

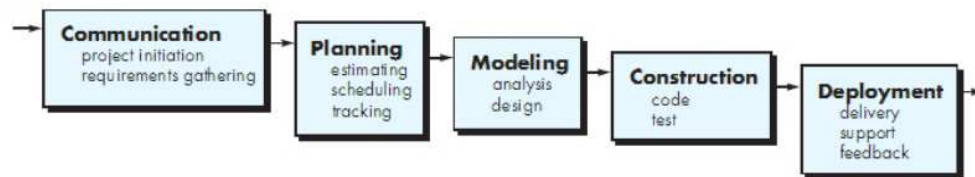
BAB 2

LANDASAN TEORI

2.1 Metode *Waterfall*

Menurut Pressman (2015:42), model *waterfall* adalah model klasik yang bersifat sistematis, berurutan dalam membangun *software*. Nama model ini sebenarnya adalah “*Linear Sequential Model*”. Model ini sering disebut juga dengan “*classic life cycle*” atau metode waterfall. Model ini termasuk ke dalam model *generic* pada rekayasa perangkat lunak dan pertama kali diperkenalkan oleh Winston Royce sekitar tahun 1970 sehingga sering dianggap kuno, tetapi merupakan model yang paling banyak dipakai dalam *Software Engineering* (SE). Model ini melakukan pendekatan secara sistematis dan berurutan. Disebut dengan *waterfall* karena tahap demi tahap yang dilalui harus menunggu selesainya tahap sebelumnya dan berjalan berurutan.

Fase-fase dalam *Waterfall Model* menurut referensi Pressman :



Gambar 2.1 *Waterfall* Pressman (Pressman, 2015:42)

a. *Communication (Project Initiation & Requirements Gathering)*

Sebelum memulai pekerjaan yang bersifat teknis, sangat diperlukan adanya komunikasi dengan *customer* demi memahami dan mencapai tujuan yang ingin dicapai. Hasil dari komunikasi tersebut adalah inisialisasi proyek, seperti menganalisis permasalahan yang dihadapi dan mengumpulkan data-data yang diperlukan, serta membantu mendefinisikan fitur dan fungsi *software*. Pengumpulan data-data tambahan bisa juga diambil dari jurnal, artikel, dan internet.

b. Planning (*Estimating, Scheduling, Tracking*)

Tahap berikutnya adalah tahapan perencanaan yang menjelaskan tentang estimasi tugas-tugas teknis yang akan dilakukan, resiko-resiko yang dapat terjadi, sumber daya yang diperlukan dalam membuat sistem, produk kerja yang ingin dihasilkan, penjadwalan kerja yang akan dilaksanakan, dan *tracking* proses pengerjaan sistem.

c. Modeling (*Analysis & Design*)

Tahapan ini adalah tahap perancangan dan permodelan arsitektur sistem yang berfokus pada perancangan struktur data, arsitektur *software*, tampilan *interface*, dan algoritma program. Tujuannya untuk lebih memahami gambaran besar dari apa yang akan dikerjakan.

d. Construction (*Code & Test*)

Tahapan *Construction* ini merupakan proses penerjemahan bentuk desain menjadi kode atau bentuk/bahasa yang dapat dibaca oleh mesin. Setelah pengkodean selesai, dilakukan pengujian terhadap sistem dan juga kode yang sudah dibuat. Tujuannya untuk menemukan kesalahan yang mungkin terjadi untuk nantinya diperbaiki.

e. Deployment (*Delivery, Support, Feedback*)

Tahapan *Deployment* merupakan tahapan implementasi *software* ke *customer*, pemeliharaan *software* secara berkala, perbaikan *software*, evaluasi *software*, dan pengembangan *software* berdasarkan umpan balik yang diberikan agar sistem dapat tetap berjalan dan berkembang sesuai dengan fungsinya. (Pressman, 2015:17)

Kapan sebaiknya metode *waterfall* digunakan? Ada teori-teori yang menyimpulkan beberapa hal, yaitu :

1. Ketika semua persyaratan yang diajukan sudah dipahami dengan baik pada awal pengembangan program

2. Definisi produk bersifat stabil dan tidak ada perubahan yang dilakukan saat pengembangan untuk alasan apapun. Oleh karena itu, teknologi yang digunakan juga harus sudah dipahami dengan baik
3. Menghasilkan produk baru, atau produk dengan versi baru. Sebenarnya, jika menghasilkan produk dengan versi baru maka itu sudah termasuk *incremental development*, yang setiap tahapannya sama dengan metode *waterfall* kemudian diulang-ulang
4. *Port-ing* produk yang sudah ada ke dalam *platform* baru

Dengan demikian, metode *waterfall* dianggap pendekatan yang lebih cocok digunakan untuk proyek pembuatan sistem baru dan juga pengembangan *software* dengan tingkat resiko yang kecil serta waktu pengembangan yang cukup lama. Tetapi salah satu kelemahan paling mendasar adalah menyamakan pengembangan *hardware* dan *software* dengan meniadakan perubahan saat pengembangan. Padahal, *error* diketahui saat *software* dijalankan, dan perubahan-perubahan akan sering terjadi.

Keuntungan menggunakan metode *waterfall* adalah prosesnya lebih terstruktur, hal ini membuat kualitas *software* baik dan tetap terjaga. Dari sisi *user* juga lebih menguntungkan, karena dapat merencanakan dan menyiapkan kebutuhan data dan proses yang diperlukan sejak awal. Penjadwalan juga menjadi lebih menentu, karena jadwal setiap proses dapat ditentukan secara pasti. Sehingga dapat dilihat jelas target penyelesaian pengembangan program. Dengan adanya urutan yang pasti, dapat dilihat pula perkembangan untuk setiap tahap secara pasti. Dari sisi lain, model ini merupakan jenis model yang bersifat dokumen lengkap sehingga proses pemeliharaan dapat dilakukan dengan mudah.

Kelemahan menggunakan metode *waterfall* adalah bersifat kaku, sehingga sulit melakukan perubahan di tengah proses. Jika terdapat kekurangan proses/prosedur dari tahap sebelumnya, maka tahapan pengembangan harus dilakukan mulai dari awal lagi. Hal ini akan memakan waktu yang lebih lama. Karena jika proses sebelumnya belum selesai sampai akhir, maka proses selanjutnya juga tidak dapat berjalan. Oleh karena itu, jika terdapat kekurangan dalam permintaan *user* maka proses pengembangan harus dimulai kembali dari

awal. Karena itu, dapat dikatakan proses pengembangan *software* dengan metode *waterfall* bersifat lambat.

Kelemahan lainnya menggunakan metode *waterfall* adalah membutuhkan daftar kebutuhan yang lengkap sejak awal. Tetapi, biasanya jarang sekali *customer* yang dapat memenuhi itu. Untuk menghindari pengulangan tahap dari awal, *user* harus memberikan seluruh prosedur, data, dan laporan yang diinginkan mulai dari tahap awal pengembangan. Tetapi pada banyak kondisi, *user* sering melakukan permintaan di tahap pertengahan pengembangan sistem. Dengan metode ini, maka *development* harus dilakukan mulai lagi dari tahap awal. Karena *development* disesuaikan dengan desain hasil *user* pada saat tahap pengembangan awal. Di sisi lain, *user* tidak dapat mencoba sistem sebelum sistem benar-benar selesai. Selain itu, kinerja personil menjadi kurang optimal karena terdapat proses menunggu suatu tahap selesai terlebih dahulu. Oleh karena itu, seringkali diperlukan personil yang “*multi-skilled*” sehingga minimal dapat membantu pengerjaan untuk tahapan berikutnya. (Pressman, 2015:42-43)

2.2 *Unified Modeling Language*

Menurut Whitten & Bentley (2007:371), *Unified Modeling Language (UML)* versi 2.0 adalah sekumpulan konversi pemodelan yang digunakan untuk menentukan atau menggambarkan sebuah sistem *software* yang terkait dengan objek.

UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang, UML sudah mulai banyak digunakan oleh para praktisi OOP. UML juga merupakan dasar bagi *design tools* berorientasi objek pada IBM. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson.

Sampai era tahun 1990, puluhan metodologi permodelan berorientasi objek telah bermunculan di dunia. Di antaranya adalah : metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa *method*

war dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbulnya masalah baru apabila kita bekerjasama dengan kelompok/perusahaan lain yang menggunakan metodologi yang berlainan.

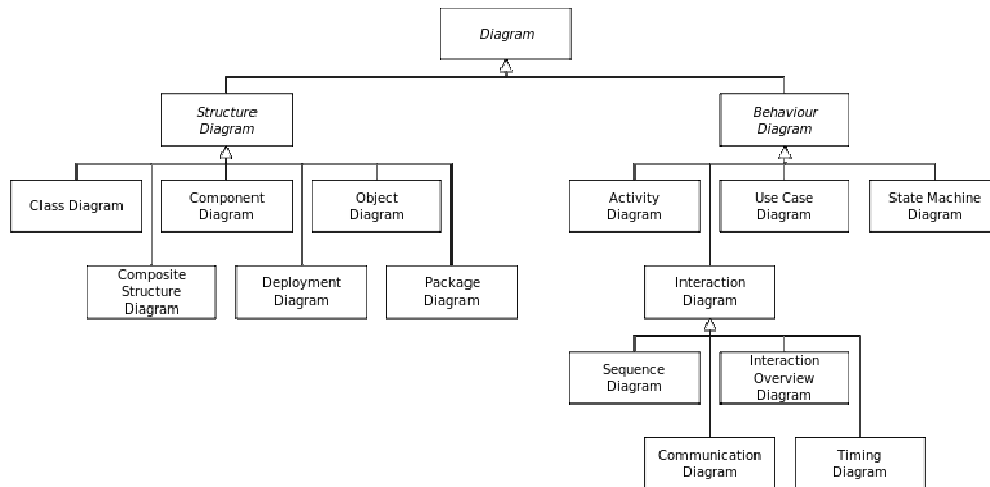
Dimulai pada bulan Oktober 1994 Booch, Rumbaugh, dan Jacobson, yang merupakan tiga tokoh yang boleh dikatakan metodologinya banyak digunakan memelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995, dirilis *draft* pertama dari UML (versi 0.8). Sejak tahun 1996, pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG - <http://www.omg.org>)

Menurut Whitten & Bentley (2007:382), UML menyediakan tiga belas macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

1. *Use Case Diagram* menggambarkan interaksi antara sistem internal, sistem eksternal, dan *user*. Dengan kata lain, secara grafik menjelaskan siapa yang menggunakan sistem, dan dengan cara apa *user* berinteraksi dengan sistem.
2. *Activity Diagram* menggambarkan alur *sequential* dari aktivitas sebuah proses bisnis atau *Use Case*. Bisa juga digunakan untuk memodelkan logika yang digunakan sistem.
3. *Class Diagram* menggambarkan struktur objek sistem. Menunjukkan kelas yang menjadi komponen dari sistem, serta hubungan antar kelas.
4. *Object Diagram* serupa dengan *Class Diagram*, memodelkan instansi objek yang sebenarnya beserta nilai atributnya.
5. *State Machine Diagram* untuk memodelkan perilaku objek di dalam sistem terhadap kejadian (*event*) selama masa hidupnya.
6. *Composite Structure Diagram* menguraikan struktur internal, komponen, atau *Use Case* dari suatu kelas.
7. *Sequence Diagram* menggambarkan bagaimana objek berinteraksi melalui pengiriman pesan (*message*) dalam pengeksekusi sebuah *Use Case* atau operasi tertentu.
8. *Communication Diagram* disebut juga *Collaboration Diagram*, mirip dengan *Sequence Diagram*. Namun, *Sequence Diagram* lebih berfokus

pada pemilihan waktu atau urutan pesan. *Communication Diagram* berfokus pada penyusunan struktur objek dalam bentuk jaringan.

9. *Interaction Overview Diagram* mengkombinasikan *Activity Diagram* dengan *Sequence Diagram* untuk menunjukkan bagaimana objek berinteraksi dalam tiap aktivitas *Use Case*.
10. *Timing Diagram* adalah diagram interaksi lain yang berfokus pada batasan pemilihan waktu dalam keadaan satu objek atau kumpulan objek yang berubah. Diagram ini sangat berguna ketika mendesain *embedded software* untuk banyak perangkat.
11. *Component Diagram* menggambarkan penyusunan kode *programming* yang dibagi menjadi beberapa komponen dan menjelaskan bagaimana komponen tersebut berinteraksi.
12. *Deployment Diagram* menggambarkan konfigurasi dari komponen *software* dalam arsitektur fisik dari “simpul - simpul” sistem *hardware*.
13. *Package Diagram* menggambarkan bagaimana kelas/konstruksi dari UML lain disusun dalam bentuk paket (berkaitan dengan paket *Java* atau *C++* dan *namespaces* dari *.NET*) dan ketergantungannya antar paket.



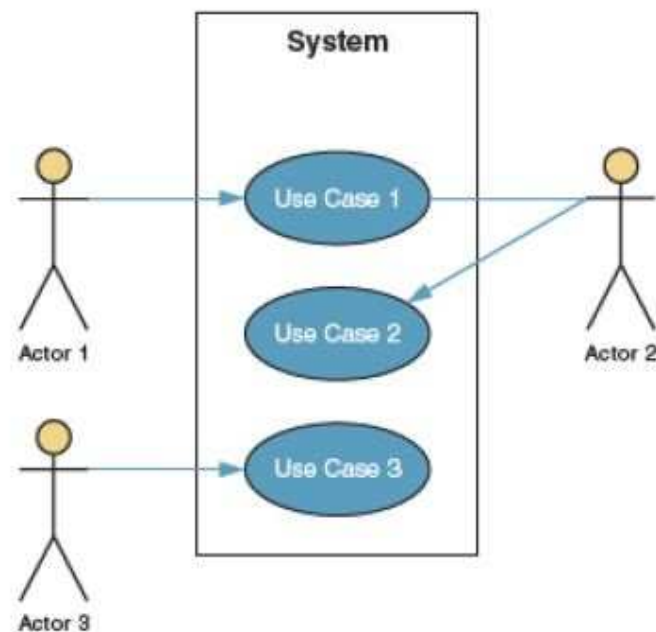
Gambar 2.2 Diagram UML

Berikut akan dijelaskan tiga macam diagram yang paling sering digunakan dalam pembangunan aplikasi berorientasi objek dan digunakan juga pada penulisan ini, yaitu:

1. *Use Case Diagram*

Use Case Diagram merupakan diagram yang menggambarkan interaksi antara sebuah sistem internal, eksternal, dan *user*. Dengan kata lain, menggambarkan siapa saja yang akan menggunakan sistem dan dengan cara seperti apa *user* dapat berinteraksi dengan sistem. (Whitten & Bentley, 2007:246)

Berikut contoh dari *Use Case Diagram* :



Gambar 2.3 Contoh *Use Case Diagram*
(Whitten & Bentley, 2007:246)

Pada Gambar 2.3, terlihat bahwa ada tiga komponen penting dalam *Use Case Diagram* yaitu:

1. *Use Case*

Use Case mendeskripsikan fungsi dari sistem dari perspektif *user* dalam kondisi yang dapat dimengerti *user*. Digambarkan dalam bentuk elips dengan nama *Use Case* di dalamnya.

2. Actor

Actor merupakan *user* yang akan berinteraksi dengan sistem untuk saling bertukar informasi. Digambarkan berupa *stick figure* dengan nama *Actor* di bawahnya.

3. Relationship

Relationship merupakan hubungan antara *Use Case* dan *Actor* yang digambarkan dalam bentuk garis. *Relationship* dibagi menjadi lima, yaitu :

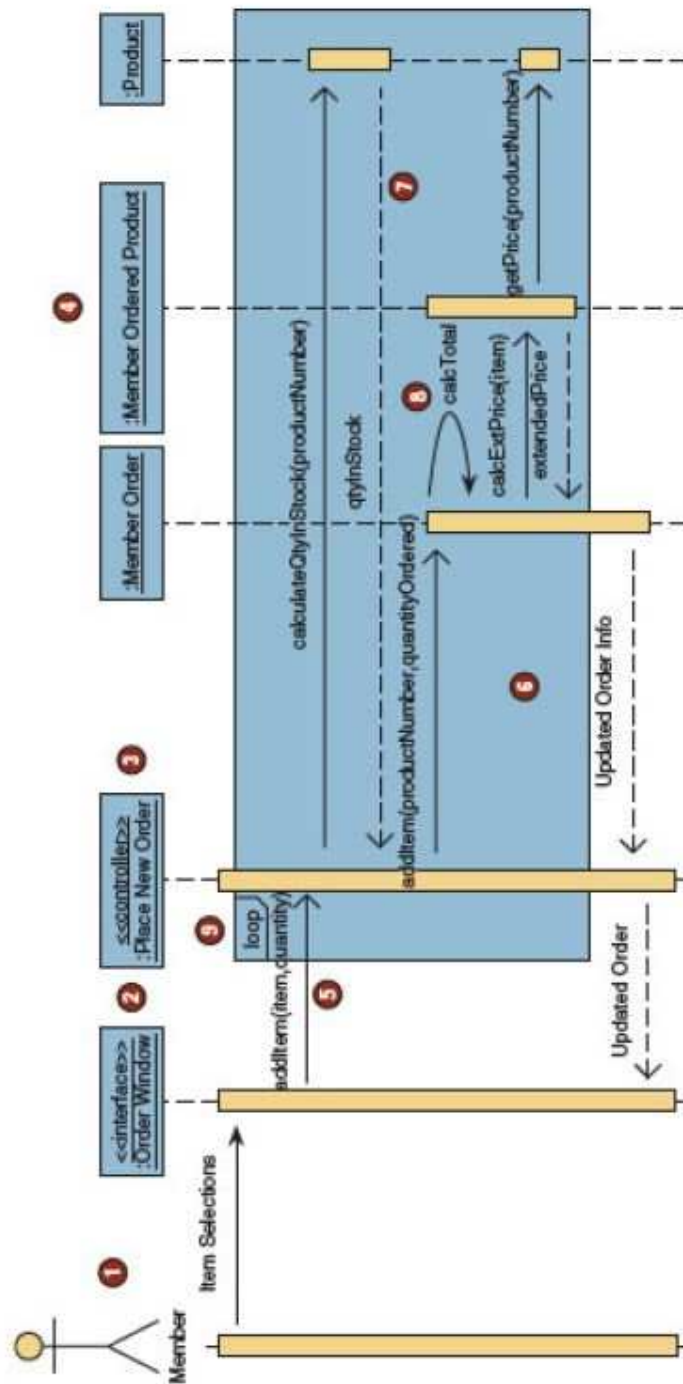
- Asosiasi (*Associations*)
Hubungan antara *Actor* dan *Use Case* terjadi ketika *Use Case* menggambarkan sebuah garis penghubung antara *Use Case* dengan *Actor*
- Perluasan (*Extends*)
Use Case berisikan fungsi rumit yang terdiri dari beberapa tahap pembuatan sebuah logika *Use Case* yang sulit untuk dimengerti.
- Includes
Use Case dapat menurunkan *redudancy* terhadap dua *Use Case* atau lebih melalui langkah kombinasi umum dalam kasus itu sendiri.
- Ketergantungan (*Depends On*)
Hubungan yang terjadi antar *Use Case* menunjukkan bahwa satu *Use Case* tidak dapat berjalan jika *Use Case* yang lain tidak dijalankan.
- Pewarisan (*Inheritance*)
Hubungan antara *Actor* menciptakan gambaran yang sederhana ketika pelaku abstrak mewarisi tugas dari *multiple real actors*.

2. Sequence Diagram

Sequence Diagram adalah diagram UML yang memodelkan logika dari sebuah *use case* dengan cara menggambarkan bagaimana interaksi antar objek satu sama lain dalam suatu urutan waktu. Diagram ini

mengilustrasikan bagaimana pesan dikirim dan diterima antar objek pada suatu waktu. (Whitten & Bentley, 2007:659)


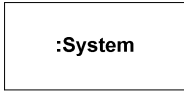


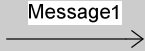
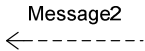

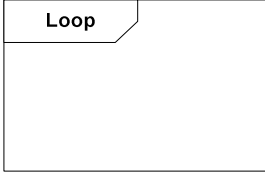
Berikut contoh dari *Sequence Diagram* :



Gambar 2.4 Contoh *Sequence Diagram* (Whitten & Bentley, 2007:659)

Berikut merupakan tabel notasi yang digunakan pada *Sequence Diagram* yang dijelaskan pada Tabel 2.1 berikut ini :

Tabel 2.1 – Notasi *Sequence Diagram* (Whitten & Bentley, 2007:660)

Notasi	Keterangan	Simbol
<i>Actor</i>	Menggambarkan <i>user</i> yang berinteraksi dengan sistem	
<i>System</i>	Menggambarkan <i>instance</i> dari sebuah <i>class</i> pada <i>class diagram</i>	
<i>Lifelines</i>	Menggambarkan keberadaan sebuah objek dalam suatu waktu atau waktu dari <i>sequence</i>	
<i>Activation Bars</i>	Menggambarkan waktu di mana <i>user</i> sedang aktif berinteraksi dengan sistem	
<i>Input Messages</i>	Menggambarkan pesan masuk yang dikirimkan berupa <i>behavior</i>	
<i>Output Messages</i>	Menggambarkan balasan dari pesan masuk yang berupa <i>attribute</i>	
<i>Receiver Actor</i>	Aktor lainnya atau sistem external yang menerima pesan dari sistem	
<i>Frame</i>	Menggambarkan area pada sistem yang mengalami perulangan (<i>loop</i>), seleksi (<i>alternate fragments</i>), atau kondisi opsional (<i>optional</i>)	

3. *Class Diagram*

Class Diagram adalah sebuah diagram yang menggambarkan struktur objek statis di dalam sistem. Diagram ini menunjukkan kelas-kelas objek yang menyusun sistem dan juga menghubungkan antar kelas objek tersebut. *Class* dapat berhubungan dengan yang lain melalui berbagai cara: *Association* (hubungan antara *class*), *Aggregation* (hubungan di mana suatu *class* merupakan bagian dari *class* lain), *Composition* (hubungan di mana suatu *class* merupakan bagian wajib dari *class* lain), *Generalization* (satu *class* merupakan spesialisasi dari *class* lainnya). Elemen utama dari sebuah *Class Diagram* adalah kotak yang lambangnya digunakan untuk mewakili *class* dan *interface*. Kotak tersebut dibagi menjadi tiga bagian, di mana bagian atas merupakan nama *class*, bagian tengah adalah atribut, dan bagian bawah adalah operasi dari suatu *class*. (Whitten & Bentley, 2007:400)








Flowchart terbagi atas lima jenis, yaitu :






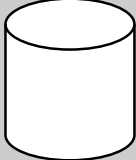
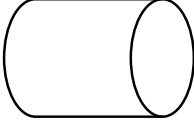




1. *Flowchart Sistem (System Flowchart)*
2. *Flowchart Paperwork / Flowchart Dokumen (Dokumen Flowchart)*
3. *Flowchart Skematik (Schematic Flowchart)*
4. *Flowchart Program (Program Flowchart)*
5. *Flowchart Proses (Process Flowchart)*



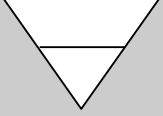
Simbol-simbol *flowchart* yang biasanya dipakai adalah simbol-simbol *flowchart* standar yang dikeluarkan oleh ANSI dan ISO. (Subrata, 2010)

Simbol-simbol ini dapat dilihat pada Tabel 2.2 – Simbol *Flowchart* Standar berikut ini :

Tabel 2.2 – Simbol *Flowchart* Standar (Subrata, 2010:9-13)

Nama	Simbol	Arti
<i>Input/Output</i>		Merepresentasikan <i>Input</i> data atau <i>Output</i> data yang diproses atau Informasi
Proses		Mempresentasikan operasi
Penghubung		Keluar ke atau masuk dari bagian lain <i>flowchart</i> khususnya halaman yang sama
Anak Panah		Merepresentasikan alur kerja
Penjelasan		Digunakan untuk komentar tambahan
Keputusan		Keputusan dalam program
<i>Predefined Process</i>		Rincian operasi berada di tempat lain

Nama	Simbol	Arti
<i>Preparation</i>		Pemberian harga awal
<i>Terminal Points</i>		Awal.akhir <i>flowchart</i>
<i>Punched Card</i>		<i>Input/Output</i> yang menggunakan kartu berlubang
<i>Dokumen</i>		I/O dalam format yang dicetak
<i>Magnetic Tape</i>		I/O yang menggunakan pita magnetik
<i>Magnetic Disk</i>		I/O yang menggunakan disk magnetik
<i>Magnetic Drum</i>		I/O yang menggunakan drum magnetik
<i>Online Storage</i>		I/O yang menggunakan penyimpanan akses langsung
<i>Punched Tape</i>		I/O yang menggunakan pita kertas berlubang
<i>Manual Input</i>		Input yang dimasukkan secara manual dari keyboard
<i>Display</i>		<i>Output</i> yang ditampilkan pada terminal

Nama	Simbol	Arti
<i>Manual Operation</i>		Operasi Manual
<i>Communication Link</i>		Transmisi data melalui channel komunikasi
<i>Offline Storage</i>		Penyimpanan yang tidak dapat diakses oleh komputer secara langsung

2.4 Delapan Aturan Emas

Menurut Shneiderman & Plaisant (2010: 88-89), terdapat delapan aturan yang harus diperhatikan dan menjadi dasar dalam merancang desain antarmuka suatu sistem. Aturan tersebut dikenal dengan sebutan delapan aturan emas (*8 Golden Rules*). Delapan aturan emas tersebut adalah:

1. Berusaha untuk konsisten
Selain konsistensi dari sisi sistem, diperlukan pula rangkaian perintah/tindakan yang konsisten pada kondisi yang sama. Konsistensi ini diperlukan pada *prompts*, *menus*, dan layar bantu. Selain itu, hal-hal yang juga perlu diperhatikan adalah warna, *layout*, huruf kapital, jenis huruf, penggunaan istilah, dan lain-lain.
2. Memungkinkan penggunaan secara umum
Kemampuan pengguna dapat dibagi menjadi dua kategori, yaitu pemula (*novice*) dan ahli (*expert*). Karena adanya perbedaan kemampuan pengguna, diperlukan rancangan yang dapat memudahkan perubahan-perubahan pada konten sesuai dengan kemampuan pengguna. Contohnya, penjelasan untuk pengguna pemula dan penyediaan fitur *shortcut* untuk pengguna ahli.
3. Memberikan umpan balik yang informatif
Sebisa mungkin, sistem harus memberikan umpan balik untuk setiap tindakan yang dilakukan pengguna. Untuk tindakan yang sering dilakukan dan tidak memerlukan banyak aksi, sistem dapat memberikan

umpan balik yang sederhana. Sedangkan, tindakan yang jarang dilakukan dan memerlukan banyak aksi harus diberikan umpan balik yang jelas.

4. Merancang dialog untuk menghasilkan suatu penutupan
 Suatu aksi seharusnya dilakukan secara berurutan mulai dari tahap awal hingga tahap akhir. Umpan balik yang informatif akan memberikan informasi yang jelas saat akhir dari suatu aksi yang menjelaskan bahwa itu adalah akhir dari aksi tersebut dan siap untuk melanjutkan ke proses berikutnya.
5. Memberikan pencegahan dan penanganan kesalahan sederhana
 Usahakan untuk membuat suatu sistem sedemikian rupa agar pengguna dapat terhindar dari *error* atau tidak melakukan *error* yang serius. Sistem harus dapat mendeteksi *error* itu sebelum terjadi. Jika *error* tersebut masih terjadi, sistem harus dapat mendeteksi dan memberikan mekanisme yang sederhana dan mudah dipahami pengguna untuk menangani *error* tersebut.
6. Mudah untuk kembali ke tindakan sebelumnya
 Hal ini dapat mengurangi kekhawatiran pengguna karena pengguna jadi mengetahui bahwa kesalahan yang dilakukan dapat dibatalkan, sehingga pengguna lebih berani untuk mengeksplorasi opsi lainnya yang belum biasa digunakan pengguna.
7. Mendukung pengguna sebagai pusat kendali internal
 Rancang suatu *interface* sehingga pengguna menjadi inisiator (pengendali sistem) bukanlah sebagai responden dari sistem.
8. Mengurangi beban ingatan jangka pendek
 Buat *interface* sesederhana mungkin dan mudah dipahami pengguna. Beberapa halaman dapat dijadikan satu, frekuensi pergerakan *window* dapat dikurangi, dan harus ada waktu yang cukup bagi pengguna untuk mempelajari kode, singkatan, serta urutan aksi. Sebaiknya, informasi seperti kode atau singkatan juga disediakan.

2.5 Steganografi

Steganografi adalah teknik menyembunyikan pesan rahasia ke dalam suatu media sehingga pesan tersebut menjadi tidak diketahui. Media yang biasa digunakan berupa gambar atau dapat disebut dengan *Cover-Image*. *Cover-Image* yang sudah diselipkan pesan rahasia disebut dengan *Stego-Image*. (Chang et al, 2008:37)

Menurut Rojali (2009), steganografi adalah sebuah mekanisme untuk melindungi data. Data yang akan dikirimkan dapat disisipkan melalui media pembawa.

Tujuan dari steganografi adalah merahasiakan atau menyembunyikan keberadaan dari sebuah pesan tersembunyi atau sebuah informasi. Pada prakteknya, kebanyakan pesan disembunyikan dengan membuat perubahan kecil terhadap data digital lain yang isinya tidak akan menarik perhatian dari pihak ketiga (penyerang potensial). Perubahan ini bergantung pada kunci (sama dengan kriptografi) dan pesan yang disembunyikan. Pihak penerima pesan kemudian dapat menyimpulkan informasi terselubung tersebut dengan cara menggunakan kunci yang benar ke dalam algoritma yang digunakan.

Pada metode steganografi, cara ini sangat berguna jika digunakan pada cara steganografi komputer karena banyak format berkas digital yang dapat dijadikan media untuk menyembunyikan pesan.

Format yang biasa digunakan antara lain:

1. Format *Image* : bmp, png, jpg, jpeg, dll.
2. Format Audio : wav, voc, mp3, dll.
3. Format lainnya : teks file, html, pdf, dll.

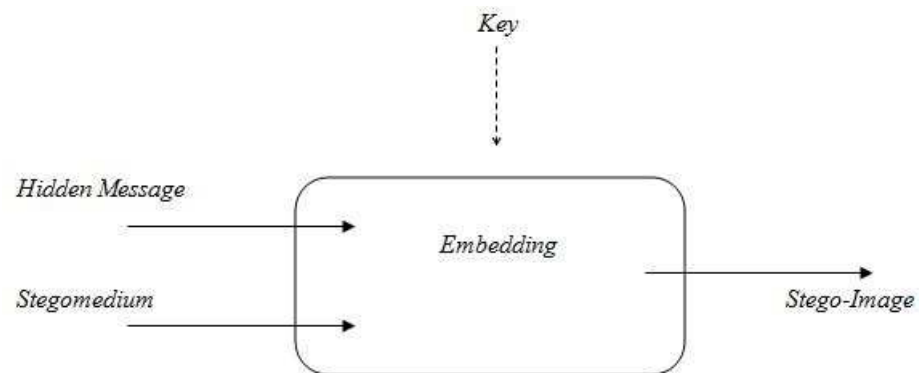
Kelebihan steganografi jika dibandingkan dengan kriptografi adalah pesan-pesannya tidak menarik perhatian orang lain. Pesan-pesan berkode dalam kriptografi yang tidak disembunyikan, walaupun tidak dapat dipecahkan tetapi akan menimbulkan kecurigaan. Seringkali, steganografi dan kriptografi digunakan secara bersamaan untuk menjamin keamanan pesan rahasianya.

Steganografi memiliki dua properti utama yaitu wadah penampung dan pesan rahasia yang akan disembunyikan. Seperti yang sudah dipaparkan

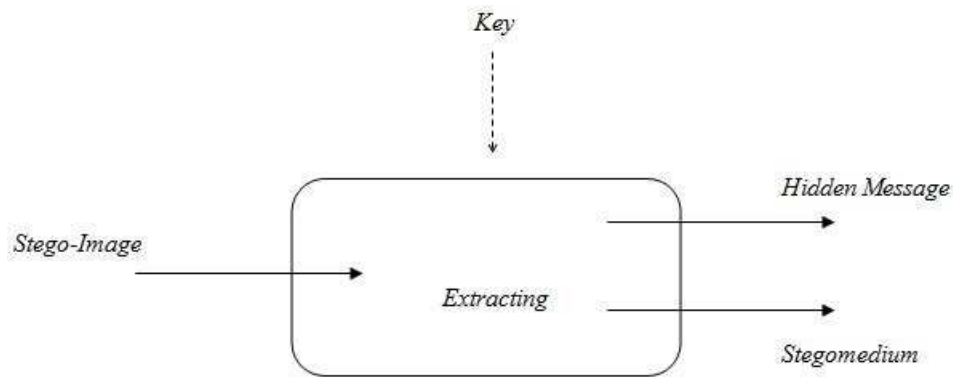
sebelumnya, banyak format berkas digital yang dapat dijadikan media untuk menyembunyikan pesan. Media tersebut dapat diartikan sebagai wadah penampung. Wadah penampung dan pesan rahasia tersebut dapat berupa gambar, audio, teks, atau video.

Steganografi juga memiliki dua proses utama yaitu proses penyisipan pesan (*Embedding*) dan juga proses ekstraksi pesan (*Extraction*). Selain itu, steganografi juga memiliki 4 komponen penting :

1. *Hidden Message* : pesan yang ingin disembunyikan
2. *Cover-Image* : media yang digunakan untuk penyembunyian *Hidden Message* (*Stegomedium*)
3. *Stego-Image* : media yang sudah berisi *Hidden Message*
4. *Key* : kunci yang digunakan untuk proses *embedding* dan ekstraksi pesan dari *Stego-Image*



Gambar 2.6 Diagram Proses *Embedding*



Gambar 2.7 Diagram Proses *Extraction*

Munir (2004:4) juga mengatakan bahwa dalam steganografi juga perlu memperhatikan beberapa kriteria dalam penyembunyian data, kriteria tersebut antara lain :

1. *Fidelity*

Kualitas media penampung tidak banyak berubah setelah disisipkan pesan rahasia. Perubahan tersebut harus tidak terlihat oleh mata telanjang/tidak mencolok, sehingga pihak luar tidak menyadari kalau media penampung tersebut sudah diubah dan terdapat data rahasia di dalamnya

2. *Robustness*

Pesan rahasia yang disisipkan harus tahan terhadap manipulasi yang dilakukan terhadap media penampung. Misalnya, media penampungnya berupa gambar. Jika gambar tersebut dimanipulasi (seperti diubah kontrasnya, dirotasi, dll.), pesan rahasia yang disisipkan tidak rusak

3. *Recovery*

Pesan rahasia yang disisipkan tersebut harus dapat dipulihkan kembali. Seperti tujuan awal steganografi yaitu penyembunyian pesan, maka sewaktu-waktu pesan rahasia ini harus dapat diambil kembali untuk dapat digunakan.

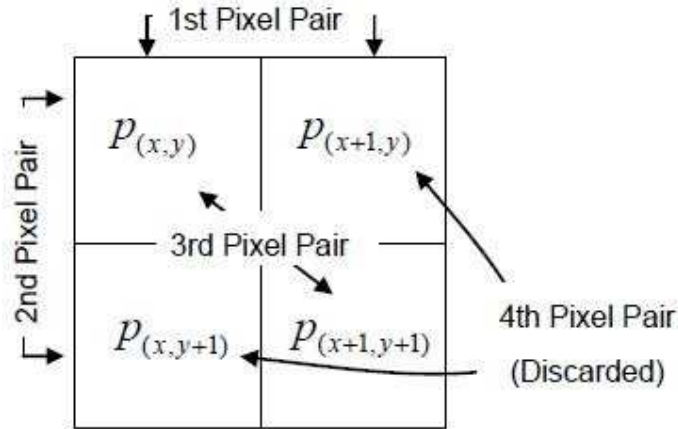
2.6 Metode *Tri-Way Pixel Value Differencing*

Metode *Tri-Way Pixel Value Differencing* (TPVD) merupakan metode steganografi pengembangan dari metode *Pixel Value Differencing* (PVD). Metode ini dikemukakan oleh Ko-Chin Chang, Chien-Ping Chang, Ping S. Huang, dan Te-Ming Tu yang berasal dari Taiwan pada Juni 2008. Metode PVD telah terbukti mampu menyediakan kapasitas penyisipan yang besar dan kualitas *Stego-Image* yang baik. Untuk meningkatkan kapasitas penyisipan metode PVD yang memanfaatkan dua *pixel* horisontal dan berurutan yang hanya dapat merepresentasikan sebuah *vertical edge*, tiga arah *edge* dipertimbangkan dan digunakan secara efektif dalam desain TPVD.

Berbeda dengan PVD, TPVD merupakan teknik yang memanfaatkan selisih nilai dari empat *pixel* yang diambil dengan ukuran matriks 2x2. Kapasitas pesan yang dapat disisipkan bergantung dari selisih ke-empat nilai tersebut. Pada metode PVD, dua *pixel* yang dapat digunakan adalah *pixel* yang berurutan secara horisontal dan vertikal. Namun, garis tidak hanya bersifat horisontal dan vertikal karena garis juga memiliki arah yang berbeda (dua garis diagonal).

2.6.1 Prosedur Partisi

Termotivasi dari metode PVD, menggunakan dua *pixel* horisontal dan berurutan dapat bekerja dengan efisien untuk menyembunyikan pesan. TPVD harusnya dapat menyelesaikan kasus menyembunyikan pesan dengan lebih efisien dengan menggunakan empat arah *edge* dari empat *two-pixel pairs*. TPVD dapat diimplementasikan dengan cara membagi gambar menjadi beberapa blok berukuran 2x2, seperti contoh yang dapat dilihat pada Gambar 2.8 berikut ini. Namun, karena perubahan nilai *pixel pairs* keempat mempengaruhi nilai *pixel pairs* pertama dan kedua, maka *pixel pairs* keempat dapat dikatakan tidak berguna dan dapat dibuang. Oleh karena itu, hanya tiga pasang *pixel* saja yang diperlukan untuk penyisipan pesan rahasia. (Chang et al, 2008:38-39)



Gambar 2.8 Contoh 4 Pixel Pairs

2.6.2 Skema *Tri-Way Differencing*

Seperti yang dapat dilihat pada Gambar 2.8, setiap blok 2x2 memiliki empat *pixel* yaitu $p_{(x,y)}$, $p_{(x+1,y)}$, $p_{(x,y+1)}$, $p_{(x+1,y+1)}$ di mana x dan y merupakan lokasi *pixel* pada gambar. Anggap $p_{(x,y)}$ sebagai titik awal, maka terbentuklah tiga pasangan *pixel* dengan mengelompokkan $p_{(x,y)}$ dengan $p_{(x+1,y)}$, $p_{(x,y+1)}$, dan $p_{(x+1,y+1)}$. Namakan pasangan tersebut dengan P_0 , P_1 , P_2 di mana :

$$P_0 = (p_{(x,y)}, p_{(x+1,y)}), P_1 = (p_{(x,y)}, p_{(x,y+1)}), P_2 = (p_{(x,y)}, p_{(x+1,y+1)})$$

Saat menggunakan metode TPVD untuk menyisipkan pesan rahasia, setiap pasangan mempunyai hasil modifikasi (P_i') dan nilai selisih baru (d_i') dimana $i = 0,1,2$. Sekarang, didapatlah nilai *pixel* baru pada setiap pasangan yang nilainya berbeda dengan nilai aslinya. Karena itu, terdapat tiga nilai baru yang berbeda untuk titik awal $p_{(x,y)}$ yang dinamakan P_0' , P_1' , P_2' . Namun, hanya salah satu pasangan dari tiga pasangan sebelumnya yang akan digunakan saat proses penyisipan selesai. Pasangan tersebut dipilih sebagai *reference point* yang nantinya digunakan untuk mengimbangi dua pasang *pixel* lainnya dan membangun blok 2x2 baru. (Chang et al, 2008:39)

2.6.3 Optimal Selection Rules for the Reference Point

Memilih *reference point* yang berbeda dapat menimbulkan beragam distorsi pada *Stego-Image*. Pada tahap ini, dijelaskan tentang pendekatan aturan

pemilihan yang optimal (*Optimal Selection*) untuk meminimalkan nilai *Mean-Square-Error (MSE)*. Misalkan, jika nilai $m_i = d_i' - d_i$ di mana d_i merupakan nilai selisih pixel sebelum proses penyisipan dan d_i' merupakan nilai selisih pixel yang baru sesudah proses penyisipan. Terdapat lima aturan yang dapat digunakan untuk memilih satu pasangan *optimal reference* tanpa perlu menghitung nilai MSE. Aturan tersebut adalah :

- a. Jika semua nilai m_i lebih besar dari 1 atau lebih kecil dari -1, maka pasangan *pixel* optimalnya adalah yang memiliki nilai $|m_i|$ terbesar. Misal $m = \{-8, -4, -3\}$, pasangan *pixel* optimalnya -8
- b. Jika semua m_i memiliki tanda yang sama dan hanya satu m_i yang bernilai $\{0, 1, -1\}$, maka pasangan *pixel* optimalnya adalah yang memiliki nilai $|m_i|$ terkecil selain 0, 1, -1. Misal $m = \{4, 3, 1\}$, pasangan *pixel* optimalnya 3
- c. Jika hanya satu m_i yang memiliki tanda yang berbeda, maka pasangan *pixel* optimalnya adalah m_i lain yang memiliki nilai $|m_i|$ terkecil. Misal $m = \{3, -5, -3\}$, pasangan *pixel* optimalnya -3
- d. Jika hanya satu m_i yang bernilai $\{0, 1, -1\}$ dan m_i lain berbeda tanda, maka pasangan *pixel* optimalnya adalah 0 atau 1 atau -1. Misal $m = \{0, -4, 2\}$, pasangan *pixel* optimalnya 0
- e. Jika terdapat lebih dari satu nilai m_i yang bernilai $\{0, 1, -1\}$, maka pasangan *pixel* optimalnya adalah 0 atau 1 atau -1. Misal $m = \{4, 0, 0\}$, pasangan *pixel* optimalnya 0 pada index 1 atau 2

Dengan mengikuti lima aturan ini, kita tidak perlu menghitung nilai MSE untuk mendapatkan pasangan *pixel* optimal sebagai *optimal reference pair*. Karena itu, kompleksitas komputasi dapat berkurang banyak. Nilai tersebut digunakan untuk mengimbangi kedua pasangan pixel lainnya. (Chang et al, 2008:39)

2.6.4 Adaptive Rules to Reduce Distortion

Penyisipan bit dalam jumlah banyak dapat dengan mudah menyebabkan distorsi yang cukup besar. Karena, distorsi tersebut diakibatkan oleh proses penyeimbangan nilai pixel (*Offsetting*). Terdapat dua kondisi yang dinamakan “*Branch Condition*” yang didesain untuk menghindari *offset* yang terlalu besar, yaitu :

1. embed_bit ($P_0 \geq 5$) dan embed_bit ($P_1 \geq 4$)
2. embed_bit ($P_0 < 5$) dan embed_bit ($P_2 \geq 6$), di mana embed_bit(P_i) mewakili jumlah bit yang disisipkan ke P_i . Jika kondisi tersebut terpenuhi, gunakan metode PVD untuk memproses P_0 dan P_3 . (Chang et al, 2008:39-40)

2.6.5 Algoritma *Embedding*

Algoritma *Embedding* dari metode *Tri-Way Pixel Value Differencing* (TPVD) adalah sebagai berikut (Chang et al, 2008:40-41) :

1. Hitung selisih *pixel* dari keempat *pixel pairs* yang dapat ditulis sebagai berikut :

$$d_0 = p_{(x+1,y)} - p_{(x,y)}$$

$$d_1 = p_{(x,y+1)} - p_{(x,y)}$$

$$d_2 = p_{(x+1,y+1)} - p_{(x,y)}$$

$$d_3 = p_{(x+1,y+1)} - p_{(x,y+1)}$$
2. Gunakan nilai mutlak dari d_0, d_1, d_2, d_3 untuk menentukan masuk ke kelompok mana *pixel* tersebut berdasarkan *Design Range Table*. Tabelnya adalah sebagai berikut :

Tabel 2.3 – *Design Range Table*

0-3	0	4	2
4-7	4	4	2
8-11	8	4	2
12-15	12	4	2
16-23	16	8	3
24-31	24	8	3
32-47	32	16	4
48-63	48	16	4
64-95	64	32	5
96-127	96	32	5
128-191	128	64	6
192-255	192	64	6

t merupakan banyak bits dari pesan rahasia yang dapat disisipkan ke setiap *pixel pair*. Nilai t pada *Design Range Table* didapat dari rumus $\lfloor 2 \log w \rfloor$.

3. Baca pesan rahasia dalam bentuk biner tersebut sebanyak t_0, t_1, t_2 bit secara berurutan, kemudian ubah ketiga susunan bit tersebut menjadi bilangan desimal, didapatlah nilai b_0, b_1, b_2 .
4. Hitung nilai selisih baru dengan rumus :

$$d'_0 = l_0 + b_0, \text{ jika nilai } d_0 \geq 0 \quad d'_0 = -(l_0 + b_0), \text{ jika nilai } d_0 < 0$$

$$d'_1 = l_1 + b_1, \text{ jika nilai } d_1 \geq 0 \quad d'_1 = -(l_1 + b_1), \text{ jika nilai } d_1 < 0$$

$$d'_2 = l_2 + b_2, \text{ jika nilai } d_2 \geq 0 \quad d'_2 = -(l_2 + b_2), \text{ jika nilai } d_2 < 0$$
5. Ubah nilai p_n dan p_{n+1} dengan rumus :

$$(p'_n, p'_{n+1}) = (p_n - \lceil m/2 \rceil, p_{n+1} + \lfloor m/2 \rfloor)$$

Di mana p_n dan p_{n+1} mewakili dua *pixel* pada P_i dan $m = d'_n - d_n$.
6. Gunakan *selection rules* untuk memilih *optimal reference pair*, kemudian gunakan titik tersebut untuk meng-*offset* kedua pasangan lainnya.
7. Blok baru yang dibangun dari semua pasangan *pixel* yang telah disisipkan pesan rahasia telah terbentuk. Ulangi ke-7 langkah ini hingga semua bit pesan rahasia selesai disisipkan.

2.6.6 Algoritma *Extraction*

Algoritma *Extraction* dari metode *Tri-Way Pixel Value Differencing* (TPVD) adalah sebagai berikut (Chang et al, 2008:41-42) :

1. Hitung selisih *pixel* dari keempat *pixel pairs* yang dapat ditulis sebagai berikut :

$$d'_0 = p_{(x+1,y)} - p_{(x,y)}$$

$$d'_1 = p_{(x,y+1)} - p_{(x,y)}$$

$$d'_2 = p_{(x+1,y+1)} - p_{(x,y)}$$

$$d'_3 = p_{(x+1,y+1)} - p_{(x,y+1)}$$
2. Gunakan nilai mutlak dari d'_0, d'_1, d'_2, d'_3 untuk menentukan masuk ke kelompok mana *pixel* tersebut berdasarkan *Design Range Table*. Tabelnya dapat dilihat pada Tabel 2.3.

3. Cari nilai b'_i dengan cara mengurangkan setiap nilai mutlak dari d'_0 , d'_1 , d'_2 dengan batas bawahnya masing-masing. Jika *Stego-Image* tidak diubah, maka $b'_i = b_i$.
4. Lalu, ubah ketiga bilangan tersebut menjadi bilangan biner dengan panjang bit sebanyak t -nya masing-masing.
5. Ulangi langkah 1-4 sampai semua pesan rahasia yang disisipkan ke dalam *Stego-Image* kembali seperti semula.

Jika proses *embedding* dan *extraction* sukses, maka seharusnya hasil dari proses *extraction* sama dengan pesan yang disembunyikan pada proses *embedding*.

2.7 Mean Square Error (MSE)

Menurut Male et al (2012:4), *Mean Square Error (MSE)* adalah nilai error kuadrat rata-rata antara *Cover-Image* dengan *Stego-Image*. MSE dinyatakan dengan rumus :

$$MSE = \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N [I(x,y) - I'(x,y)]^2$$

Di mana x dan y adalah koordinat *pixel*, M dan N adalah panjang dan lebar gambar dalam satuan *pixel*. $I(x,y)$ menyatakan nilai *pixel Cover-Image* dan $I'(x,y)$ adalah nilai *pixel Stego-Image*.

Untuk gambar warna dengan komponen RGB (*Red, Green, Blue*), nilai MSE secara keseluruhan merupakan jumlah MSE untuk setiap komponen kemudian dibagi tiga.

2.8 Peak Signal to Noise Ratio (PSNR)

Menurut Male et al (2012:4), *Peak Signal to Noise Ratio (PSNR)* adalah perbandingan antara nilai maksimum dari sinyal yang diukur berdasarkan seberapa besar *error* yang berpengaruh pada sinyal tersebut. PSNR biasanya diukur dengan satuan decibel (db). PSNR digunakan untuk mengetahui perbandingan kualitas gambar sebelum dan sesudah proses penyisipan pesan.

Untuk menentukan PSNR, kita harus menghitung nilai MSE terlebih dahulu. Rumusnya adalah :

$$PSNR = 10 \log_{10} \left(\frac{MAX i^2}{MSE} \right)$$

Di mana MAXi adalah nilai maksimum dari *pixel* dalam media yang digunakan dengan contoh sebagai berikut :

$$\max i^2 \begin{cases} 1, double - precision \\ 255, uint8 bit \end{cases}$$

Semakin besar nilai PSNR, maka semakin baik kualitas *Stego-Image*. Begitupun sebaliknya, jika semakin rendah nilai PSNR, maka semakin rendah pula kualitas *Stego-Image*.

Berikut adalah contoh singkat dari perhitungan PSNR :

7	1	1
2	3	4
5	0	6

Citra awal

6	1	1
3	3	3
5	3	6

citra akhir

$$MSE = \frac{(7-6)^2 + (1-1)^2 + (1-1)^2 + (2-3)^2 + (3-3)^2 + (4-3)^2 + (5-5)^2 + (0-3)^2 + (6-6)^2}{3 \times 3}$$

$$MSE = \frac{1+0+0+1+0+1+0+9+0}{9} = 1,33$$

$$PSNR = 20 \log_{10} \left(\frac{7}{\sqrt{1,33}} \right) = 15,665$$

Gambar 2.9 Contoh Soal Peak Signal to Noise Ratio (PSNR)

2.9 Android

Android adalah sistem operasi *mobile* yang berbasis Linux yang sudah dimodifikasi. Android dikembangkan dengan nama perusahaan yang sama, Android, Inc. Pada tahun 2005, sebagai bagian dari strategi untuk memasuki pasar *mobile*, Google membeli Android dan mengambil alih pengembangannya (begitu pula tim pengembangannya).

Google ingin agar Android menjadi terbuka dan bebas, karena itulah sebagian besar *code* Android dirilis di bawah lisensi Apache *open source*, yang artinya siapapun yang ingin menggunakan Android dapat melakukannya dengan cara men-*download* seluruh *source code* Android. Selain itu, vendor (biasanya produsen *hardware*) dapat menambahkan ekstensi milik mereka sendiri untuk Android dan menyesuaikan Android untuk membedakan produk mereka dengan produk yang lain. Model pengembangan yang simpel membuat Android menjadi sangat atraktif dan menarik perhatian banyak vendor.

Keuntungan utama dari mengadopsi Android adalah Android menawarkan pendekatan *Unified* ke pengembangan aplikasi. *Developers* hanya perlu mengembangkan Android, dan aplikasi mereka harus bisa dijalankan dalam bermacam-macam *devices*, selama *devices* tersebut menggunakan Android. Dalam dunia *smartphone*, aplikasi merupakan bagian yang paling penting dari rantai kesuksesan. (Lee, 2012:2)

Tabel 2.4 – Tabel Versi Android

0.9	22 Agustus 2008	
1.0	23 September 2008	<i>Apple Pie</i>
1.1	9 Februari 2009	<i>Banana Bread</i>
1.5	30 April 2009	<i>Cupcake</i>
1.6	15 September 2009	<i>Donut</i>
2.0/2.0.1/2.1	26 October 2009	<i>Eclair</i>
2.2	20 Mei 2010	<i>Froyo</i>
2.3	6 Desember 2010	<i>Gingerbread</i>
3.0/3.1/3.2	22 Februrari 2011	<i>Honeycomb</i>
4.0	18 Oktober 2011	<i>Ice Cream Sandwich</i>
4.1/4.2/4.3	9 Juli 2012	<i>Jellybean</i>
4.4	31 Oktober 2013	<i>Kit Kat</i>
5.0/5.1	17 Oktober 2014	<i>Lollipop</i>
6		<i>M</i>

(<http://socialcompare.com/en/comparison/android-versions-comparison>,

29 Mei 2015)

2.9.1 Android SDK

Android SDK adalah *tools* bagi para *programmer* yang ingin mengembangkan aplikasi berbasis google android. Android SDK mencakup seperangkat alat pengembangan yang komprehensif. Android SDK terdiri dari *debugger*, *libraries*, *handset emulator*, dokumentasi, contoh kode, dan tutorial. (Lee, 2011: 10)

2.9.2 Android Development Tools (ADT)

ADT adalah perangkat tambahan untuk Eclipse yang menunjang dalam pembuatan dan menjalankan atau mencoba aplikasi Android. Dengan menggunakan ADT, kita bisa melakukan hal-hal berikut dalam Eclipse :

- Membuat proyek aplikasi Android.
- Mengubah aplikasi Android menjadi Android *Packages* (APK).
- Membuat tanda tangan digital untuk aplikasi kita.
- Memeriksa dan mencari kesalahan aplikasi Android.

(Lee, 2011: 15)

2.9.3 Java

Bahasa pemrograman Java adalah *general-purpose*, bersifat *concurrent*, *class based*, dan berorientasi objek. Hal ini dirancang sederhana agar *programmer* dapat mudah memahami bahasa Java. Bahasa pemrograman Java terkait dengan C dan C++ namun diatur agak berbeda, dengan sejumlah aspek C dan C++ dihilangkan dan beberapa ide dari bahasa lain dimasukkan. Hal ini dimaksudkan menjadi bahasa produksi, bukan bahasa penelitian, dan sebagainya. (Gosling, et. al. 2013: 1)

2.9.4 Eclipse

Eclipse adalah alat untuk membuat *software*. Eclipse mirip dengan Microsoft Visual Studio, tetapi Eclipse bisa didapatkan secara gratis atau biasa disebut dengan *open source*. IBM memulai Eclipse sebagai proyek *closed source*. Namun, setelah semakin berkembang, IBM mengubah Eclipse menjadi *open source*. Perkembangan Eclipse kini dikelola oleh *Eclipse Foundation*, yang merupakan organisasi non-profit.

Eclipse dapat dengan mudah dikembangkan oleh *programmer*. Eclipse dilengkapi dengan dokumentasi yang ekstensif tentang cara untuk melakukan ini. Ini adalah salah satu alasan Eclipse telah mendapatkan popularitas. Eclipse ditulis dalam bahasa Java. Namun, Eclipse dapat digunakan untuk membangun proyek-proyek dalam bahasa apapun. (Turner dan Chae, 2010)