BAB II
DASAR TEORI

2.1 Citra Digital

Ada beberapa definisi citra yang dapat disimpulkan yaitu sebagai berikut :

a. Menurut Pearson (1991)[12], citra adalah representasi dua dimensi dari dunia visual, menyangkut bebrbagai disiplin ilmu yang mencakup seni, *human vision*, astronomi, teknik dan sebagainya yang diartikan sebagai suatu kumpulan piksel – piksel atau titik – titik yang berwarna yang berbentuk dua dimensi.

b. Menurut Wolfarm Research, Inc. (2002)[12], citra adalah sinyal diskrit 2 dimensi, sinyal ini dapat direpresentasikan sebagai fungsi dari variabel – variabel 2 dimensi, setiap elemen dari arry tersebut disebut piksel.

c. Menurut Michael C. Fairhust[12], citra merupakan fungsi dua dimensi, yaitu *f(x,y)*, dimana *x* dan *y* merupakan koordinat spasial dan besaran atau nilai dari *f* dari setiap koordinat *(x,y)* yaitu intensitas atau tingkat keabuan atau kecemerlangan dari citra pada koordinat tersebut.

Dalam tinjauan matematis, citra merupakan fungsi kontinu dari intensitas cahaya pada bidang dua dimensi. Ketika sumber cahaya menerangi objek, objek memantulkan kembali sebagian cahaya tersebut. Pantulan ini ditangkap oleh alat – alat pengindera optik, misalnya mata manusia, kamera, *scanner* dan sebagainya. Bayangan objek tersebut akan terekam sesuai intensitas pantulan cahaya. Ketika alat optik yang merekam pantulan cahaya itu merupakan mesin digital, misalnya kamera digital, maka citra yang dihasilkan merupakan citra digital. Pada citra digital, kontinuitas intensitas cahaya dikuantisasi sesuai resolusi alat perekam.

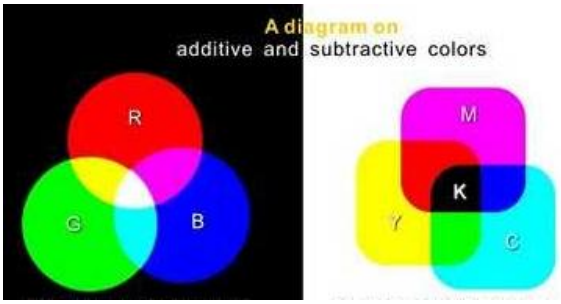
Di dalam komputer, citra digital disimpan sebagai suatu file dengan format tertentu. Format citra tersebut menunjukan cara sebuah citra digital disimpan, misalnya apakah dengan suatu kompresi atau tidak. Contoh format citra digital adalah *.bmp, *.jpg, *.png, *.tif dan sebagainya. Ukuran citra digital dinyatakan dalam *pixel (picture element)*. Umumnya, nilai setiap *pixel* merupakan kuantisasi harga intensitas cahaya. Dengan demikian, suatu citra digital dapat dipandang

sebagai sebuah matriks yang elemen – elemennya menunjukkan intensitas cahaya terkuantisasi. Bedanya terletak pada urutan penyebutan angka ukuran tersebut. Citra digital dengan ukuran 92x112 *pixel* sebenarnya merupakan sebuah matriks dengan ukuran 112x92, dimana 112 merupakan banyaknya baris dan 92 merupakan banyaknya kolom.



Gambar 2.1.Citra digital.[5]

Pada gambar 2.1 di atas merupakan salah satu contoh dari dunia nyata yang diinterpretasikan ke dalam bentuk digital. Kombinasi warna yang terdapat didalamya merupakan gabungan dari warna primer, sehingga dapat membetuk warna yang kompleks. Untuk membuat warna yang kompleks atau degradasi biasanya menggunakan sistem piksel bukan vektor, karena piksel mampu membuat gambar lebih detil daripada vektor. Berikut gambar 2.2 akan menjelaskan kombinasi dari warna *RGB* dan *CMYK*.



(a) (b)

Gambar 2.2.(a) Kombinasi RGB dan (b) Kombinasi CMYK.[11]

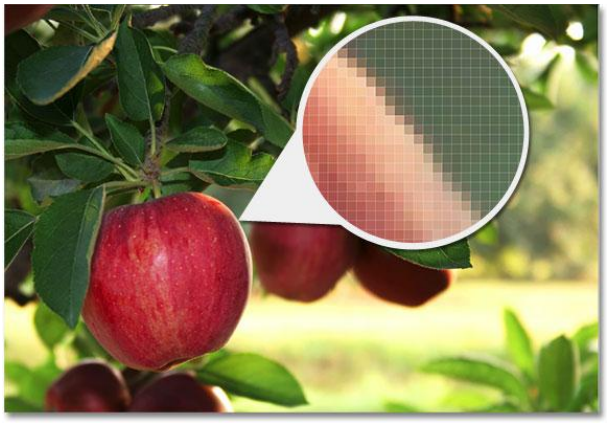
Perbedaan yang mendasar dari kombinasi warna RGB dan *CMYK* adalah cara melakukan kombinasi untuk mendapatkan warnanya. Jika pada standar *RGB* yang harus dilakukan adalah menambahkan warna primer untuk mendapatkan

warna sekunder, sedangkan pada standar *CMYK* kita memang menambahkan juga warna primernya, namun kuncinya ada pada warna hitam yang harus dikurangi kadarnya, karena kalau tidak dikurangi maka warna tersebut akan tetap menjadi hitam.

2.1.1 Piksel

Piksel adalah unsur gambar atau representasi sebuah titik terkecil dalam sebuah gambar grafis yang dihitung per inci atau biasa disebut dpi *(dot per inch)*. Gambar analog dibagi menjadi N baris dan M kolom sehingga menjadi gambar diskrit. Persilangan antara baris dan kolom tertentu disebut dengan piksel. Contohnya adalah gambar atau titik diskrit pada baris N dan kolom M ditulis dengan piksel *[n,m]*. Dilihat dari segi yang membentuk sebuah gambar digital dapat terbagi atas 2 bagian yaitu raster dan vektor. Raster atau bitmap merupakan kumpulan kotak – kotak kecil (piksel). Titik – titik piksel tersebut ditempatkan pada lokasi – lokasi tertentu dengan nilai – nilai warna tersendiri yang secara keseluruhanakan membentuk sebuah tampilan, sedangkan *vector* merupakan gambar digital yang berbasiskan garis dalam membentuk sebuah gambar digital dengan persamaan perhitungan matematis.

Setiap piksel memiliki alamat koordinat matriks tersendiri. Piksel sendiri berasal dari akronim bahasa Inggris yaitu *Picture Element* yang disingkat menjadi *pixel*.



Gambar 2.3. Piksel dari sebuah gambar digital.[6]

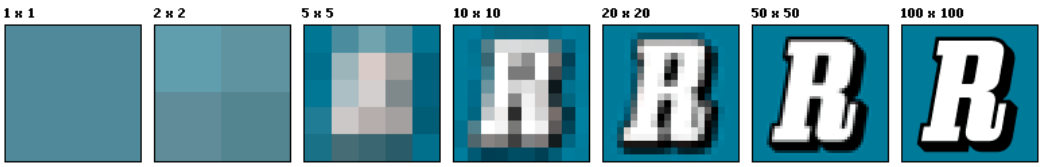
Dari gambar 2.3 di atas terlihat piksel dari pinggiran buah apel yang telah di *zoom* sehingga menjadi bukti bahwa penyusun sebuah gambar digital adalah kombinasi titik atau kotak yang disusun rapat dan teratur sehingga membentuk pola warna dan objek tertentu.

2.1.2 Resolusi

Dalam gambar digital ada yang disebut dengan resolusi gambar yang mendeskripsikan tentang banyaknya detail gambar yang tersimpan. Resolusi gambar bisa juga digunakan untuk mendefinisikan tentang gambar digital, video, maupun yang lainnya. Resolusi dapat dinyatakan dalam berbagai macam ukuran, seperti baris per milimeter dan garis per tinggi gambar, atau yang lebih dikenal hanya dengan garis.

Resolusi sering digunakan sebagai jumlah piksel dalam pencitraan gambar digital. Sebuah gambar dengan tinggi sejumlah N piksel dan lebar M piksel, dapat memiliki resolusi garis yang kurang dari itu. Namun, jika jumlah piksel digunakan sebagai pengukur resolusi, metode yang digunakan adalah mengambil dua buah bilangan bulat yang menunjukkan berapa piksel tinggi gambar tersebut dan berapa piksel lebarnya, kemudian mengalikan angka ini, dan membaginya dengan satu juta untuk mendapatkan angka *megapixel*.

Dibawah ini adalah ilustrasi untuk menjelaskan bagaimana jumlah pixel mempengaruhi kualitas gambar dengan ukuran gambar digital yang sama.



Gambar 2.4. Contoh perbedaan resolusi dari gambar digital.[7]

Sebuah gambar digital sangat erat hubunganya dengan resolusi, karena sebuah gambar digital dibangun oleh sekumpulan titik – titik yang disebut

resolusi. Semakin banyak resolusi suatu gambar digital maka akan terlihat semakin halus *(smooth)* gambar tersebut, sebaliknya jika resolusi suatu gambar digital semakin sedikit, maka akan terlihat buram *(blur)*. Pada gambar 2.4 di atas kita dapat melihat perbedaan kualitas dari sebuah gambar digital pada saat resolusi yang berbeda dengan sumber *sampling* gambar digital yang sama.

2.2 Sabotase Gambar Digital

Sabotase gambar digital adalah proses pengubahan gambar digital, baik gambar yang berasal dari foto digital atau gambar digital bentuk lainnya. Manipulasi ini biasanya dilakukan dengan sebuah program komputer disebut editor gambar untuk mengubah dan memperindah gambar. Program lainnya yang dapat mengubah photo digital adalah editor grafik vektor atau editor grafik bitmap. Biasanya gambar disimpan dalam komputer dalam bentuk piksel. Dimana piksel ini terdapat informasi tentang warna dan keterangan gambar. Editor gambar dapat mengubah piksel ini untuk merubah gambar dalam banyak cara. Piksel – piksel ini dapat diubah dalam grup atau satuan, oleh algoritma rumit dalam editor gambar.

Forensik (berasal dari bahasa Latin *forensis* yang berarti "dari luar", dan serumpun dengan kata forum yang berarti "tempat umum") adalah bidang ilmu pengetahuan yang digunakan untuk membantu proses penegakan keadilan melalui proses penerapan ilmu atau sains. Dalam kelompok ilmu – ilmu forensik ini dikenal antara lain ilmu fisika forensik, ilmu kimia forensik, ilmu psikologi forensik, ilmu kedokteran forensik, ilmu toksikologi forensik, ilmu psikiatri forensik, komputer forensik, dan sebagainya.

Komputer forensik yang juga dikenal dengan nama digital forensik, adalah salah satu cabang ilmu forensik yang berkaitan dengan bukti legal yang ditemui pada komputer dan media penyimpanan digital.

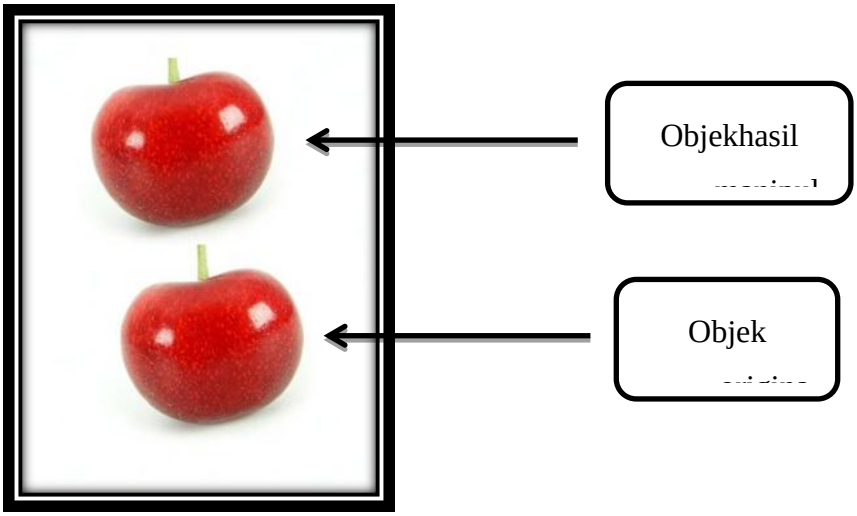
2.3 Pengolahan Gambar Digital

Gambar digital sebagai salah satu komponen multimedia memegang peranan penting dalam informasi visual. Gambar digital mempunyai karakteristik

yang tidak dimiliki oleh data teks, yaitu gambar digital kaya dengan informasi. Walaupun gambar digital kaya akan informasi, namun terkadang gambar digital yang kita miliki tidak sesuai dengan apa yang kita harapkan, misal gambar tersebut kurang terang *(contrast)*, kabur *(blur)*, dan sebagainya. Umumnya pengolahan gambar digital dilakukan untuk :

1. Meningkatkan kualitas penampakan atau untuk menonjolkan beberapa aspek informasi yang terkandung di dalam gambar digital.
2. Elemen di dalam gambar perlu dikelompokkan, dicocokkan, atau diukur.
3. Sebagian gambar digital perlu digabung dengan bagian gambar yang lain atau gambar itu sendiri.

Gambar digital sangat mudah untuk dimanipulasi karena ketersediaan editor perangkat lunak yang banyak serta canggih, sehingga sulit untuk membedakan gambar digital tersebut telah disabotase atau belum. Salah satu teknik yang digunakan dalam sabotase gambar digital adalah *copy-move.* Teknik ini bekerja dengan cara menutupi atau menambahkan sebuah objek, dimana objek tersebut diambil dari sebagian gambar digital itu sendiri dengan tujuan merubah nilai informasi yang terkandung didalam sebuah gambar digital.



Gambar 2.5.Contoh gambar digital yang telah disabotase

Pada gambar 2.5 diatas terlihat bahwa objek buah ceri telah digandakan menjadi dua, yang asalnya buah ceri di dalam gambar digital tersebut hanya satu. Proses penggandaan objek didalam gambar digital cukup mudah untuk dilakukan

karena ketersediaan *software editor* yang cukup canggih serta cukup sulitnya kita dalam membedakan dengan mata telanjang jika gambar digital tersebut telah dimanipulasi.

2.4 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*object-oriented programming)* disingkat OOP merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas – kelas atau objek – objek. Bandingkan dengan logika pemrograman terstruktur, dimana setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya. Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.[9]

Class adalah kumpulan atas definisi data dan fungsi – fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *'class of vehicle'* adalah suatu unit yang terdiri atas definisi – definisi data dan fungsi – fungsi yang menunjuk pada berbagai macam karakteristik dari kendaraan. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang non-programmer sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah *class* sebaiknya relatif bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek – aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

Objek adalah pembungkus data dan fungsi menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah pemrograman komputer berorientasi objek.

Abstraksi dalam OOP merupakan cara bagaimana melihat suatu sistem dalam bentuk yang lebih sederhana. Sama halnya dalam berusaha, kita harus melihat keberhasilan dalam bentuk yang lebih sederhana. Tidak mungkin kita dapat langsung menuju keberhasilan tanpa proses – proses atau capaian – capaian yang lebih sederhana. Seperti seorang kuli bangunan yang sedang berusaha membuat sebuah rumah. Dia pasti akan berusaha dalam bentuk yang lebih sederhana tanpa harus langsung berfikir untuk membuat rumah langsung jadi. Orang tersebut pasti akan berusaha menyelesaikan fondasinya, jika sudah berhasil maka dilanjutkan dengan dindingnya, dilanjutkan dengan atap, hingga *finishing*. Semua dilakukan bertahap, dan dimana setiap tahapannya kita pasti memiliki capaian – capaian yang dimana jika keseluruhannya telah tercapai pasti akan menghasilkan keberhasilan.

Enkapsulasia adalah menyembunyikan proses dalam sistem. Seperti proses autentifikasi *username* dan *password* saat login disebuah *website*. Kita sama sekali tidak tau bagaimana cara *username* dan *password* yang kita kirim ke *server* lalu di autentifikasi. Yang kita tahu hanya pada saat kita login dengan *username* dan *password* yang tepat, maka kita dapat login ke *account* kita. Sifat ini kerap terjadi saat kita berusaha untuk mencapai keberhasilan yang dimana keberhasilan tersebut akan dilihat banyak orang. Contoh usaha yang dilakukan oleh seorang pembalap guna menjadi pembalap nomor satu, secara garis besar dapat dikatakan sebagai berikut :

1. Apakah kita tahu usaha yang telah dilakukan pembalap tersebut?.
2. Berapa banyak putaran *lap* yang dia tempuh dalam sehari.
3. Berapa banyak waktu yang dia habiskan untuk latihan fisik.
4. Berapa banyak dia mengurangi jam liburannya untuk berlatih.

Kita tidak pernah mengetahui secara *detail* usaha pembalap tersebut, hanya hasil yang dia dapat sebagai pembalap papan atas yang kita tahu.

Polimorfisme (polymorphism) erat kaitannya dengan Pewarisan. *Polimorfisme* adalah pemikiran bahwa objek dinamis suatu kelas dasar dapat berperilaku seperti kelas turunan. Ketika objek tersebut menunjuk kelas dasar, objek tersebut berperilaku seperti kelas dasar, tetapi ketika objek tersebut

menunjuk kelas turunan, objek tersebut berperilaku seperti kelas turunan. Dalam hal ini objek dapat memiliki beberapa bentuk, tergantung pada saat itu kelas mana yang ditunjuk. Yang perlu menjadi catatan, bahwa perubahan perilaku ini dari kelas dasar kepada kelas turunan, tidak dapat objek kelas turunan menunjuk kelas dasar.[9]

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

2.5 Bahasa Pemrograman Standar (UML)

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. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka UML lebih cocok untuk penulisan piranti lunak dalam bahasa bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk *modelling* aplikasi prosedural dalam VB atau C.

Seperti bahasa – bahasa lainnya, UML mendefinisikan notasi dan syntax. 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)*.

Sejarah UML sendiri cukup panjang, sampai era tahun 1990 puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Masa itu terkenal dengan masa perang metodologi *(method war)* dalam pendesainan berorientasi objek. Masing – masing metodologi membawa notasi sendiri – sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group atau perusahaan lain yang menggunakan metodologi yang berlainan.

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikatan metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 di-*release* draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 2.3 yang dirilis bulan Mei 2010.Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.

Berikut ini langkah – langkah 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 tes 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, Pendekatan *use case* , dengan meng-*assign* setiap *use case* kepada tim pengembang tertentu untuk mengembangkan *unit code* yang lengkap dengan tes, dan melakukan pendekatan komponen, yaitu meng-*assign* setiap komponen kepada tim pengembang tertentu.

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, bisa langsung dipahami dengan mudah apabila kita melihat tabel 2.1 dibawah. *Main concepts* bisa kita pandang sebagai *term* yang akan muncul pada saat kita membuat diagram, dan *view* adalah kategori dari diagaram tersebut. Untuk menguasai UML ada dua hal yang harus kita perhatikan, yaitu menguasai pembuatan UML serta langkah – langkah dalam analisa dan pengembangan dengan UML. Konsep dasar UML bisa dirangkumkan dalam tabel dibawah :

Tabel 2.1. Konsep dasar UML.[10]

Major Area	***Views***	***Diagrams***	***Main Concepts***

Structural	*static view*	*class diagram*	*class, association, generalization, dependency, realization, interface*
	use case view	*use case diagram*	*use case, actor, association, extend, include, use case generalization*
	Implementation view	*component diagram*	*component, interface, dependency, realization*
dynamic	*state machine view*	*statechart diagram*	*state, event, transition, action*
	actifity view	*activity diagram*	*state, activity, completion transition, fork, join*
	interaction view	*sequence diagram*	*interaction, object, message, activation*
		colaborating diagram	*collaborating, interaction, collaboration rule, message*
Model management	*model management view*	*class diagram*	*package, subsystem, model*

extensibility	*all*	*all*	*constraint, stereotype, tagged values*
Major Area	***Views***	***Diagrams***	***Main Concepts***
Structural	*static view*	*class diagram*	*class, association, generalization, dependency, realization, interface*
	use case view	*use case diagram*	*use case, actor, association, extend, include, use case generalization*
	Implementation view	*component diagram*	*component, interface, dependency, realization*
dynamic	*state machine view*	*statechart diagram*	*state, event, transition, action*
	actifity view	*activity diagram*	*state, activity, completion transition, fork, join*
	interaction view	*sequence diagram*	*interaction, object, message, activation*
		colaborating diagram	*collaborating, interaction,*

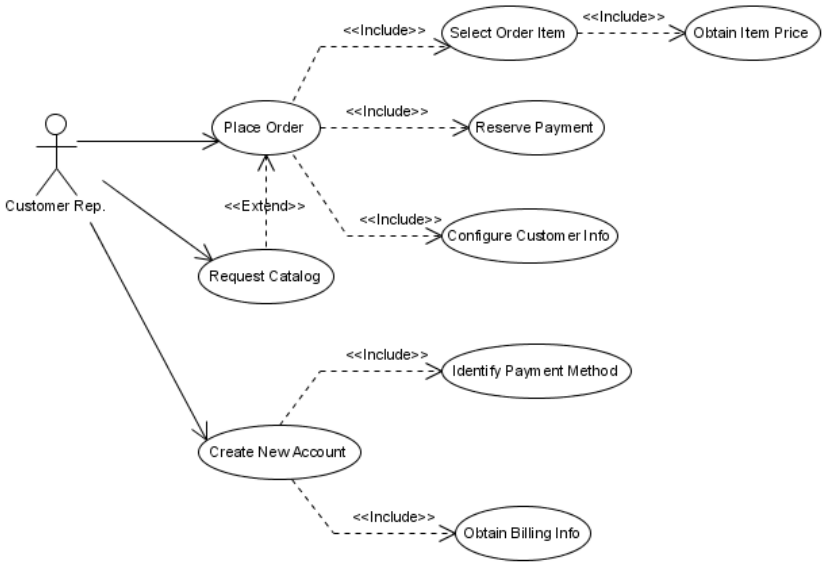
			collaboration rule, message
Model management	*model management view*	*class diagram*	*package, subsystem, model*
extensibility	*all*	*all*	*constraint, stereotype, tagged values*

Seperti juga tercantum pada tabel 2.1 diatas, UML mendefinisikan diagram – diagram sebagai berikut:

1. *Use case diagram.*
2. *Class diagram.*
3. *Statechart diagram.*
4. *Activity diagram.*
5. *Sequence diagram.*
6. *Collaboration diagram.*
7. *Component diagram.*
8. *Deployment diagram.*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem yang menekankan pada "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 atau 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 *client*, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Berikut contoh dari use case diagram :



Gambar 2.6.Contoh diagram *use case* seorang *customer*

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.

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut atau properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda atau fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti c*ontainment*, 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 :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak – anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja.

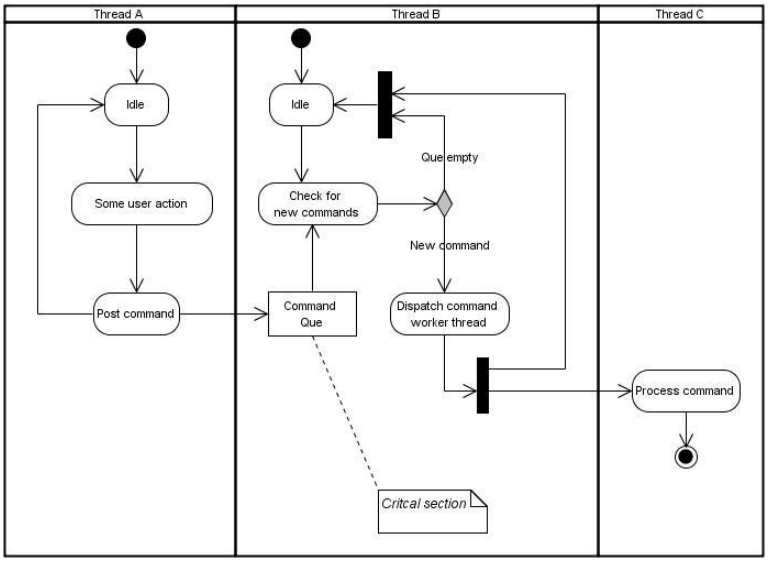
Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. Sesuai dengan perkembangan *class model*, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*. Antar *class* memiliki beberapa keterhubungan, diantaranya :

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain.
2. Agregasi, yaitu hubungan yang menyatakan bagian (“terdiri atas..”).
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan *(message)* yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence*.

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.

Activity diagram 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* pada saat *state* sebelumnya selesai *(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. Berikut contoh diagram *activity* sebuah proses :

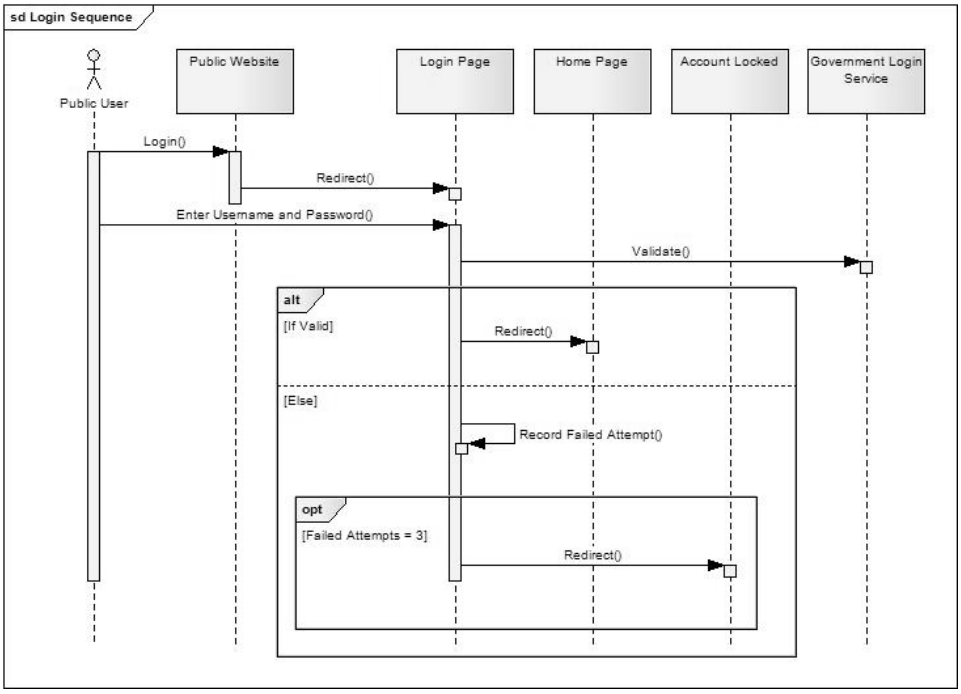


Gambar 2.7.Contoh diagram *activity*

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.

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.



Gambar 2.8.Contoh diagram *sequence login*

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 atau 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*.

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.

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.

Deployment atau 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.

2.6 Antarmuka dan *Core* Program

Python adalah bahasa pemrograman interpretatif multi guna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. *Python* diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. *Python* mendukung multi paradigma pemrograman, namun tidak dibatasi pada pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Salah satu fitur yang tersedia pada *python* adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan

manajemen memori otomatis. Seperti halnya pada bahasa pemrograman dinamis lainnya, *pyhton* umumnya digunakan sebagai bahasa skrip meski pada prakteknya penggunaan bahasa ini lebih luas mencakup konteks pemanfaatan yang umumnya tidak dilakungan dengan menggunakan bahasa skrip. *Python* dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai *platform* sistem operasi.

Python dikembangkan oleh Guido van Rossum pada tahun 1990 di *CWI*, *Amsterdam*. Versi terakhir yang dikeluarkan *CWI* adalah 1.2. Sisi utama yang membedakan *python* dengan bahasa lain adalah dalam hal aturan penulisan kodeprogram. *Python* memiliki kelebihan tersendiri dibandingkan dengan bahasa lain terutama dalam hal penanganan modul, ini yang membuat beberapa programmer menyukai *python*. Selain itu *python* merupakan salah satu produk yang *opensource*, *free*, dan *multiplatform*.

Fitur *python* yang lain beberapa diantaranya adalah :

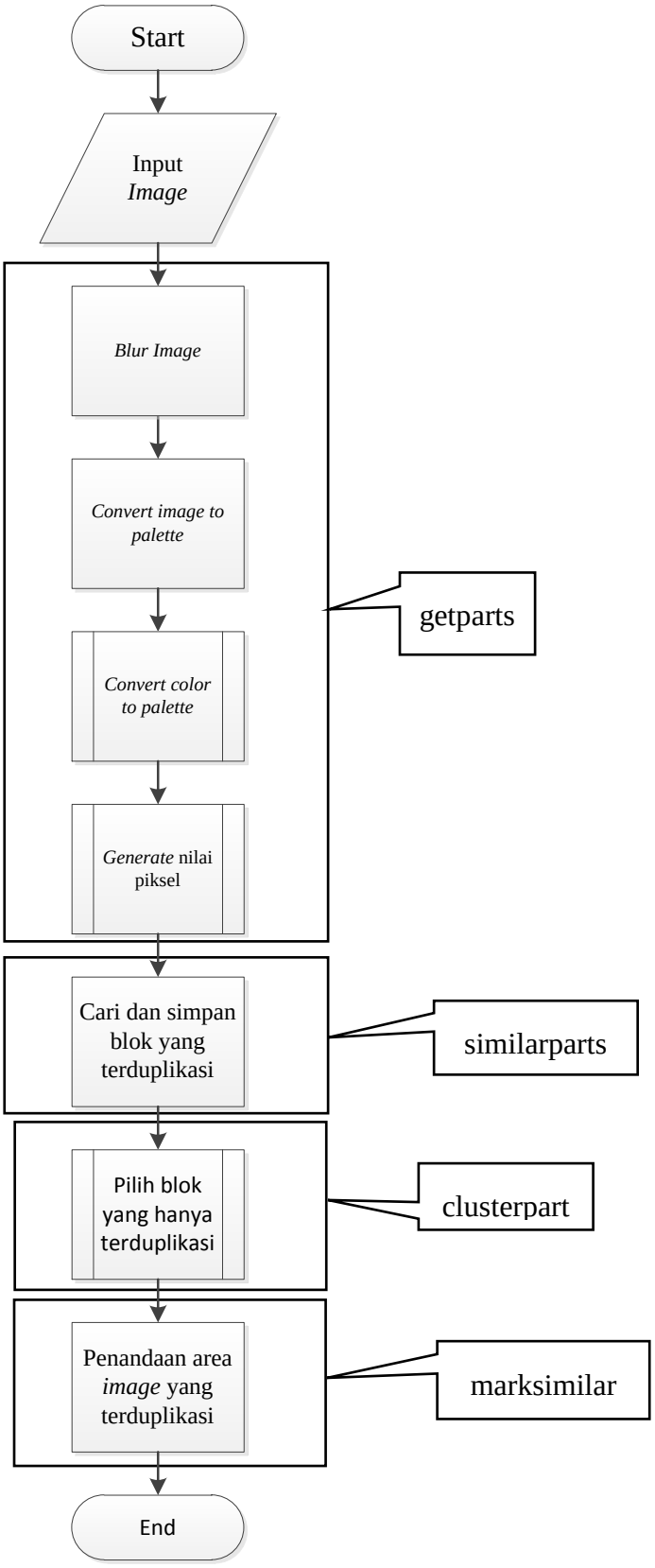
1. Memiliki kepustakaan yang luas, dalam distribusi *python* telah disediakan modul – modul siap pakai.
2. Untuk berbagai keperluan.
3. Mmemiliki tata bahasa yang jernih dan mudah dipelajari.
4. Memiliki aturan layout kode sumber yang memudahkan pengecekan, pembacaan kembali danpenulisan ulang kode sumber.
5. Berorientasi obyek.
6. Bemiliki sistem pengelolaan memori otomatis (*garbage collection*, seperti java)
7. Modular, mudah dikembangkan dengan menciptakan modul – modul baru.
8. Modul – modul yang terdapat dalam *python* dapat dibangun dengan bahasa *python* sendiri maupun C/C++.
9. Memiliki fasilitas pengumpulan “sampah” otomatis, seperti halnya pada bahasa pemrograman java.
10. *Python* memiliki fasilitas pengaturan penggunaan ingatan komputer sehingga para pemrogram tidak perlu melakukan pengaturan ingatan komputer secara langsung.

Secara spesifik dalam pengerjaan *software* tugas akhir ini menggunakan *python* versi 2.6 yang *release* pada tanggal 01 Oktober 2008, dengan tambahan *library,* yaitu P*ython Image Library (PIL)* 1.1.7 yang *release* pada tanggal 15 November 2009. Sedangkan *interface* atau sering disebut *GUI (Graphical User Interface)* yang nanti digunakan untuk menerima input dari *user*, yang juga menjadi parameter skrip *python* akan dibuat berbasiskan web dengan bahasa pemrograman HTML, untuk itu dalam menggunakan program ini harus menggunakan sebuah *web browser*.

2.7 Algoritma *Robust*

Algoritma *robust* menuntut pengguna menginisialisasi nilai segmen yang akan dijadikan blok piksel kotak NxN. Pemecahan segmen gambar digital ini dilakukan dimulai dari kiri atas ke kanan atas dan berakhir pada kanan bawah, dengan kata lain pemecahan segmen ini dilakukan per baris. Setiap NxN blok piksel akan dipilih per kolom pertama dan akan disimpan ke dalam array 2 dua dimensi. Setalah tersimpan di dalam array, blok piksel tersebut diubah ke dalam gambar serta warna palet berdasarkan nilai piksel. Jika sudah terurut berdasarakan nilai piksel maka akan lebih untuk menemukan blok piksel yang identik. Berikut

flowchart dari algoritma robust :



Gambar 2.9. *Flowchart* Algoritma *Robust*

Langkah *flowchart* dimulai dengan *user* menginputkan sebuah gambar digital yang akan diproses, proses pertama kali yang dilakukan pada gambar digital ini adalah proses *blurring,* proses *blurring* ini ditujukan untuk menghilangkan detil gambar. Setelah itu gambar digital tersebut dipecah menjadi palet, setelah berbentuk palet proses berikutnya ambil nilai dari setiap piksel untuk berikutnnya *disorting* berdasarkan nilai piksel, jika telah *tersorting* berdasarkan nilai piksel akan lebih mudah untuk menemukan blok piksel yang identik, setelah mendapatkan blok piksel yang identik maka buang blok piksel yang tidak memiliki kembaran yang identik, setelah mendapatkan kelompok piksel yang memiliki kembaran, tandai lokasi dimana blok piksel tersebut pada gambar digital.

Suatu algortima dapat dikatakan sebagai algoritma *robust* dikarenakan algoritma tersebut mampu bertahan atau kuat terhadap sesuatu. Dalam hal mendeteksi adanya objek yang terduplikasi, algoritma *robust* harus mampu mendapatkan adanya objek yang terduplikasi walaupun gambar digital yang akan diproses berformat *.jpg, karena gambar digital yang berformat *.jpg dalam proses penyimpanannya menggunakan metode *lossy compression.*

Lossy compression adalah suatu metode untuk mengkompresi data dan men-dekompresinya, data yang diperoleh mungkin berbeda dari yang aslinya tetapi cukup dekat perbedaaanya. *Lossy* kompresi ini paling sering digunakan untuk kompres data multimedia (audio, gambar digital). Sebaliknya, *lossles compression* diperlukan untuk data teks dan file, seperti catatan bank, artikel teks dll.

Format *lossy compression* mengalami *generation loss* yaitu jika melakukan berulang kali kompresi dan dekompresi file akan menyebabkan kehilangan kualitas secara progresif. Hal ini berbeda dengan kompresi data *lossless* yang tidak kehilangan kualitasnya. ketika pengguna yang menerima file terkompresi secara *lossy*, file yang diambil dapat sedikit berbeda dari yang asli dilevel bit namun tidak dapat dibedakan oleh mata manusia.

Inilah sebabnya algoritma ini dikatakan algoritma yang kuat, karena algoritma ini mampu memproses data yang menggunakan metode *Lossy compression.*Pada dasarnya algoritma ini mengacu pada algoritma yang bernama *“ad-hoc”*.