yang matang adalah dimungkinkannya penggunaan kembali modul atau komponen untuk aplikasi piranti lunak lain yang membutuhkan fungsionalitas yang sama.

Pemodelan (*modeling*) adalah proses merancang piranti lunak sebelum melakukan pengkodean (*coding*). Model piranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem

semacam itu secara menyeluruh. Semakin komplek sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik.

Dengan menggunakan model, diharapkan pengembangan piranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor-faktor seperti *scalability, robustness, security*, dan sebagainya.

Kesuksesan suatu pemodelan piranti lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuan segitiga sukses (*the triangle for success*). Ketiga unsur tersebut adalah metode pemodelan (*notation*), proses (*process*) dan *tool* yang digunakan.

Memahami notasi pemodelan tanpa mengetahui cara pemakaian yang sebenarnya (proses) akan membuat proyek gagal. Dan pemahaman terhadap metode pemodelan dan proses disempurnakan dengan penggunaan tool yang tepat.

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

Dengan menggunakan UML kita dapat membuat 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.

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak.

Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (Object-Oriented Design), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*).

Untuk menguasai UML, sebenarnya cukup dua hal yang harus kita perhatikan:

1. Menguasai pembuatan diagram UML
2. Menguasai langkah-langkah dalam analisa dan pengembangan dengan UML

Seperti juga tercantum pada gambar diatas UML mendefinisikan diagram-diagram sebagai berikut:

a. Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya.

Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

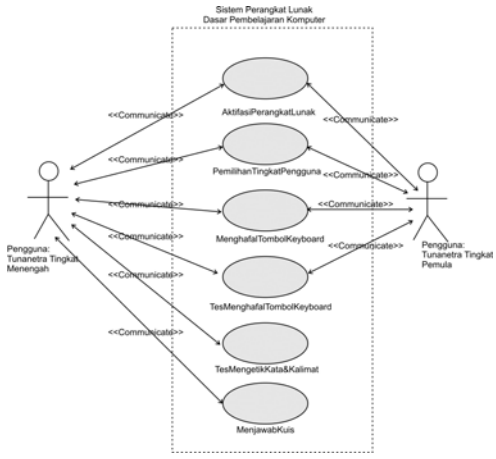
Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri.

Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Contoh *use case diagram* :



Gambar 2.2 Contoh *Use Case*

b. Class Diagram

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

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

Class memiliki tiga area pokok :

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

Atribut dan metoda dapat memiliki salah satu sifat berikut :

• *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan

• *Protected*, dapat dipanggil oleh *class* yang bersangkutan dan yang mewarisinya

• *Public*, dapat dipanggil oleh siapa saja

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.*

Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*.

Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.

2.2.1 Hubungan Antar *Class*

1. *Asosiasi*, yaitu hubungan statis antar class. Panah navigability menunjukkan arah query antar class.
2. *Agregasi*, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar class. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (message) yang di-passing dari satu class kepada class lain.

2.2.2 Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring.

Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

2.2.3 Activity Diagram

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

Contoh *activity diagram* tanpa *swimlane*:

2.2.4 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display,* dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal.

Message digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*.

Activation bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message.

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

2.2.5 Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*.

Setiap *message* memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1.

Messages dari level yang sama memiliki prefiks yang sama.

2.2.6 Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya.

Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*.

Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil.

Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

2.2.7 Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan

jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal.

Sebuah *node* adalah server, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

2.2.8 Langkah-Langkah Penggunaan UML

Berikut ini adalah tips pengembangan piranti lunak dengan menggunakan UML:

1. Buatlah daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan use case untuk tiap business process untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Kemudian perhalus use case diagram dan lengkapi dengan requirement, constraints dan catatan-catatan lain.
3. Buatlah deployment diagram secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan requirement lain (non-fungsional, security dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan use case diagram, mulailah membuat activity diagram.
6. Definisikan objek-objek level atas (package atau domain) dan buatlah sequence dan/atau collaboration diagram untuk tiap alir pekerjaan. Jika

sebuah use case memiliki kemungkinan alir normal dan error, buatlah satu diagram untuk masing-masing alir.

7. Buatlah rancangan user interface model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario use case.
8. Berdasarkan model-model yang sudah ada, buatlah class diagram. Setiap package atau domain dipecah menjadi hirarki class lengkap dengan atribut dan metodanya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.
9. Setelah class diagram dibuat, kita dapat melihat kemungkinan pengelompokan class menjadi komponen-komponen. Karena itu buatlah component diagram pada tahap ini. Juga, definisikan tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.
10. Perhalus deployment diagram yang sudah dibuat. Detilkan kemampuan dan requirement piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam node.
11. Mulailah membangun sistem. Ada dua pendekatan yang dapat digunakan :
    1. Pendekatan use case, dengan meng-assign setiap use case kepada tim pengembang tertentu untuk mengembangkan unit code yang lengkap dengan tes.
    2. Pendekatan komponen, yaitu meng-assign setiap komponen kepada tim pengembang tertentu.

12. Lakukan uji modul dan uji integrasi serta perbaiki model berserta codenya. Model harus selalu sesuai dengan code yang aktual.
13. Piranti lunak siap dirilis.

2.3 Dasar Pembelajaran Komputer untuk Tunanetra.

Pembelajaran adalah proses memahami suatu keilmuan, adapun pembelajaran komputer itu sendiri adalah proses memahami ilmu komputer.Penggunaan teknologi informasi untuk kegiatan pembelajaran sudah dirasa sebagai keharusan. Karena luasnya aspek implementasi teknologi informasi, perlu dibahasa bersama untuk menentukan prioritas urutan aspek-aspek yang perlu didahulukan. Persoalannya adalah bahwa implementasi teknologi informasi tidak semata-mata masalah teknologi tapi dalam prakteknya lebih banyak berurusan dengan pelaku-pelaku pembelajaran itu sendiri.

Pengembangan TI untuk pembelajaran melibatkan 3 unsur:

1. Penyediaan sarana fisik berupa peralatan TI dan jaringannya,
2. Persiapan untuk perubahan proses pembelajaran, dan
3. Pengembangan materi presentasi dalam proses pembelajaran itu sendiri.

Ketiga unsur ini punya hubungan saling ketergantungan yang erat. Materi presentasi digital memerlukan sarana proyektor LCD, sedangkan pengembangan prasarana TImemerlukan perencanaan penggunaan yang matang mengingat harga yang relatif mahal serta umur ekonomi yang relatif pendek.

Komputer selain sebagai alat bantu untuk belajar, juga bisa sebagai objek untuk dipelajari juga. Pengenalan komputer tidak dibatasi penggunaannya, artinya siapapun boleh mempelajarinya termasuk dengan para penyandang tunanetra. Sudah menjadi suatu kewajiban kita semua untuk berperan serta dalam mengenalkan ilmu komputer terhadap mereka, memberikan pembelajaran dasar komputer seperti, mengenal keyboard, mouse, mengoperasikan dan mengetik di komputer.

Perkembangan Dunia Pembelajaran Untuk Tunanetra di Indonesia.

Perkembangan PLB(Pendidikan Luar Biasa) di Indonesia secara berangsur-angsur mendapat perhatian dari pemerintah dan masyarakat, terutama setelah adanya UU Pendidikan Nomor 12 tahun 1954. Undang-Undang tersebut memuat ketentuan tentang pendidikan dan pengajaran luar biasa. Namun demikian, lembaga yang mengelola, mengawasi, dan mensupervisi mengalami perubahan seiring dengan perubahan tata usaha negara. Dari tahun 1954 sampai dengan tahun1963 lembaga PLB, yaitu SLB (Sekolah Luar Biasa) , dan lembaga pendidikan guru PLB, yaitu SGPLB(Sekolah Guru Pendidikan Luar Biasa) , dikelola dan diawasi oleh instansi yang sama, dan organisasi pengawasanya mengalami perkembangan.

Secara singkat dapat dikemukakan bahwa lembaga yang mengawasi PLB adalah sebagai berikut: (a) Tahun 1954 diawasi oleh seksi Pengajaran Luar Biasa (Bagian dari Balai Pendidikan Guru yang berkedudukan di Bandung), (b) Tahun 1955 diawasi oleh Urusan Pendidikan Luar Biasa yang merupakan bagian dari

Jawatan Pengajaran dan berkedudukan di Jakarta, (c) tahun 1957 diawasi oleh Urusan Pendidikan Luar Biasa yang merupakan bagian dari Jawatan Pendidikan Umum dan berkedudukan di Jakarta. Pada tahun 1963 terjadi perubahan struktur organisasi pada Kementerian Pendidikan dan Kebudayaan, yaitu terjadi pemisahan pengawasan antara SLB dengan SGPLB sebagai berikut: (a) PLB diawasi oleh Dinas PLB yang merupakan bagian dari Direktorat Pendidikan Prasekolah (TK), Sekolah Dasar (SD), dan Sekolah Luar Biasa (SLB), (b) Lembaga Pendidikan Guru PLB dikelola oleh Dinas Pendidikan Guru dan Tenaga Teknis yang merupakan bagian dari Direktorat Pendidikan Menengah dan Guru, (c) IKIP Bandung mempelopori program Sarjana Muda PLB dengan lebih menekankan pada upaya menghasilkan tenaga ahli dalam PLB dan dikelola oleh Direktorat Jenderal Pendidikan Tinggi.

Pada tahun 1980 terjadi pemisahan pengelolaan PLB dan LPKPLB (Lembaga Pelatihan Keterampilan Pendidikan Luarbiasa), dimana PLB dikelola oleh Sub Direktorat Pembinaan SLB di bawah Direktorat Pendidikan Dasar pada Ditjen Dikdasmen, sedangkan lembaga pendidikan guru SLB dikelola oleh Direktorat Pendidikan Guru dan Tenaga Teknis, Ditjen Dikdasmen, dan program Sarjana PLB tetap dikelola oleh Direktorat Jenderal Pendidikan Tinggi. Sampai dengan tahun 1984 telah didirikan sekitar 400 SLB, dan sebagian besar dikelola oleh organisasi swasta berbentuk Yayasan. SLB diklasifikasikan menjadi enam kelompok berdasarkan jenis ketunaan, yakni: SLB-A untuk menyandang tunanetra, SLB-B, untuk penyandang tunarungu dan wicara, SLB-C untuk penyandang tunagrahita, SLB-D untuk penyandang tunadaksa, SLB-E untuk

penyandang tunalaras, SLB-G untuk penyandang tunaganda. Semua SLB pembelajarannya dilaksanakan secara terpisah.

Tahun 1984 memiliki arti penting bagi perkembangan PLB di Indonesia. Hal ini disebabkan karena adanya kemauan politik pemerintah (*political will*) untuk menyelenggarakan Program Wajib Belajar 6 Tahun. Ini berarti bahwa semua anak usia sekolah harus menyelesaikan pendidikannya minimal sampai dengan pendidikan sekolah dasar (SD). Program tersebut ditindaklanjuti dengan perintisan Program Wajib Belajar Pendidikan Dasar 9 tahun yang perintisannya dimulai tahun 1989 dan diimplementasikan pada tahun 1994. Dengan demikian, semua anak usia sekolah tanpa kecuali diharapkan memperoleh kesempatan mengikuti pendidikan sampai dengan SLTP. Gerakan wajib belajar tersebut secara langsung mempunyai dampak positif, sebab anak penyandang ketunaan tertentu tidak semuanya dapat tertampung di SLB yang ada, sehingga harus disalurkan/ditampung di sekolah umum atau kelompok belajar. Oleh karena jumlah SLB yang ada sangat terbatas dan letak sebagian besar SLB berada di perkotaan, serta sebagian besar SLB dikelola oleh swasta, maka kondisi tersebut mendorong pemerintah (Depdikbud) untuk mencari upaya pemecahannya dengan beberapa alternatif, yaitu:

Pengenalan bentuk pelayanan PLB yang baru melalui SDLB,. dilakukan melalui dana proyek Inpres tahun 1984 dan telah didirikan 208 buah SLB di 200 kabupaten/ kotamadya yang sama sekali belum memiliki SLB.

Ujicoba di beberapa SD umum/biasa untuk menerima anak yang memiliki ketunaan tertentu (tunanetra) dengan syarat anak yang bersangkutan memiliki kemampuan akademik yang normal. Sekolah yang demikian selanjutnya disebut sekolah dasar (SD) Terpadu.

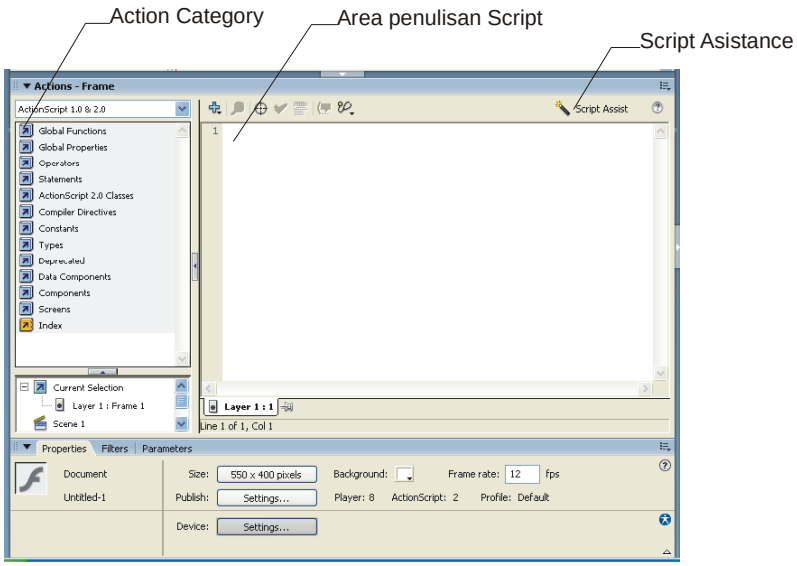
Pendirian SLB Pembina di berbagai daerah di Indonesia sekaligus mempunyai tujuan untuk penelitian, pelatihan, dan pendidikan dalam bidang PLB. Menurut Direktorat Pendidikan Guru dan Tenaga Teknis (Ditgutentis) (1991) yang dikutip Sunardi (1997), sampai dengan tahun 1990, jumlah SLB di Indonesia mencapai 525 dengan rincian 502 SLB dikelola oleh Yayasan Swasta dan 23 SLB negeri dikelola oleh Depdikbud. Jumlah tersebut telah mencakup penyelenggara PLB di tingkat SLTP dan SM.

Lebih lanjut pada tahun 1994 diberlakukan kebijakan Depdikbud tentang penggunaan kurikulum 1994, khusus untuk PLB. Kebijakan tersebut telah memilah-milah jenjang PLB yaitu: SDLB (Sekolah Dasar Luar Biasa), SLTPLB (Sekolah Lanjutan Tingkat Pertama Luar Biasa), dan SMLB (Sekolah Menengah Luar biasa). Dengan berlakunya kebijakan itu, ada kecenderungan anak penyandang ketunaan yang memiliki kemampuan akademik yang normal didorong untuk berintegrasi dengan SD, SLTP, dan SMU. Tingkat SLP dan SMU memberi peluang lebih pada anak yang memiliki ketunaan untuk berkembang lebih baik, mengingat kurikulum jenjang tersebut memberikan banyak program keterampilan. Di samping itu, jenjang pendidikan calon guru PLB yang dianggap layak dari 2 tahun setelah SM diubah menjadi program sarjana di IKIP/FKIP

Universitas. Dampak dari kebijakan tersebut adalah dialihfungsikannya beberapa SGPLB ke SLTP dan SM, sedangkan lainnya dialihkan ke jurusan PLB pada IKIP/FKIP universitas terdekat. Selanjutnya, kurikulum program sarjana PLB disempurnakan dan tingkatkan untuk menghasilkan calon guru PLB yang sesuai dengan tuntutan kemajuan iptek.

Nampaknya perubahan PLB di Indonesia tidak begitu pesat seperti di negara maju lainnya dan bentuk layanannya masih cenderung terpisah. Walaupun telah dikembangkan layanan baru dengan cara mengintegrasikan ke sekolah umum/biasa, tetapi hasilnya masih belum menggembirakan. Untuk mengetahui lebih lanjut tentang berbagai permasalahan dalam layanan PLB di sekolah terpadu dan upaya pemecahannya, seluruh aspek/komponen yang mempengaruhi pendidikan terpadu PLB perlu diteliti/dikaji secara hati-hati dan cermat. Sistem layanan PLB yang terbaik hingga kini masih diperdebatkan. Sebagai contoh, di kalangan pendukung konsep inclusion menghindari pemakaian istilah luar biasa, sementara kelompok yang lain tetap menginginkan pemakaian istilah tersebut. Apa pun yang masih menjadi polemik di kalangan para ahli PLB, nampaknya semuanya mengarah pada perbaikan dan atau pengembangan PLB sesuai dengan situasi dan kondisi yang ditunjang oleh berbagai aturan/kebijakan dan kebutuhan masa kini dan mendatang.

2.4 Konsep Macromedia Flah 8



Gambar 2.3 Panel *Action* di Flash Profesional 8

Macromedia Flash Profesional 8 mengalami peningkatan konsep *Action Script* di bandingkan dengan versi lamanya. *ActionScript 2.0* pada Flash Profesional 8 Sudah mendukung OOP (*Object Oriented Programming*), OOP mempunyai banyak kelebihan dan salah satunya ialah sifat yang *reusable* (dapat digunakan kembali).

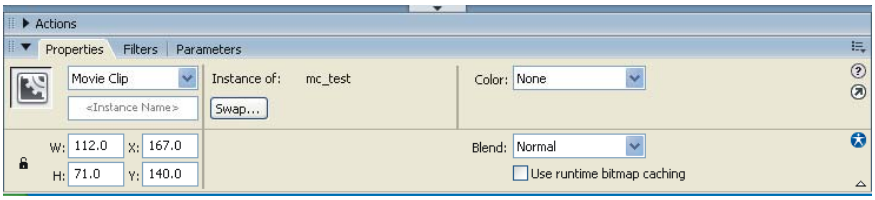
Variabel *ActionScript 2.0* mempunyai tipe data yang lebih spesifik. Hal ini dimaksudkan untuk meminimalisasi kesalahan pengalokasian memori pada data. *ActionScript 2.0* juga mempunyai sifat *case sensitive* sehingga aturan penulisan sintaks menjadi lebih ketat.

2.4.1 *Flash Behaviour*

Dalam Flash Profesional 8 terdapat 3 jenis symbol yang sering digunakan, antara lain;

1. *Movieclip*
2. *Button*
3. *Graphic*.

2.4.2 MovieClip

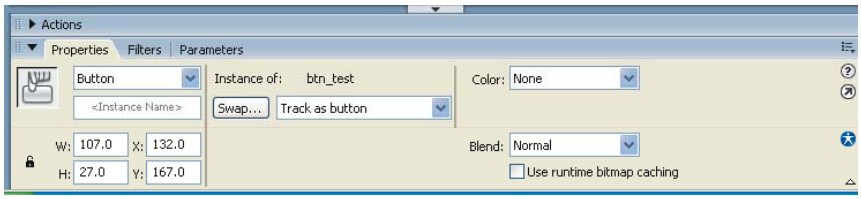


Gambar 2.4 Panel Properti *Movieclip*

Movieclip merupakan *symbol* yang dapat memainkan animasi dalam aplikasi Flash. *Movieclip* merupakan tipe data yang menghubungkan elemen grafik. Tipe data *movieclip* memberi kita fasilitas untuk mengontrol *symbol movieclip* dengan *method* yang terdapat pada *class movieclip*. Kita dapat menggunakan *method* tersebut dengan *syntax* titik (.) seperti pada contoh berikut :

```
My_mx.startDrag(true);
```

2.4.3 Button.



Gambar 2.5 Panel Properti *Button*

Button/tombol sebenarnya merupakan *movieclip* dengan empat frame interaktif. Saat kita membuat *symbol button*, Flash membuat

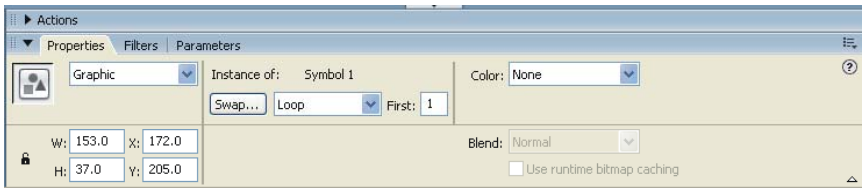
sebuah *Timeline* dengan empat frame. Tiga frame yang pertama menampilkan tiga kemungkinan *state button* tersebut, sedangkan frame keempat mendefenisikan area aktif dari button tersebut. *Timeline* tidak dimainkan antar frame. Setiap frame berreaksi sesuai dengan pointer mouse bereaksi pada *button.*

Untuk membuat *button* yang interaktif, kita dapat membuat sesuatu *instance* dari *symbol button* pada *stage*, kemudian memberikan perintah pada *instance* pada *symbol* tersebut. Kita dapat menuliskan perintah pada *instance* dari *button* tersebut pada *stage*, bukan pada frame dalam *Timeline Button*.

Setiap frame pada *Timeline button* mempunyai fungsi spesifik masing-masing, antara lain ;

a. Frame pertama merupakan frame yang dijalankan saat *pointer/mouse* tidak melewati (*over*) *button.*
b. Frame kedua merupakan frame yang dijalankan saat *pointer/mouse* melewati *button* tersebut.
c. Frame ketiga merupakan frame yang dijalankan saat *button* tersebut diklik.
d. Frame keempat merupakan frame yang mendefinisikan area/ luas *button* tersebut merespon mouse. Area ini tidak terlihat pada file.swf.

2.4.4 Graphic.



Gambar 2.6 Panel Properti Grafik

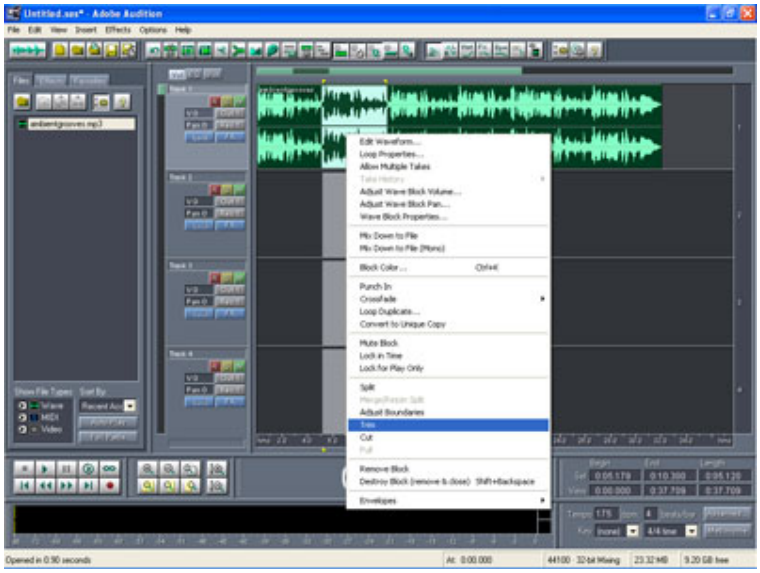
Grafik yang dibentuk oleh flash adalah vektor grafik. Vektor grafik adalah grafik yang membentuk gambar berdasarkan garis serta lekukan yang juga memasukkan warna serta posisi properti grafik tersebut. Sebagai contoh : gambar daun dideskripsikan oleh titik melalui garis yang melewatinya. Warna daun tersebut ditentukan warna outline yang melewatinya dan warna area yang menyertainya.

Saat kita mengedit sebuah grafik vektor, kita memodifikasi properti garis dan lekukan yang membentuknya. Kita dapat menggeser, mengubah ukurannya, mengubah bentuknya, dan juga mengubah warnanya tanpa mengubah kualitas gambar. Grafik vektor tidak bergantung pada resolusi monitor, karena itu grafik vektor dapat dimunculkan dimonitor dengan berbagai resolusi tanpa harus mengubah kualitasnya.

2.5 Adobe Audition .

Adobe Audition adalah perangkat lunak yang dirilis oleh *Adobe Inc*, perangkat lunak ini berfungsi untuk mengedit audio seperti, memotong *audio*, mencampur *audio*, mengkonversi format *audio*, dan memberi spesial

efek pada *audio*. Adapun tampilan salah satu proses pengeditan *audio* dengan menggunakan *Adobe Audition*, seperti yang terlihat pada gambar dibawah ini;



Gambar 2.7 Proses pengeditan *Audio* dengan *Adobe Audition*