BAHAN AJAR

REKAYASA PERANGKAT LUNAK

PENYUSUN

Putri Mentari Endraswari, S.Tr.Kom., M.Kom.

(NIDN. 0005039601, NP. 309621013)

PROGRAM STUDI TEKNIK ELEKTRO

FAKULTAS TEKNIK

UNIVERSITAS BANGKA BELITUNG

2022

Lembar Pengesahan

Judul	: Bahan Ajar Rekayasa Perangkat Lunak
Dosen Pengampu	: 1. Ghiri Basuki Putra, S.T., M.T.
	2. Nurhaeka Tou, S.Kom., M.Kom.
	3. Putri Mentari Endraswari, S.Tr.Kom., M.Kom.
Penyusun	: Putri Mentari Endraswari, S.Tr.Kom., M.Kom.
Program Studi	: Teknik Elektro
Fakultas	: Teknik
Universitas	: Bangka Belitung

Balunijuk, 10 Agustus 2022

Ketua Program Studi

Teknik Elektro

Ghiri Basuki Putra, S.T., M.T.

NIP. 198107202012121003

Kata Pengantar

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, atas segala berkat dan karunia-Nya sehingga buku ajar ini dapat diselesaikan dengan baik dan tepat pada waktunya. Bahan ajar ini ditunjukkan untuk mata kuliah Rekayasa Perangkat Lunak yang mana dapat membantu pengajar serta mahasiswa dalam proses belajar mengajar. Dalam menyelesaikan modul ajar ini, penulis banyak memperoleh bantuan dari berbagai pihak. Oleh karena itu, penulis menyampaikan terima kasih dan penghargaan kepada Ghiri Basuki Putra, S.T., M.T. Selaku dosen payung untuk mata kuliah Rekayasa Perangkat Lunak yang dengan penuh perhatian memberikan dorongan, motivasi, saran, serta masukan yang sangat penting. Penulis juga ingin menyampaikan ucapan terima kasih kepada semua pihak yang terlibat dalam penyelesaian modul ajar ini, kepada:

1. Orang tua serta keluarga tercinta yang telah memberi dukungan dengan penuh rasa kasih sayang, dan senantiasa memberi semangat serta dorongan kepada penulis.
2. Ibu/Bapak dosen-dosen pengampu mata kuliah Rekayasa Perangkat Lunak yang senantiasa selalu bekerja sama dengan baik agar dapat memberikan pelajaran semaksimal mungkin kepada para mahasiswa.
3. Ibu/Bapak dosen-dosen serta staf program studi Teknik Elektro, Fakultas Teknik, Universitas Bangka Belitung.
4. Semua pihak yang tidak bisa penulis sebutkan satu-persatu atas bantuan dan dukungan selama penyusunan bahan ajar ini.

Penulis berharap, bahan ajar ini dapat bermanfaat serta menjadi sumber inspirasi bagi para pembaca. Penulis juga memohon maaf jika dalam penulisan buku ajar ini terdapat kekurangan maupun kekeliruan, penulis menerima kritik serta saran yang membangun demi menyempurnakan buku ajar ini.

Pangkal Pinang, 08 Agustus 2022

Penulis,

Putri Mentari Endraswari, S.Tr.Kom., M.Kom.

Daftar Isi

Daftar Gambar

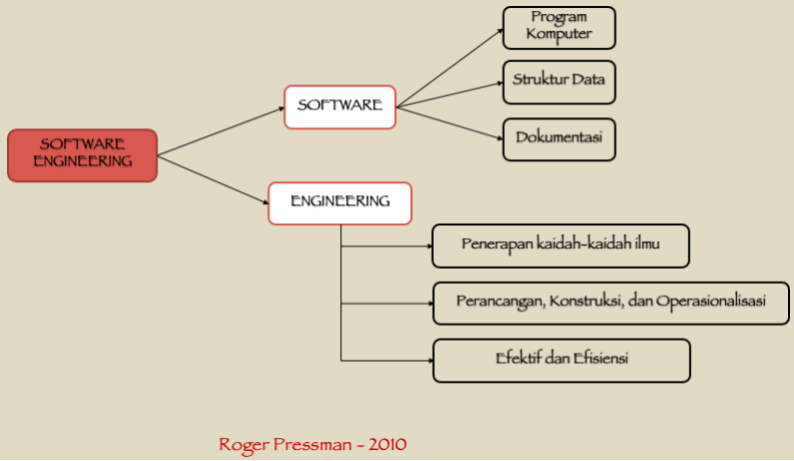
BAB I

Rekayasa Perangkat Lunak

Istilah Rekayasa Perangkat Lunak (RPL) secara umum disepakati sebagai terjemahan dari istilah *Software Engineering*. Istilah *Software Engineering* mulai dipopulerkan pada tahun 1968 pada *software engineering Conference* yang diselenggarakan oleh NATO. Sebagian orang mengartikan RPL hanya sebatas pada bagaimana membuat program komputer. Namun, ternyata terdapat perbedaan yang mendasar antara perangkat lunak (*software*) dan program komputer.

Menurut Rosa A.S, M. Shalahuddin pada bukunya yang berjudul Rekayasa Perangkat Lunak: Terstruktur dan Berorientasi Objek yang diterbitkan pada tahun 2014. Perangkat lunak (*software*) adalah program komputer yang terasosiasi dengan dokumentasi perangkat lunak seperti dokumentasi kebutuhan, model desain, dan cara penggunaan (*user manual*). Perangkat lunak adalah seluruh perintah yang digunakan untuk memproses informasi. Perangkat lunak dapat berupa program atau prosedur. Namun, Perangkat Lunak (*Software*) tidak sama dengan program komputer. Jadi, Perangkat lunak tidak hanya mencakup program, tetapi juga semua dokumentasi dan konfigurasi data yang berhubungan, yang diperlukan untuk membuat program beroperasi dengan benar. Jadi dapat dikatakan juga jika sebuah program komputer tanpa terasosiasi dengan dokumentasinya maka belum dapat disebut sebagai perangkat lunak (*software*). Program adalah kumpulan perintah yang dimengerti oleh komputer sedangkan prosedur adalah perintah yang dibutuhkan oleh pengguna dalam memproses informasi.

Menurut Roger Pressman (2010), *Softaware Engineering* terdiri dari 2 kata, yaitu *software* dan *engineering*. *Software* merupakan satu kesatuan dari tiga unsur yaitu program komputer, stuktur data, dan dokumentasi. *Engineering* adalah penerapan kaidah-kaidah ilmu untuk melakukan perancangan konstruksi, operasionalisasi, tujuannya untuk mendapatkan output secara efektif dan efisien.



Gambar 1. *Softaware Engineering*

A. Karakteristik Perangkat Lunak

Karakter Perangkat lunak adalah sebagai berikut:

- Perangkat lunak dibangun dengan rekayasa (*software engineering*) bukan diproduksi secara manufaktur atau pabrikan.
- Pembuatan perangkat lunak berorientasi kepada customer dan tidak dapat dirakit dari komponen yang sudah ada.
- Perangkat lunak tidak pernah usang (*wear out*).

Oleh karena itu, *software engineering* merupakan Ilmu yang mendalami cara-cara mengembangan perangkat lunak termasuk perancangan, pembuatan, pengujian, pemeliharaan, sampai pada manajemen proyek dan manajemen kualitas.

Lalu seperti apakah *software* yang baik itu? Berikut ini adalah karakteristik atau sifat-sifat dari *software* yang baik.

a. *Maintainability*

Software yang baik haruslah bersifat *Maintainability* atau mudah dipelihara dan dikembangkan. Karena *software* itu akan dioperasikan secara terus menerus untuk melakukan suatu pekerjaan sehingga dia harus mudah dimaintain, harus mudah di pelihara, suatu saat *software* tersebut juga harus diubah untuk memenuhi kebutuhan yang mungkin sudah berubah, sehingga dia juga harus bersiat mudah dikembangkan. Maka karakteristik yang pertama adalah *maintainability*.

b. *Dependability*

Dependability yang artinya dapat dipercaya. *Output* dari sebuah *software* haruslah dapat dipercaya. Makanya harus melalui uji, harus melalui *testing*, sehingga kita yakin, *output* dari *software* itu adalah hasilnya benar dan valid sesuai dengan keinginan.

c. *Efficiency*

Sebuah *softaware* juga harus *efficiency*. Efisien dalam penggunaan sumber daya. Kalau kita mengembangkan *software* tetapi ternyata malah membuat tidak efisien dalam sumber daya, misalkan membuat biaya terlalu membengkak, kemudian sumber daya manuasia yang mengoperasikan menjadi bertambah, yang seperti itulah yang tidak efisien.

d. *Usability*

Karakteristik yang keempat adalah *usability*. *Software* yang baik haruslah mudah digunakan, seusai dengan keinginan *user*. Karena *software* itu nanti yang mengoperasikan adalah *user* atau pengguna. Kalau *software* tidak mudah digunakan sesuai keinginan *user*, maka biasanya *software* itu juga tidak akan bisa digunakan dengan baik. Karena *user* menjadi malas menggunakannya karena *Software* tersebut susah untuk digunakan.

Jika *software* sudah memenuhi 4 kriteria tersebut, berarti *software* sudah bisa dikatakan sebagai *software* yang baik.

B. Perbedaan RPL dengan Rekayasa Sistem

Rekayasa sistem berkaitan dengan semua aspek dalam pembangunan sistem berbasis komputer termasuk hardware, rekayasa PL dan proses. Sedangkan rekayasa perangkat lunak merupakan bagian dari rekayasa sistem yang meliputi pembangunan PL, infrasktruktur, kontrol, aplikasi dan database pada sistem.

Gambar 2. Perbedaan RPL dengan Rekayasa Sistem

C. Proses Perangkat Lunak

Bicara tentang rekayasa perangkat lunak pasti terkait dengan *software methodology*. *Software* proses ini merupakan serngkaian aktivitas secara terstruktur yang bertujuan untuk melakukan pengembangan dan evolusi perangkat lunak. Artinya ketika kita mengembangkan perangkat lunak itu tidak bisa sembarangan, harus terstruktur, harus secara disiplin menerapkan langkah-langkah tertentu sehingga *software* yang dihasilkan dapat sesuai dengan keinginan. Secara umum, aktivitas dalam proses perangkat lunak terdapat 4 hal, yaitu *spesification*, *development*, *validation*, dan *evolution*.

1) *Spesification*

   Mendefinisikan apa yang *software* bisa lakukan dan batasan-batasan pengembangannya.

2) *Development*

   Tahapan produksi *software.*

3) *Validation*

   Pengujian apakah *software* telah dikembangkan sesuai keinginan *user*.

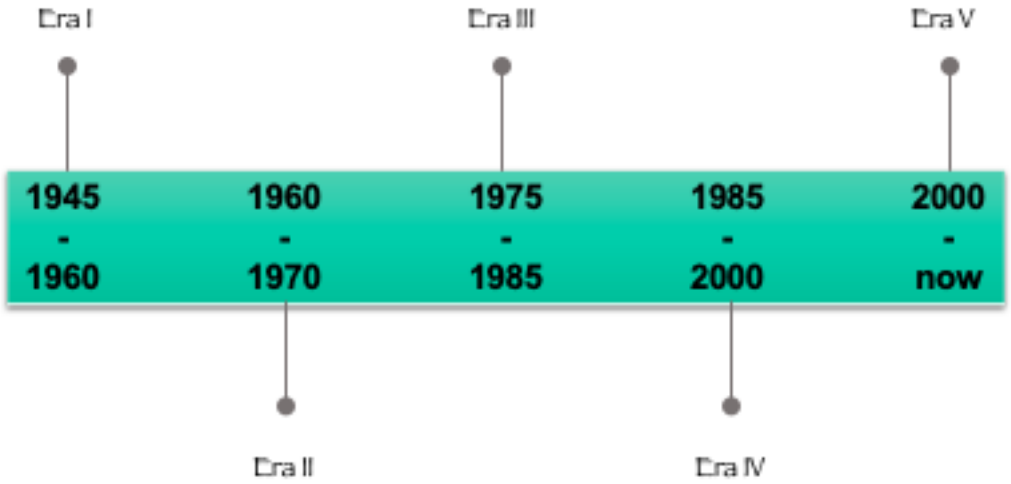
4) *Evolution*

   Perubahan *software* untuk merespon perubahan kebutuhan, tentu perubahan itu adalah sebuah harapan untuk merespon perubahan kebutuhan yang dimilki oleh *user*.

D. Evolusi Perangkat Lunak

Perangkat lunak telah semakin berkembang sejak pertama kali diciptakan tahun 1945. Fokus utama pembuatannya untuk mengembangkan praktik dan teknologi dalam

meningkatkan produktivitas para praktisi pengembang PL dan kualitas aplikasi yang dapat digunakan oleh pemakai. Evolusi dipicu adanya tuntutan bisnis dan lingkungan kerja yang berkembang sangat dinamis.



Gambar 3. Evolusi Perangkat Lunak

1) Era I (1945-1960)
   - Munculnya teknologi perangkat keras di tahap awal.
   - Penggunaan perangkat lunak yg berorientasi *batch*
   - Distribusi perangkat lunak masih terbatas
   - Didominasi perangkat lunak model *customer*
   - Munculnya istilah *software engineering* (akhir 1950an/awal 1960-an)
   - Belum didefinisikan secara jelas tentang aspek *software engineering*
2) Era II (1960-1970)
   - Disebut era krisis perangkat lunak *(software crisis*).
   - Penggunaan perangkat lunak sudah meluas
   - Telah hadir perusahaan yang membangun *software (software house*)
   - Perangkat lunak sudah mengenal multiprogram, *multiuser, real-time*, dan penggunaan *database*
   - Banyak *project* PL yang gagal
     - *Over budget*/anggaran
     - Berakibat rusak fisik dan kematian
     - Meledaknya Roket Ariane , kesalahan perintah dalam PL

- Dua konferensi tentang *software engineering*:
  - Disponsori Komite Sains NATO
  - Tahun 1968 dan 1969
  - Profesi resmi bidang *software engineering*

3) Era III (1975-1985)
   - Pengembangan sistem mengarah ke konsep sistem terdistribusi.
   - Penerapan sistem *embeded intelligence*
   - Harga perangkat keras sudah jauh lebih murah sehingga pemakaian meluas
   - Pemanfaatan jaringan global dan lokal serta sudah diperkenalkan komunikasi digital
4) Era IV (1985-2000)
   - Kemampuan PC sudah setara dengan komputer *mainframe*
   - Penerapan teknologi yang berorientasi pada objek
   - Implementasi sistem pakar
   - Jaringan saraf tiruan
   - Komputasi paralel
   - Jaringan komputer sudah semakin canggih
5) Era V (2000-sekarang)
   - Penggunaan media digital
   - Media web berkembang pesat
   - *Wireless* sudah meluas
   - Teknologi meluas hingga di *mobile computing, mobile programming*
   - Perangkat keras sudah semakin kecil namun *powerfull*
   - Dilakukan berbagai penelitian yang menghasilkan model proses/paradigma pengembangan PL untuk mengatasi krisis PL

E. Tantangan Dalam RPL

- **Tantangan Warisan**

  Dikembangkan bertahun-tahun dengan orang-orang yang berbeda.
- **Tantangan Heterogenesis**

  Dalam hal distribusi dan teknologi
- **Tantangan Pengiriman**

Bagaimana mengirim sistem yang besar dan kompleks dengan cepat namun kualitas tetap terjaga.

F. Pelaku dalam RPL

- **Manajer**

  Manajer proyek, konfigurasi, penjamin kualitas PL

- **Software Development**

  Analisis Sistem, Desainer, Programmer, Inspektor PL, Pengontrol Perubahan

- **Pendukung**

  Staf Administrasi, Humas, Pencatat Teknis, Administrator Database, Administrator Jaringan.

BAB II

Siklus Hidup Perangkat Lunak / *Software Development Life Cycle* (SDLC)

SDLC sesungguhnya merupakan bagian dari proses perangkat lunak. Karena sebuah proses pengembangan perangkat lunak sesungguhnya akan langsung bersentuhan dengan SDLC itu sendiri sebagai sebuah rangkaian proses hidup dari sebuah perangkat lunak, mulai dari analisa hingga sebuah perangkat lunak dikatakan “mati” atau tidak terpakai lagi. Atau bahkan juga saat perangkat lunak tersebut dinyatakan “hidup” kembali dalam bentuk sebuah revisi atau pengembangan baru.

Sebelum memulai tentang pembahasan dan kaitan SDLC dengan RPL, biasamnya ada satu pertanyaan yang umum dipertanyakan oleh sebagian orang. Pertanyaan itu adalah **“Mengapa harus mempelajari proses dan siklus hidup perangkat lunak ?”**. Pertanyaan tersebut terkesan sangat dilematis bagi sebagian orang yang memang penasaran dengan pembelajaran ini. Dan pertanyaan ini juga sebenarnya merupakan kunci jawaban dari alur disiplin RPL sendiri.

Seperti halnya sebuah mahluk hidup, perangkat lunak juga memiliki siklus hidup didalamnya. Perbedaan utama didalamnya adalah jika mahluk hidup hanya mengalami satu kali kehidupan di dunia, maka sebuah perangkat lunak mampu memiliki variasi kelangsungan hidup di dunia.

Gambar 4. Siklus Mahluk Hidup

Gambar 5. Siklus Hidup Perangkat Lunak

Sehingga dengan mempelajari siklus hidup dari perangkat lunak, maka secara otomatis juga akan mempelajari proses hidup dari perangkat lunak itu sendiri, dan jika dirasa perlu maka dapat diputuskan apakah perangkat lunak itu sudah dianggap usang serta mati atau harus direvisi lebih lanjut menjadi sebuah perangkat lunak yang baru.

Dengan mempelajari proses hidup dari sebuah perangkat lunak, maka secara terintegrasi pula akan mampu mempelajari hal apa yang seharusnya dilakukan oleh pengembang perangkat lunak (*software engineer*) dalam proses pengembangan perangkat lunak itu sendiri. Sehingga dengan mempelajari siklus hidup berarti juga mempelajari langkah-langkah untuk menjadi seorang *software engineer* yang baik, di dalam lingkup teori dan juga implementasi.

Seringkali siklus hidup atau *life cycle* sebuah perangkat lunak disamakan dengan proses atau *software process*. Meski sekilas terlihat sama, tetapi sangatlah berbeda antara siklus hidup dan proses dalam konteks RPL. Jika ditinjau dari sisi definisi, siklus hidup memiliki beberapa definisi sebagai berikut:

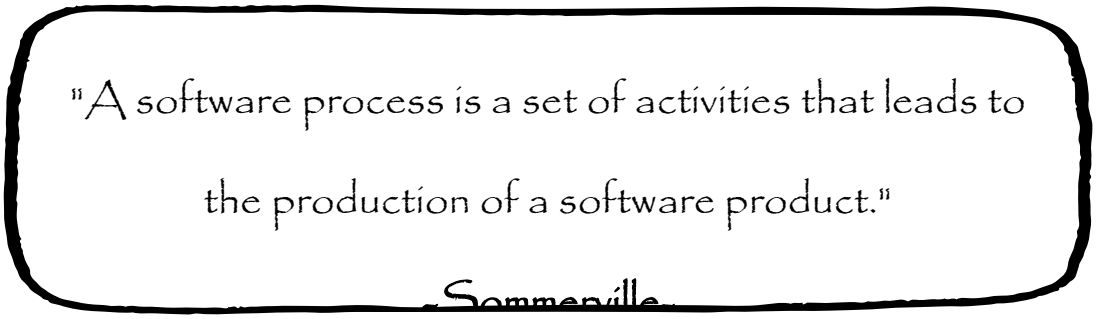
- Menurut Gustafson dalam penelitiannya yang berjudul ***"Theory and problems of software engineering"*** pada tahun 2002, menyatakan bahwa "*The software life cycle is the sequence of different activities that take place during software development*"

  Maksudnya apa? Jadi siklus hidup adalah urutan dari kegiatan yang ada di dalam sebuah pengembangan perangkat lunak. Dalam penjelasannya bahwa urutan tersebut tidaklah harus benar-benar urut, tetapi dapat mengikuti dengan jenis siklus hidup yang dianut oleh pengembang perangkat lunak itu sendiri.
- Menurut Jessica Keyes pada tahun 2002 dalam bukunya yang berjudul **"Software engineering handbook"**, menyatakan bahwa "*A system has a life of its own. It starts out as an idea and progresses until this idea germinates and then is born*".

Jessica Keyes menekankan bahwa sebuah perangkat lunak bisa saja mengalami sebuah siklus hidup bergantung dari proses pengembangannya mulai dari ide dasar hingga saat lahirnya perangkat lunak itu sendiri.

Dari kedua definisi tersebut, Secara umum dapat disimpulkan bahwa siklus hidup perangkat lunak adalah urutan hidup sebuah perangkat lunak berdasarkan perkembangan perangkat lunak yang ditentukan oleh pengembang perangkat lunak itu sendiri. Sehingga dapat ditentukan usia fungsional dari sebuah perangkat lunak, apakah akan menjadi usang dan mati, ataukah lahir kembali dalam bentuk berbeda menggunakan model proses tertentu.

Jika dilihat dari definisi siklus hidup perangkat lunak, mungkin sekilas terlihat bahwa yang dimaksud dengan siklus hidup dalam konteks RPL berarti tidak sama dengan definisi dari proses perangkat lunak. Dari Sommervill e dalam penelitiannya tentang *Software Engineering* pada tahun 2001, menyatakan bahwa proses PL ini merupakan kumpulan aktifitas yang menuju ke sebuah produksi perangkat lunak. Di dalam penjelasannya, dalam sebuah proses perangkat lunak dapat melibatkan kegiatan pemrograman dengan bahasa pemrograman tertentu, tetapi tidak cukup dengan kegiatan tersebut saja. Melainkan juga harus didukung dengan kegiatan lain seperti desain, hingga ke proses evolusi.

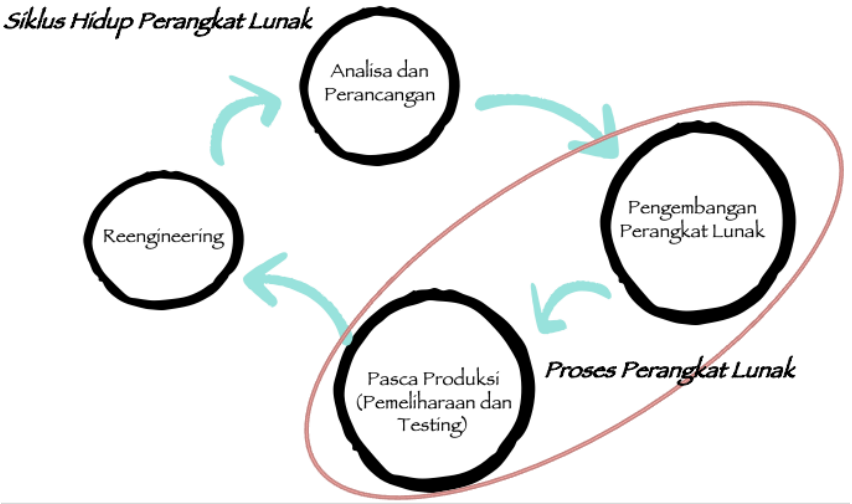


Secara terpisah, Gustafson telah membedakan antara proses dengan siklus hidup dari sebuah perangkat lunak. Siklus hidup lebih menekankan pada milestone atau urutan waktu sedangkan proses lebih menekankan pada kegiatan atau aktifitas yang dijalankan dalam menyusun atau membangun sebuah perangkat lunak. Sehingga jenis proses perangkat lunak lebih banyak mengarah ke penggambaran diagram untuk aktifitas yang terlibat dalam pengembangan perangkat lunak seperti DFD, use case dan sejenisnya.

"A software process model (SPM) describes the processes that are done to achieve software development."

Dari definis-definisi yang telah disebutkan tadi, jelas sudah jika memang proses perangkat lunak adalah sekumpulan aktifitas maupun metode yang digunakan pengembang perangkat lunak dalam melakukan penyelesaian perangkat lunak. Walaupun dari definisi-definisi yang telah disebutkan, menganggap bahwa antara siklus dan proses PL berbeda, namun semuanya memiliki satu muara yang tak tergantikan. Satu titik temu tersebut adalah bahwa baik siklus hidup maupun proses keduanya sangat penting untuk dipahami bagi para pengembang perangkat lunak dan juga pengguna dari perangkat lunak itu sendiri. Sebab jika antara kedua pelaku utama dari RPL tersebut sama-sama tidak memahami siklus hidup maupun proses dari sebuah pengembangan perangkat lunak, maka dapat dipastikan bahwa perangkat lunak yang telah dibangun (terlebih jika memiliki nilai komersil yang tinggi) akan punah dalam sesaat.

Siklus Hidup Perangkat Lunak



Gambar 6. Siklus Hidup Perangkat Lunak

Pemahaman mengenai siklus hidup dan proses sangatlah penting dalam implementasi, karena tanpa keduanya kedua belah pihak (pengembang dan pengguna) tak akan pernah memiliki sudut pandang yang sama dalam satu proses pengembangan perangkat lunak. Sebagai contoh, jika seorang pengembang perangkat lunak yang tidak memahami siklus hidup dan proses perangkat lunak umumnya akan langsung melakukan proses pemrograman sesaat setelah melakukan survey sejenak mengenai kebutuhan pelanggan, tanpa melakukan perancangan ataupun tahapan analisa. Akibatnya, bukan mempercepat proses tapi malah memperlambat waktu pengembangan PL yang diakibatkan banyaknya revisi yang harus dilakukan saat proses *testing*.

Bagi para pengembang PL yang melakukan hal tersebut dijuluki sebagai aliran "**HAJAR BLEH".** Julukan ini mengandaikan seseorang yang tanpa pikir panjang langsung menangani masalah, tetapi akibat yang diderita bukannya penyelesaian dari masalah tersebut malah menambah masalah di kemudian hari. Lalu apa akibatnya bagi para pelanggan?

Bagi para pelanggan yang tidak mengenal siklus hidup dan proses PL, mereka akan mengganggap bahwa PL yang mereka miliki telah usang dan ada kemungkinan mereka akan memanggil pengembang PL lain untuk mengerjakan proyek PL baru. Bagi pihak pengembang, tentu saja hal tersebut akan menguntungkan secara finansial, tetapi secara etika seharusnya hal tersebut tidak terjadi. Jika pemahaman mengenai konsep dasar RPL dapat dijelaskan ke pelanggan atau pengguna, maka semua unsur pelaku RPL akan mampu mengatasi masalah dengan solusi yang lebih efisien. Kesimpulannya adalah Siklus hidup lebih berorientasi kepada urutan waktu sedangkan proses lebih fokus kepada metode yang digunakan dalam aktifitas pengembangan perangkat lunak.

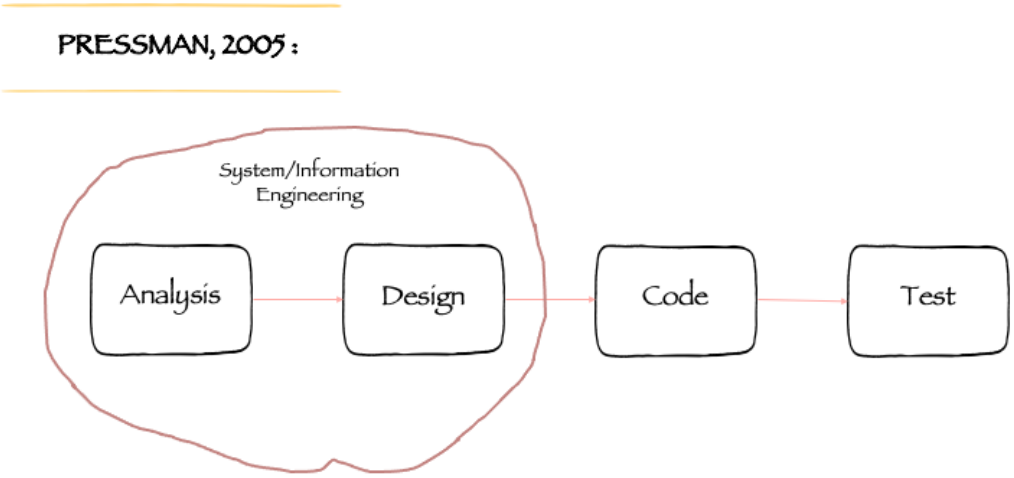
Contoh Model Siklus Hidup Perangkat Lunak

1. *Waterfall model*

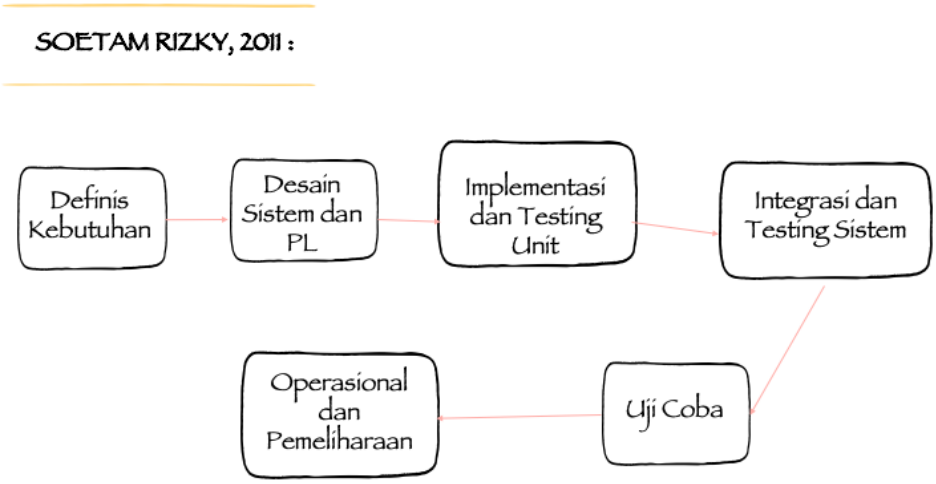
   Siklus hidup yang paling terkenal dalam dunia RPL adalah *waterfall model*. *Waterfall model* merupakan sebuah siklus hidup yang terdiri dari mulai fase hidup perangkat lunak sebelum terjadi hingga pasca produksi. Sebuah proses hidup perangkat lunak memiliki sebuah proses yang linear dan sekuensial. *Waterfall model* diciptakan pertama kali oleh William Royce pada tahun 1970 dan mulai terkenal karena logika fase yang ditampilkan benar adanya.

   Dalam perkembangan jaman, banyak kalangan menyatakan bahwa model siklus hidup *Waterfall model* sudah kuno atau usang. Tetapi dalam kenyataan model ini masih relevan

untuk diterapkan dalam sebuah proyek pengembangan perangkat lunak dengan melakukan adaptasi pada kebutuhan sistem dan skala proyek tersebut.



Gambar 7. Tahapan *Waterfall Model* menurut Pressman-2005



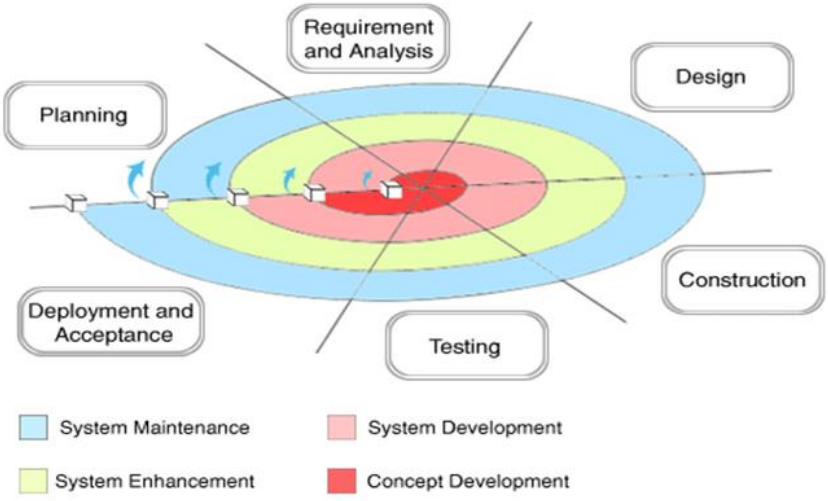
Gambar 8. Tahapan *Waterfall Model* menurut Soetam Rizky-2011

Secara umum, prinsip dari waterfall model adalah bahwa tiap tahapan tidak akan dapat dilaksanakan jika tahapan sebelumnya belum dilakukan. Karenanya, pengembang perangkat lunak alias *software engineer* memiliki kewajiban untuk menjaga siklus hidup tersebut tetap dapat terlaksana tanpa terjadinya sebuah *overlap* antara satu tahap dengan tahap yang lain sehingga proses pengembangan perangkat lunak dapat berjalan efisien. Model ini hanya cocok jika diterapkan pada sebuah sistem yang secara ekstrim telah mengetahui kebutuhannya dengan sangat jelas.

Penyebab kegagalan dari *waterfall model* adalah karena pengguna sendiri yang seringkali kesulitan untuk mendefinisikan kebutuhannya sendiri dalam satu waktu (terutama di saat tahapan definisi kebutuhan). Sehingga kebutuhan pengguna sering tercetus pada saat perangkat lunak telah melalui tahap implementasi.

2. Model Spiral

Teori lain yang ada dalam siklus hidup sebuah perangkat lunak adalah model spiral. Model ini dibuat oleh Boehm dan merupakan evolusi dari *waterfall model*. Disebut spiral karena memang memiliki sebuah model spiral.



Gambar 9. Model Spiral

Dalam model ini secara umum dijelaskan bahwa siklus hidup perangkat lunak diawali dari *planning* atau perencanaan dari perangkat lunak itu sendiri yang didalamnya termasuk waktu pengerjaan, sumber daya yang dibutuhkan serta informasi menyangkut pengerjaan proyek. Kemudian tahap berikutnya adalah analisa resiko atau *risk analysis* yang menyatakan resiko yang mungkin terjadi baik secara teknis maupun secara manajerial. Setelah itu, baru dilakukan proses engineering atau pembuatan dari perangkat lunak itu sendiri.

Saat perangkat lunak telah dianggap selesai, maka akan masuk ke tahap *construction and release* yang berarti bahwa perangkat lunak telah siap diinstalasikan ke pengguna serta proses yang berkaitan seperti training dan pembuatan dokumentasi. Kehidupan perangkat lunak dianggap berakhir pada tahapan *customer evaluation* yang berarti telah ada umpan balik dari pengguna sekaligus sebagai dasar untuk pengembangan perangkat lunak berikutnya.

Menurut IEEE, Siklus Hidup Perangkat Lunak haruslah :

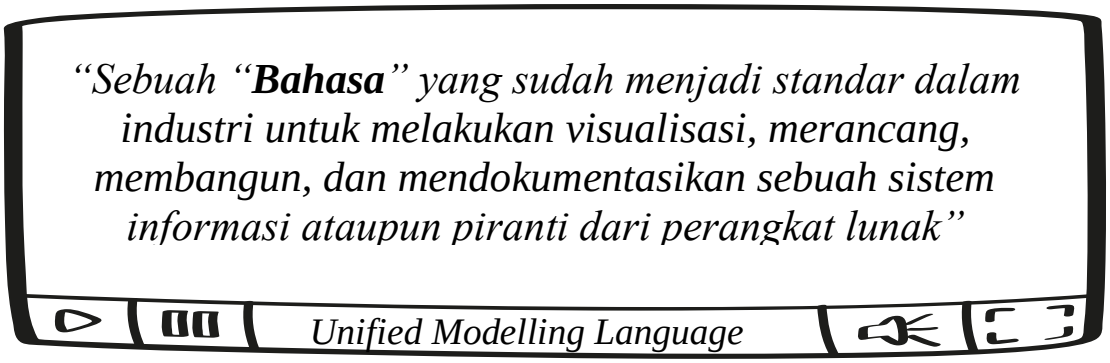
1. Mendeskripsikan dan menyimpan catatan sebuah perangkat lunak dalam siklus hidup perangkat lunak tersebut.
2. Mendefinisikan pengaturan proses perangkat lunak
3. Menyediakan dokumentasi mengenai apa yang terjadi dalam perangkat lunak mulai dari proses pengembangan hingga pasca produksi (pemeliharaan dan pengembangan lebih lanjut)
4. Menyediakan bukti bahwa proses telah dilaksanakan dengan baik.

Menurut IEEE, siklus hidup dalam sebuah perangkat lunak sesungguhnya lebih terfokus kepada siklus hidup sebuah data dalam perangkat lunak. Meski secara umum hal tersebut dapat dibenarkan,tetapi beberapa perangkat lunak bisa jadi tidak memiliki data yang cukup signifikan untuk disimpan. Siklus hidup (dalam konteks data) lebih banyak fokus kepada dokumentasi dari perangkat lunak. Dokumentasi dalam lingkup ini bukan hanya sekedar sebuah dokumen *help* atau *user guide*, tetapi lebih mengarah semacam catatan tentang apa yang terjadi dalam sebuah proses pengembangan perangkat lunak.

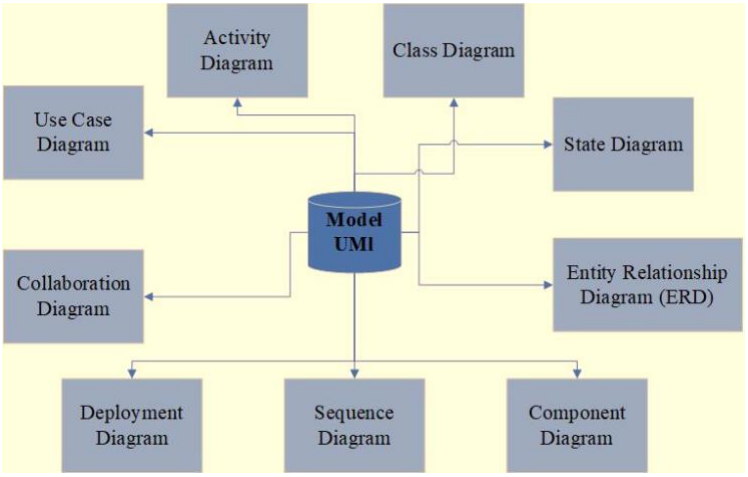
BAB III

Unified Modelling Language (UML)

Secara definisi UML ini merupaka sebuah Bahasa yang menyediakan berbagai model diagram untuk keperluan perancangan system ataupun piranti rekayasa perangkat lunak. Jadi dengan adanya UML ini dapat membantu kita untuk melakukan visualisasi, merancang, membangun, dan mendokumentasikan kebutuhan dari system yang akan kita bangun atau kita buat. Jadi sbelum masuk ke tahap pengcodingan atau *coding system*, dan sebelum msuk ke implemntasi sistem. Kita akan melalui tahap perancangan sistemnya. **"Sistem seperti apa yang akan kita buat?"**, **"siapa aja pengguna dari system ini", "bagaimana prosses sitem ini berjalan", fitur-fitur apa saja nanti yang ada dalam sistem"**. Semua kebutuhan tersebut perlu direncanakan terlebih dahulu, salah satu perencanaan yang dapat dilakukan adalah dengan menggunakan UML ini.



Jenis-jenis UML

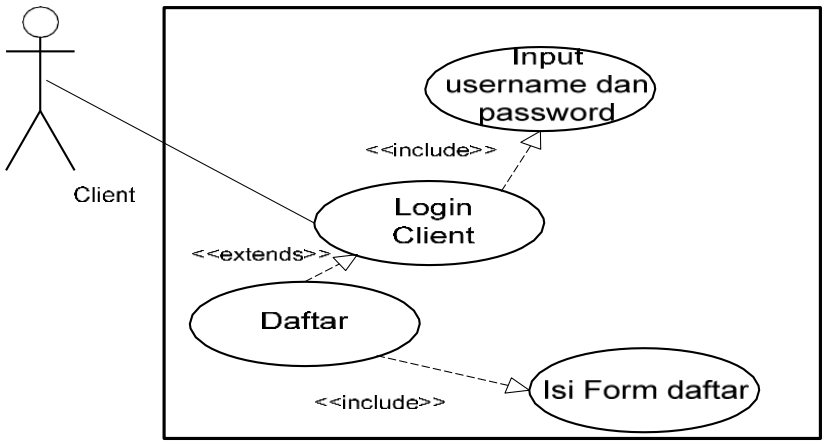


Gambar 10. Jenis-jenis UML

Dapat dilihat pada Gambar 10, terdapat 9 macam diagram, yaitu *use caase diagram*, *activity diagram*, *class diagram*, *state diagram*, dan *entity relationship diagram* atau sering disebut dengan ERD.

a) *Use Case Diagram*

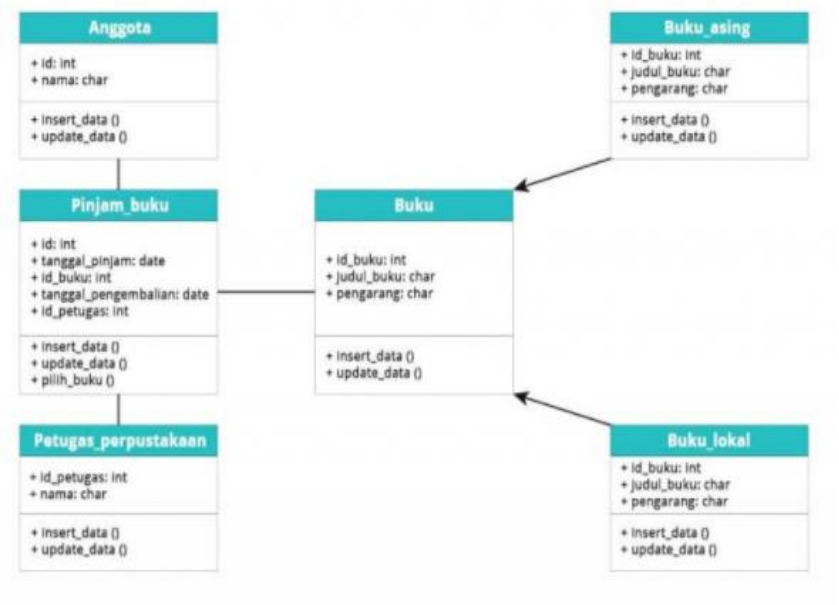
Use case diagram ini sebuah diagram yang menekankan apa yang akan di perbuat *system*, siapa saja yang akan menggunakan *system*. Diagram *use case* ini menggambarkan apa saja yang menjadi kebutuhan *system*, dari sudut pandang *user* atau pengguna.



Gambar 11. Contoh *Use Case Diagram*

b) Class Diagram

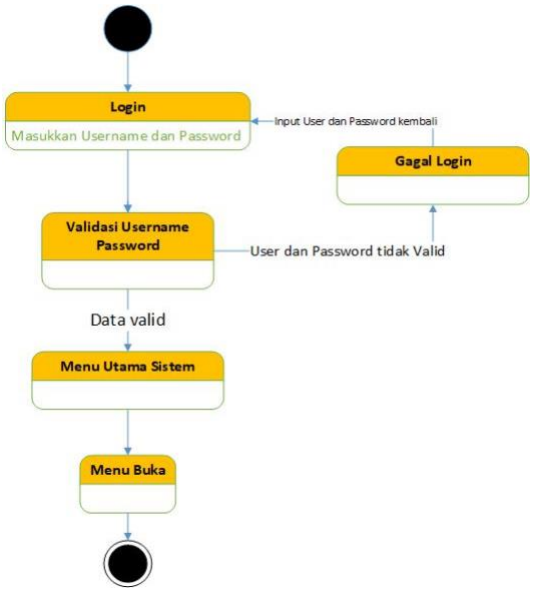
Class diagram adalah salah satu diagram yang terdapat pada struktur UML. *Class diagram* menggambarkan struktur dan deskripsi dari *class*, atribut, metode dan hubungan dari setiap objek. Class diagram dapat mengambarkan keadaan (atribut) suatu system dan menawarkan metode atau fungsi yang digunakan. Class diagram menjelaskan hubungan apa yang terjadi dalam kelas.



Gambar 12. *Class Diagram*

c) State Diagram

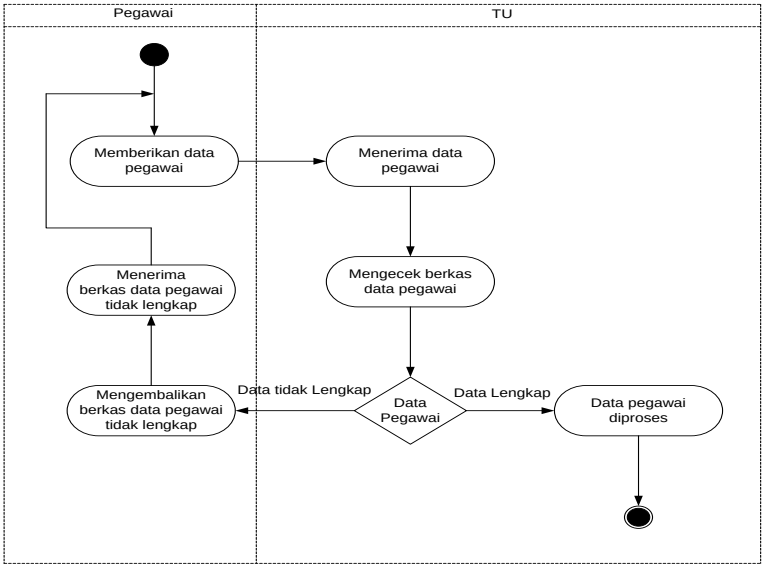
Secara definisi, *State Diagram* disebut juga sebagai *state transtions diagram*. Artinya, *state diagram* menunjukan *state* atau keadaan dari satu objek, *event* atau *massage* yang menyebabkan transisi atau perubahan dari *state* yang satu ke *state* yang lain. *State Diagram* tidak harus dibuat untuk setiap *class* yang ada, kecuali saat dibutuhkan. *State Diagram* berkesinambungan dengan *class diagram*.



Gambar 13. *State Diagram*

d) *Activity Diagram*

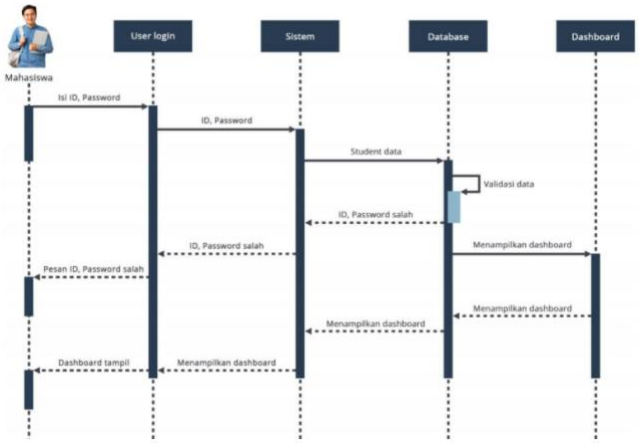
Activity Diagram merupakan sebuah diagram yang menggambarkan dan memvisualisasikan diagram alir yang berisi aktivitas, tindakan, pilihan, perulangan dan *concurrency* untuk menjelaskan aktifitas yang terjadi dalam sebuah sistem ataupun alur aktifitas yang terjadi dalam suatu organisasi.



Gambar 14. *Activity Diagram*

e) ***Sequence Diagram***

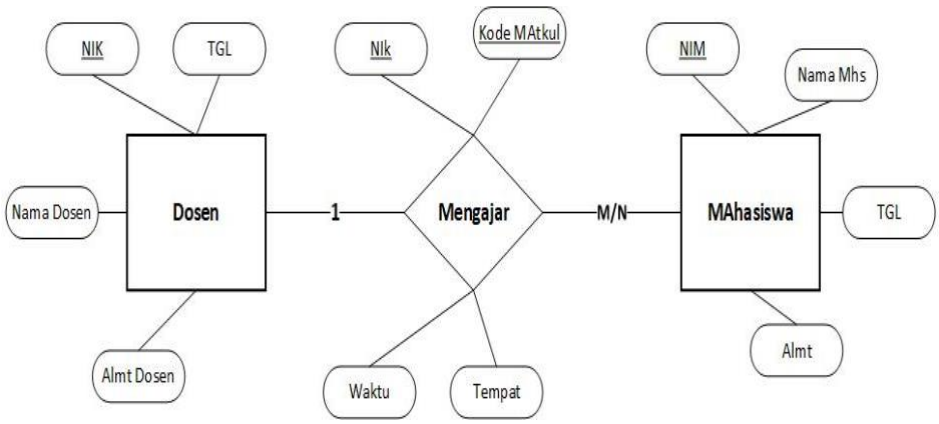
Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri antar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).



Gambar 15. *Sequence diagram*

f) ***Entity Relationship Diagram*** **(ERD)**

ERD merupakan model untuk menjelaskan hubungan antar data dalam basis data berdasarkan suatu persepsi bahwa *real word* terdiri dari objek-objek dasar yang mempunyai hubungan atau relasi antara objek-objek tersebut. ERD juga didefinisikan sebagai diagram yang menggambarkan struktur logis dari basisdata berbasis garis.



Gambar 16. *Entity Relationship Diagram*

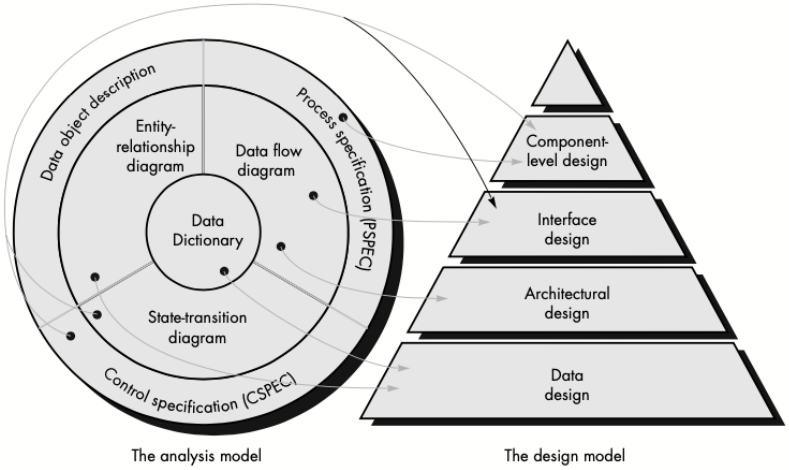
BAB IV

Perancangan Perangkat Lunak

A. Desain Perangkat Lunak

Desain perangkat lunak ini merupakan inti dari teknisnya rekayasa perangkat lunak yang bisa diterapkan menggunakan model proses perangkat lunak manapun. Desain perangkat lunak menjadi urutan pertama dari tiga kegiatan teknis rekayasa perangkat lunak (Desain perangkat lunak, Pembuatan kode, dan Pengujian). Jadi, desain perangkat lunak ini dimulai setelah persyaratan perangkat lunak selesai di analisis dan ditentukan. Kenapa kita perlu menganalisis persayaratan perangkat lunak? Analisis persyaratan sangat dibutuhkan untuk keperluan membangun dan memverifikasi perangkat lunak yang akan dibuat. Jadi, bagaimana cara menganalisis persyaratan perangkat lunaknya?

Menganalisis persyaratan perangkat lunak berkaitan dengan analisis model yang bisa digunakan dengan menggunakan model-model desain yang ada pada UML. Masing-masing dari elemen model analisis akan memberikan informasi yang diperlukan untuk membuat model-model desain yang berguna untuk melengkapi spesifikasi desain.



Gambar 17. Menerjemahkan model analisis ke model desin perangkat lunak

(R. Pressman-2001)

Desain perangkat lunak adalah suatu tahapan pengembangan atau pembuatan perangkat lunak yang menjadi jembatan antara analisis dan implementasi program. Desain

PL sama seperti pendekatan teknik desain yang lainnya, terus berubah-ubah seiring berkembangnya metode-metode baru, analisis yang lebih baik dan pemahaman yang lebih luas. Desain PL juga bisa untuk mendefinisikan arsitektur PL, komponen, modul, antarmuka, pendekatan pengujian, serta data untuk memenuhi kebutuhan yang sudah ditentukan sebelumnya.

B. Desain Proses

Desain proses merupakan urutan langkah-langkah yang memungkinkan desainer atau perancang untuk menggambarkan semua aspek yang ada di perangkat lunak yang akan dibangun. Desain PL memainkan prosesnya secara berulang, di mana persyaratan diterjemahkan ke dalam "Blueprint" untuk membangun perangkat lunak.

C. Prinsip Desain

Prinsip desain merupakan serangkaian langkah interaktif yang memungkinkan perancang menggambarkan semua aspek perangkat lunak yang dibangun. Prinsip dasar dari desain adalah:

1. Proses desain tidak boleh mengalami *"tunnel vision"*
2. Desain harus bisa ditelusuri ke model analisis
3. Hasil desain harus original
4. Mampu mengurangi jarak antara proses PL dengan proses dunia nyata
5. Seragam dan Terintegrasi
6. Terstruktur untuk mengakomodasi perubahan
7. Desain bukan coding, dan coding bukan desain
8. Desain dinilai pada saat dibuat, bukan setelah dibuat
9. Desain harus ditinjau

D. Faktor Kualitas Perangkat Lunak

- Faktor Kualitas Eksternal

  Properti perangkat lunak yang dapat dengan mudah diamati oleh pengguna (misalnya, kecepatan, keandalan, kebenaran, kegunaan).

- Faktor Kualitas Internal

  Mengarah pada desain berkualitas tinggi dari perspektif teknis. Untuk mencapai faktor kualitas internal, perancang harus memahami konsep desain PL.

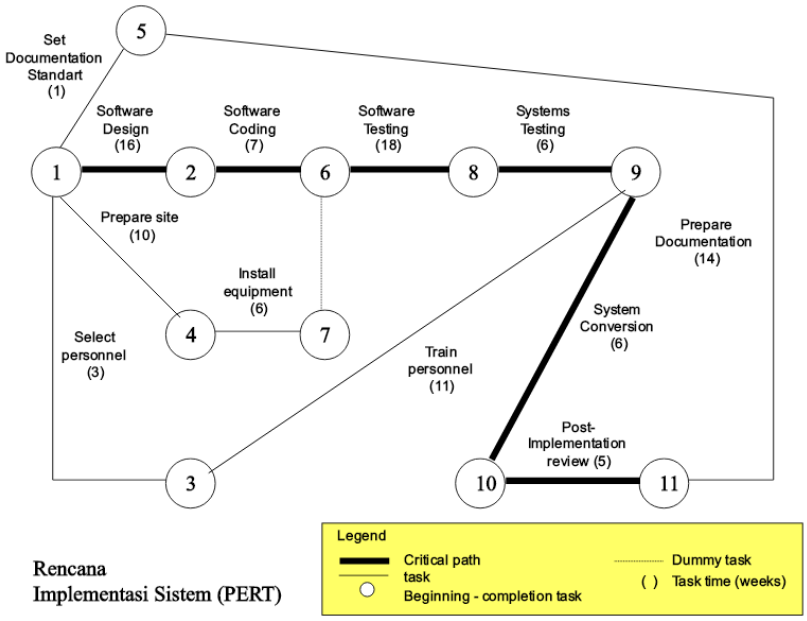
Ketika prinsip-prinsip desain ini diterapkan dengan benar, maka Perancang PL mampu membuat sebuah desain yang menunjukkan faktor kualitas eksternal dan internal.

BAB V

Implementasi dan Pengujian Perangkat Lunak

A. Implementasi Perangkat Lunak

Tahap implementasi pengembangan perangkat lunak adalah Suatu proses pengubahan spesifikasi sistem menjadi sistem yang dapat dijalankan. Artinya tahap implementasi pada sebuah sistem informasi merupakan tahap dimana sistem yang telah dirancang, menjelaskan mengenai pembuatan sistem yang sesuai dengan analisis dan perancangan sebelumnya. Setelah tahap implementasi dilakukan maka dibutuhkan sebuah pengujian sistem untuk membuktikan bahwa aplikasi dapat berjalan sesuai dengan yang diharapkan. Baik buruknya implementasi sangat tergantung pada baik buruknya hasil final dari tahap desain.



Gambar 18. Rencana Implementasi PL

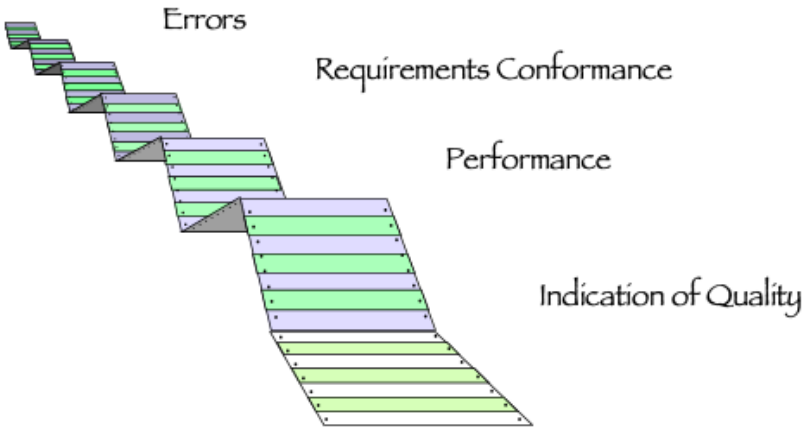
Hal penting dalam implementasi perangkat lunak yaitu:

1. Persiapan Tempat
2. Pelatihan Personil
3. Cakupan Pelatihan
4. Program Pelatihan
5. Teknik dan Alat Bantu Pelatihan

6. Software untuk mendukung pelatihan
7. Persiapan atau pembuatan dokumen
8. Konversi file dan sistem

B. Pengujian Perangkat Lunak

Software Testing adalah proses percobaan penggunaan program dengan menguji segala kemungkinan *error* untuk mendapatkan *software* yang diterima oleh pengguna. *Testing* merupakan basis (*fase*) terakhir dimana kualitas dapat dinilai dan *error* dapat diidentifikasi.



Gambar 19. Hal-hal yang perlu diperhatikan saat *Testing*

Prinsip Pengujian

Prinsip Pengujian adalah sebagai berikut:

- Harus bisa dilacak hingga sampai ke kebutuhan customer.
- Harus direncanakan sejak model dibuat.
- Prinsip Pareto: 80% error uncovered.
- Dari lingkup kecil menuju yang besar.
- Tidak bisa semua kemungkinan diuji.
- Dilakukan oleh pihak ketiga yang independen.

Karakteristik *Software*

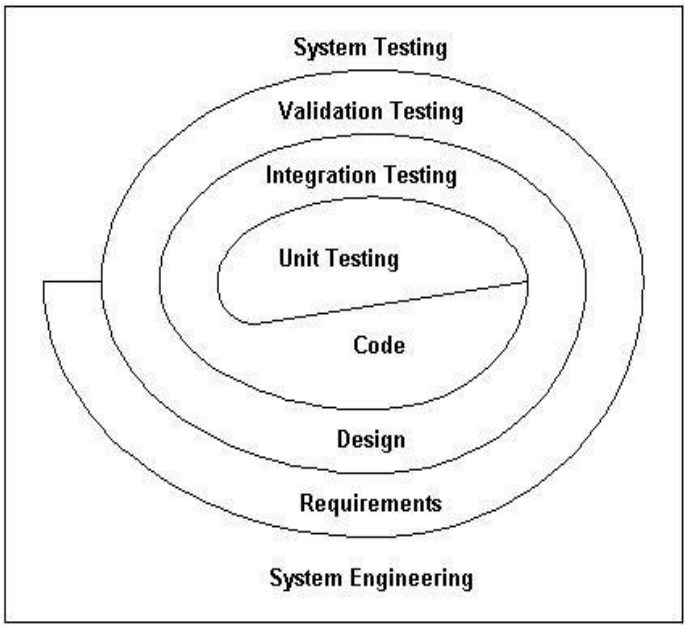
Karakteristik sebuah perangkat lunak yang mudah di uji adalah sebagai berikut:

- *Operability*: mudah digunakan.
- *Observability*: mudah diamati.
- *Controlability*: mudah dikendalikan.
- *Decomposability*: mudah diuraikan.

- *Simplicity*: lingkup kecil, semakin mudah diuji.
- *Stability:* jarang berubah.
- *Understandability*: mudah dipahami

Strategi *Testing*

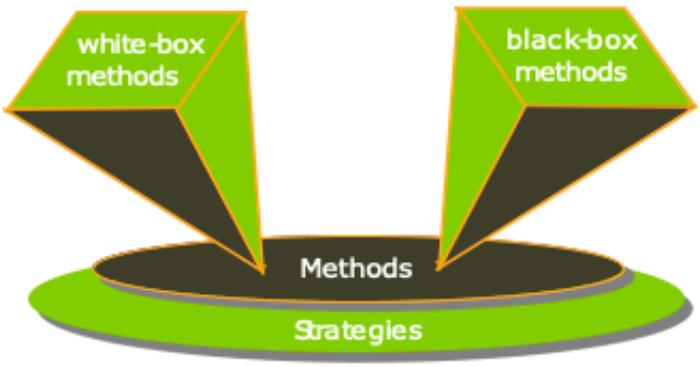
Strategi *testing software* mengintegrasikan metode-metode disain *test cases* ke dalam suatu rangkaian tahapan yang terencana dengan baik sehingga pengembangan *software* dapat berhasil. Strategi menyediakan peta yang menjelaskan **tahap-tahap yang harus dilakukan** sebagai bagian dari *testing*, dan membutuhkan usaha, waktu, serta sumber daya untuk tahap-tahap ini dilaksanakan.



Gambar 20. Strategi *Testing*

Teknik Pengujian Perangkat Lunak

Setelah memahami terkait strategi pengujian perangkat lunak, maka harus juga mengerti metode apa saja yang digunakan untuk menguji *software* yang telah dibuat. Ada 2 cara untuk melakukan pengujian perangkat lunak, yaitu dengan melakukan teknik pengujian *white-box* dan *black-box*.



Gambar 21. Teknik Pengujian PL

a) *White-box*

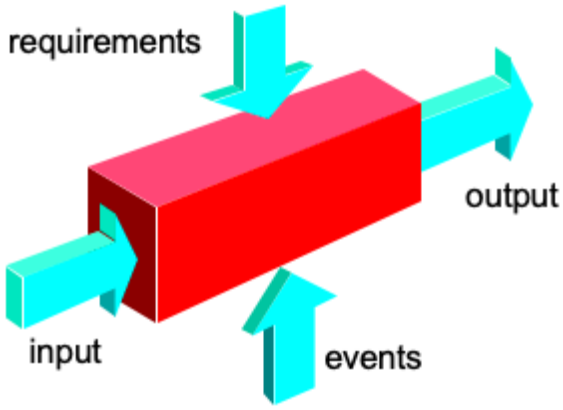
Tujuannya adalah untuk meyakinkan semua. perintah dan kondisi dieksekusi minimal sekali. Kenapa *white-box*? Karena:

- Adanya kesalahan logik dan asumsi yang tidak tepat pada setiap kemungkinan eksekusi.
- Ada kemungkinan alur program yang tidak tereksekusi
- Ada kemungkinan kesalahan typography yang sulit ditemukan kalau tidak dijalankan.

Ada 2 macam teknik *white-box*, yaitu *Basis Path Testing* dan *Control Structure Testing*.

b) *Black-box*.

Pengujian *black-box* disebut juga dengan pengujian perilaku (*behavioral testing*) atau pengujian fungsi (*functional testing*). Fokus dari pengujian ini adalah pada kebutuhan fungsi (*functional requirement*) dari PL.



Gambar 22. Pengujian *Black-box*