# BAB III

# OBJEK DAN METODE PENELITIAN

## 3.1 Objek Penelitian

Penelitian dilakukan di Dinas Pengelolaan Sumber Daya Air Bandung pada bagian kepegawaian. Penelitian ini dilakukan untuk untuk merancang sistem informasi kepegawaian. Sistem informasi yang dirancang meliputi pengolahan data pegawai, proses kenaikan pangkat, proses kenaikan gaji berkala, proses mutasi dan proses pensiun.

### 3.1.1 Sejarah berdirinya Dinas Pengelolaan Sumber Daya Air Bandung

Saat ini nama Dinas PU Pengairan diganti menjadi Dinas Pengelolaan Sumber Daya Air dengan Peraturan Daerah No.15 tahun 2000, untuk mengetahui sejarah singkat tentang berdirinya PU Pengairan Propinsi Jawa Barat, kita harus mengetahui dasar-dasar dan landasan hukum terbentuknya dinas PU pengairan sejak jaman penjajahan sampai sekarang. Perjuangan Dinas PU telah mengalami 3 (tiga) jaman, yaitu :

a. Jaman Penjajahan Belanda

Pada jaman penjajahan belanda yaitu pada tahun 1883 dibentuk brigade irigasi *(irrigate brigade)* di bawah pimpinan Ir. Heskes. Setelah itu tahun 1889 dibentuk pula bagian irigasi (*Agdelling Irigatie*) dalam Departemen Der Burgeliyke Openbare Warken (BOW)

Setelah pemerintahan Hindia Belanda mendirikan Departemen BOW kemudian terjadi perubahan menjadi Departemen V En W atau *Departement*

*Verkeer En Waterstaat Ban West Java* yang berkedudukan di Bandung. Dalam V En W ini tergabung didalamnya jawatan pengairan , PTT (Pos Telegraf dan Telepon) dan jawatan Lalu Lintas Jalan Raya. Khusus tugas di bidang pengairan diatur dalam :

1. *Algemaene Waterelegment* tahun 1963 (sub 1963 No.489)
2. *Algemaene Waterbeheerverordening* (sub 1957 No.503)
3. *Provincial Waterglement* 1940 No.7

b. Jaman Penjajahan Jepang

Pada jaman penjajahan Jepang, Dinas PU Pengairan bernama "Doboko Jimuso" yang dibentuk serta pembagian susunannya sama seperti jaman *V En W*. Setelah Jepang kalah dan Indonesia memproklamasikan kemerdekaannya, bentuk dan susunan *doboko jimuso* masih dipakai akan tetapi personilnya yang dijabat oleh orang-orang Jepang yang diambil alih dengan paksa dan diganti dengan tenaga kerja Indonesia. Setelah keadaan semakin buruk, maka dalam mempertahankan kemerdekaan bangsa, *V En W* terutama para pemudanya tidak ketinggalan untuk mempertahankan "Gedung Sate" di Bandung sebagai kantor pusat *V En W*.

c. Jaman Setelah Kemerdekaan RI

Pada tanggal 13 Desember 1945 di kota Bandung terjadi puncak perjuangan fisik bangsa Indonesia dalam rangka mempertahankan kemerdekaan yang telah di proklamasikan dari tangan Penjajahan Belanda dan Jepang serta mengalami puncak perlawanan yag tidak sedikit menumpahkan darah. Pada pertempuran melawan tentara sekutu dalam rangka mempertahankan Gedung pertama dari

Dinas Pekerjaan Umum telah gugur 7 (tujuh) orang pemuda warga jawatan Pekerjaan Umum untuk mempertahankan gedung tersebut yaitu "*V En W*", sekarang dikenal dengan sebutan "Gedung Sate". Oleh karena itu hingga sekarang setiap tanggal 3 Desember dijadikan hari Kebangkitan Pekerjaan Umum.

Setelah Negara Republik Indonesia terbentuk dan pembubaran Negara Bagian dengan Undang-Undang Nomor:22/1945 tentang Otonomi Daerah termasuk pembentukan Propinsi Jawa Barat tahun 1953 dengan PP No.181/1953 dilaksanakan penyerahan tugas-tugas Pekerjaan Umum pada Pemerintah Daerah. Struktur Organisasi Pekerjaan Umum pada Pemerintah Daerah Tingkat I Jawa Barat. Setelah dikeluarkannya Undang-Undang No.22/1945 dan Dekrit Presiden 5 Juli 1959, Jawatan ini diberi nama Dinas Pekerjaan Umum Propinsi Daerah Tingkat I Jawa Barat.

Kantor pusat Dinas Pekerjaan Umum sebelum perang kemerdekaan bertempat di Gedung Sate Bandung, kemudian Dinas Pekerjaan Umum dibentuk menjadi 3 (tiga) Dinas yaitu :

1. Dinas Pekerjaan Umum Pengairan Propinsi Daerah Tingkat I Jawa Barat.
2. Dinas Pekerjaan Umum Bina Marga Propinsi Daerah Tingkat I Jawa Barat.
3. Dinas Pekerjaan Umum Cipta Karya Propinsi Daerah Tingkat I Jawa Barat.

Setelah mengalami perubahan-perubahan yang sifatnya Struktural, Dinas PU Pengairan Daerah Tingkat I Jawa Barat dibentuk berdasarkan peraturan No.4 tahun 1988 tentang pembentukan Organisasi dan tata kerja Dinas PU Pengairan

Daerah Tingkat I Jawa Barat di Jl. Braga No.137 Bandung dan berdasarkan Peraturan Daerah No.15/2000 Dinas PU Pengairan namanya dirubah lagi menjadi Dinas Pengelolaan Sumber Daya Air Propinsi Jawa Barat.

### 3.1.2 Visi dan Misi

#### 3.1.2.1 Visi

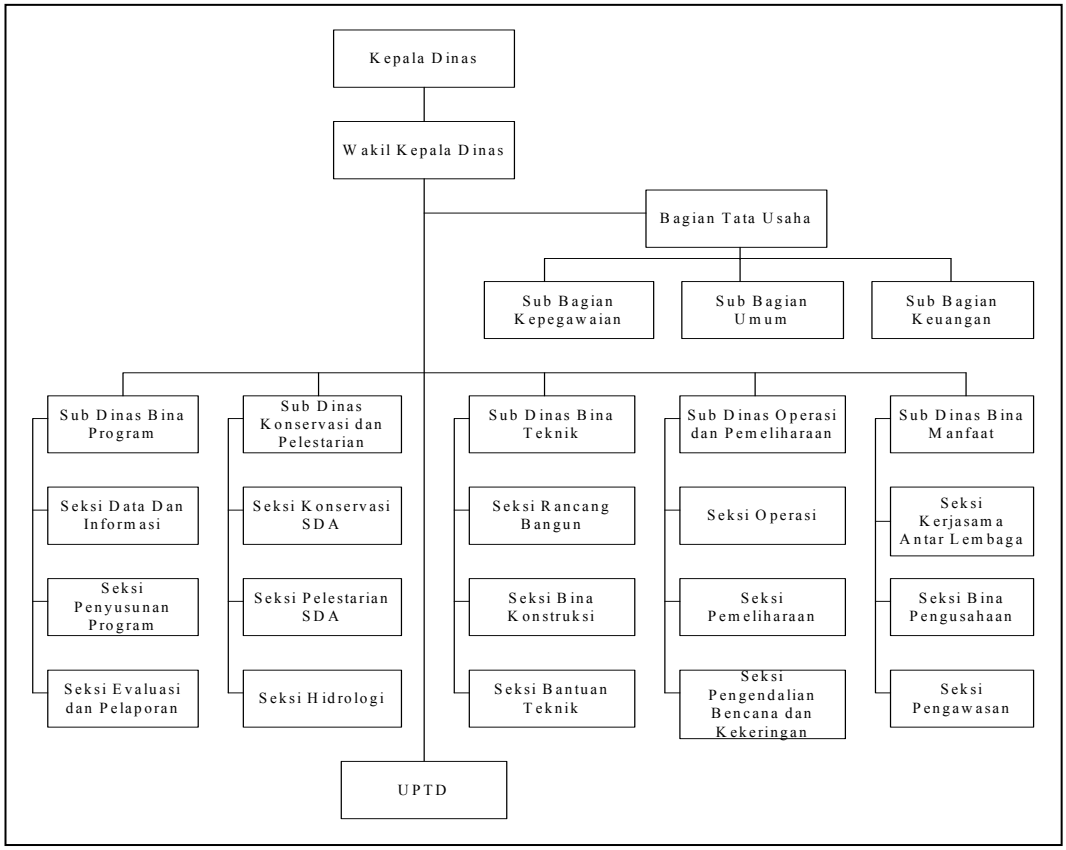
**"**Dinas sebagai pengelolaan sumber daya air yang andal, berkeadilan dan berkesinambungan tahun 2010".

#### 3.1.2.2 Misi

Misi dari Dinas Pengelolaan Sumber Daya air Bandung adalah sebagai berikut:

1. Mengembangkan sistem pengelolaan sumber daya air secara terpadu, mandiri dan berkelanjutan.
2. Meningkatkan konservasi, pendayagunaan sumber daya air dan pengendalian daya rusak air.
3. Menciptakan pelayanan secara optimal, efektip dan efisien untuk kesejahteraan masyarakat.
4. Memenuhi semua kebutuhan air dengan tepat waktu, ruang, jumlah dan mutu.

### 3.1.3 Struktur Organisasi Perusahaan




Sumber : Dinas PSDA Bandung

**Gambar 3.1**

Struktur Organisasi Dinas Pengelolaan Sumber Daya Air Propinsi Jawa Barat

### 3.1.4 Uraian Tugas

Berdasarkan struktur organisasi diatas, uraian tugas pada Dinas Pengelolaan Sumber Daya Air Bandung adalah sebagai berikut :

1. Kepala Dinas
   a. Memimpin, mengatur, membina dan mengendalikan pelaksanaan tugas pokok dan fungsi Dinas.
   b. Menetapkan kebijakan operasional dinas sesuai dengan kebijakan umum Pemerintah Propinsi Jawa Barat.

c. Menetapkan rencana kerja dan program pembangunan di bidang pengelolaan sumber daya air.
d. Menyelenggarakan fasilitas yang berkaitan dengan penyelenggaraan program, pengembangan dan pelestarian, teknik, operasi dan pemeliharaan, manfaat serta ketatausahaan.
e. Melaksanakan pengendalian operasional terhadap pelaksanaan tugas dekonsentrasi dan tugas pembantuan.
f. Melaksanakan tugas kedinasan lain sesuai dengan pelimpahan dari Gubernur.

2. Wakil Kepala Dinas
   a. Mengkoordinasikan kegiatan internal dinas.
   b. Melaksanakan pembinaan administrasi kegiatan Dinas dan UPTD yang meliputi pembinaan administrasi kepegawaian, keuangan, perlengkapan, bagian umum dan kesisteman.
   c. Mengkoordinasikan perumusan Perencanaan Strategis (RENSTRA) dan Laporan Akuntabilitas Kinerja Instansi Pemerintahan (LAKIP) Dinas.
   d. Melaksanakan tugas operasional lain sesuai dengan pelimpahan Kepala Dinas.
3. Bagian Tata Usaha

Bagian Tata Usaha mempunyai tugas pokok menyelenggarakan pengelolaan kepegawaian, keuangan dan umum.

1. Sub Bagian Kepegawaian
    a. Melaksanakan pengumpulan, pengelolaan, penyimpanan dan pemeliharaan data dan kartu kepegawaian dilingkungan dinas.
    b. Melaksanakan penyiapan rencana kebutuhan pegawai, formasi dan penunjukan dalam jabatan dilingkungan dinas.
    c. Melaksanakan penyiapan dan pengusulan pegawai yang akan pensiun, peninjauan masa kerja serta pemberian penghargaan.
    d. Melaksanakan penyiapan bahan kenaikan pangkat, DP3, DUK, sumpah/janji pegawai, gaji berkala dan peningkatan kesejahteraan pegawai.
    e. Melaksanakan penyiapan bahan mutasi dan pemberhentian pegawai.
    f. Melaksanakan penyiapan pegawai untuk mengikuti pendidikan/pelatihan kepemimpinan teknis dan fungsional.
    g. Melaksanakan penyiapan rencana pegawai yang akan mengikuti ujian dinas dan ijin/tugas belajar.
    h. Melaksanakan penyiapan bahan pembinaan kepegawaian dan disiplin pegawai.
    i. Melaksanakan penyiapan bahan standar kompetensi pegawai tenaga teknis dan fungsional.
    j. Melaksanakan penyiapan bahan pembinaan kelembagaan dan ketatalaksanaan dilingkungan dinas.
    k. Melaksanakan penyiapan bahan rancangan dan pendokumentasian peraturan perundang-undangan.

l. Melaksanakan penyusunan bahan evaluasi dan laporan kegiatan SubBagian Kepegawaian.

m. Melaksanakan koordinasi dengan unit kerja terkait.

2. Sub Bagian Keuangan

a. Melaksanakan pengumpulan bahan dan penyiapan rencana anggaran pendapatan dan belanja rutin serta pembangunan.

b. Melaksanakan pengadministrasian dan pembukuan keuangan anggaran belanja rutin dan pembangunan.

c. Melaksanakan penyusunan pembuatan daftar gaji dan tunjangan daerah serta pembayaran.

d. Melakasanakan pembendaharaan keuangan anggaran belanja rutin dan pembangunan.

e. Melaksanakan penyiapan bahan pembinaan administrasi dan pembukuan keuangan anggaran pendapatan / penerimaan dinas.

f. Melaksanakan penyiapan bahan pertanggung jawaban anggaran pendapatan, anggaran belanja rutin dan pembangunan.

g. Melaksanakan penyusunan bahan evaluasi dan laporan kegiatan Sub Bagian Keuangan.

h. Melaksanakan koordinasi denga unit kerja terkait.

3. Sub Bagian Umum

a. Melaksanakan penerimaan, pendistribusian dan pengiriman surat-surat, naskah dinas dan pengelolaan kearsipan.

b. Melaksanakan pengadaan naskah dinas.

c. Melaksanakan pengelolaan dan penyiapan bahan pembinaan kearsipan kepada unit kerja dilingkungan dinas.
d. Melaksanakan penyiapan dan pengendalian administrasi perjalanan dinas pegawai.
e. Melaksanakan pengelolaan perpustakaan, hubungan masyarakat dan pendokumentasian kegiatan dinas.
f. Melaksanakan pengurusan rumah tangga dinas dan ketertiban serta keamanan kantor.
g. Melaksanakan pemeliharaan dan perawatan dilingkungan kantor, kendaraan dinas, perlengkapan kantor dan aset lainnya.
h. Melaksanakan penyiapan rencana kebutuhan pengadaan sarana dan prasarana perlengkapan dilingkungan dinas.
i. Melaksanakan pengurusan pengadaan, penyimpanan, pendistribusian inventarisasi dan penghapusan perlengkapan dinas.
j. Melaksanakan penyiapan bahan pengelolaan administrasi perlengkapan dan perbekalan.
k. Melaksanakan penyusunan evaluasi dan laporan kegiatan Sub Bagian Umum.
l. Melaksanakan koordinasi dengan unit kerja terkait.

4. Sub Dinas Bina Program
    a. Menyelenggarakan penyusunan rencana strategis dan Laporan Akuntabilitas Kinerja Instansi Pemerintah (LAKIP) di bidang pengelolaan sumber daya air.
    b. Menyelenggarakan koordinasi penyusunan program kerja dinas.
    c. Menyelenggarakan pengumpulan data, pengolahan dan analisis data.
    d. Menyelenggarakan pengelolaan statistik mengenai kegiatan dinas.
    e. Menyelenggarakan pengelolaan sistem inormasi manajemen bidang pengelolaan sumber daya air.
    f. Menyelenggarakan koordinasi penyusunan anggaran pembangunan.
    g. Menyelenggarakan evaluasi dan laporan pelaksanaan program kerja dinas.
    h. Menyelenggarakan koordinasi dengan unit terkait.
5. Sub Dinas Konservasi dan Pelestarian
    a. Menyelenggarakan perumusan bahan dan rencana pembinaan teknis pelaksanaan konstruksi prasarana konservasi sumber daya air, konstruksi prasarana pelestarian sumber daya air dan pengelolaan hidrologi.
    b. Menyelenggarakan perumusan bahan pembinaan operasional dan pengelolaan hidrologi.
    c. Menyelenggaraka perumusan bahan pemantauan, evaluasi dan pelaporan kegiatan pelaksanaan konstruksi prasarana, konservasi sumber daya air, konstruksi prasarana pelestarian sumber daya air dan hidrologi.
    d. Melaksanakan koordinasi dengan unit kerja terkait.

6. Sub Dinas Bina Teknik
    a. Menyelenggarakan pengumpulan dan pengolahan data di bidang rancang bangun, bina konstruksi dan bantuan teknis.
    b. Menyelenggarakan perumusan bahan pembinaan teknis di bidang rancang bangun, bina konstruksi dan bantuan teknis.
    c. Menyelenggarakan perumusan bahan pembinaan bantuan teknik.
    d. Menyelenggarakan perumusan bahan pemantauan, evaluasi dan pelaporan pelaksanaan kegiatan bina teknik.
    e. Menyelenggarakan koordinasi dengan unit kerja terkait.
7. Sub Dinas Operasi dan Pemeliharaan
    a. Menyelenggarakan perumusan bahan pembinaan operasional pengelolaan kegiatan operasi prasarana dan sarana sumber daya air.
    b. Menyelenggarakan perumusan bahan pembinaan teknis pelaksanaan dan pengendalian bencana banjir dan kekeringan.
    c. Menyelenggarakan perumusan kebutuhan dan perumusan bahan pengendalian bencana banjir dan kekeringan.
8. Sub Dinas Bina Manfaat
    a. Menyelenggarakan pengumpulan dan pengelolaan data kegiatan kerjasama antar lembaga bina pengusaha dan pengendalian pemanfaatan sumber daya air.
    b. Menyelenggarakan perumusan bahan pembinaan operasional kegiatan kerjasama antar lembaga, pengusahaan dan pengendalian pemanfaatan sumber daya air.

c. Menyelenggarakan pengaturan pembinaan kerjasama antar lembaga, pengusahaan dan pengendalian pemanfaatan sumber daya air.
d. Menyelenggarakan perumusan bahan pemantauan, evaluasi dan pelaporan pelaksanaan kegiatan bidang bina manfaat.
e. Menyelenggarakan koordinasi dengan unit kerja terkait.

## 3.2 Metode Penelitian

Metode penelitian yang akan dilakukan dalam penelitian ini adalah metode yang menghasilkan produk juga melibatkan individu sebagai objek penelitian.

### 3.2.1 Desain Penelitian

Desain penelitian ini termasuk kedalam penelitian yang bersifat deskriptif. Penelitian deskriptif adalah penelitian yang bertujuan untuk memperoleh ciri-ciri variabel, dimana dalam penelitian ini adalah untuk memperoleh gambaran tentang kinerja program dirancang dan diimplementasikan kepada pengguna (*user*) pada Dinas Pengelolaan Sumber Daya Air Bandung.

### 3.2.2 Sumber dan Metode Pengumpulan Data

Sumber dan Metode pengumpulan data yang dilakukan oleh penulis dibagi menjadi dua bagian, yaitu :

#### 3.2.2.1 Sumber Data Primer

1. Observasi

   Observasi adalah penelitian yang dilakukan dengan cara pengamatan langsung terhadap objek yang diteliti, hal ini dilakukan untuk mengetahui

sistem pengolahan data pegawai pada bagian kepegawaian di Dinas Pengelolaan Sumber Daya Air Bandung.

2. Interview/Wawancara

   Penulis menyiapkan daftar pertanyaan dan mengadakan tanya jawab langsung yang berhubungan dengan sistem pengolahan data pegawai kepada orang yang berkompeten dan berwenang, dalam hal ini pegawai yang ditunjuk oleh Kepala Dinas untuk memberikan informasi kepada peneliti.

#### 3.2.2.2 Sumber Data Sekunder

Sumber data sekunder yang mendukung dalam penelitian ini adalah dokumentasi yang ada pada Dinas Pengelolaan Sumber Daya Air Bandung baik dari buku-buku pedoman kerja ataupun file-file yang ada.

### 3.2.3 Metode Pendekatan dan Pengembangan Sistem

#### 3.2.3.1 Metode Pendekatan Sistem Berorientasi Objek

Menurut **[Adi05]** ada beberapa karakterisitik yang menjadi ciri-ciri dari pendekatan berorientasi objek adalah :

1. Pendekatan lebih pada data dan bukannya pada prosedur/fungsi.
2. Program besar dibagi pada apa yang dinamakan objek-objek.
3. Sturktur data dirancang dan menjadi karakteristik dari objek-objek.
4. Fungsi-fungsi yang mengoperasikan data tergabung dalam suatu objek yang sama.
5. Data tersembunyi dan terlidung dari prosedur/fungsi yang ada di luar.

6. Objek-objek dapat saling berkomunikasi dengan saling mengirim *message* (pesan) satu sama lain.
7. Pendekatan adalah dari bawah ke atas (*bottom up approach*).

**[Lar04]** mendefinisikan analisis adalah proses mendekomposisi atau menginvestigasi *domain problem* atau *requirements* menjadi bagian-bagian yang bisa dimengerti. Proses analisis lebih menekankan kepada pertanyaan seperti apakah sistem yang akan dikerjakan, bagaimana sistem tersebut digunakan sehari-hari oleh user.

Perancangan adalah melakukan sintesa model *software* dari masalah yang di analisis, menterjemahkan *domain* model kepada *software* model sehingga bisa dimengerti dan dikomunikasikan kepada anggota tim lain dan juga para profesional *software* lainnya. Bisa disebut juga analisis adalah menentukan apa yang seharusnya dikerjakan, sedangkan perancangan adalah menentukan cara terbaik untuk mengerjakan sesuatu dengan benar setelah diketahui apa yang harus dikerjakan.

Terlihat bahwa analisis dilakukan sebelum perancangan. Analisis berorientasi objek menekankan kepada menemukan dan menjelaskan objek-objek dan interelasinya yang menjadi bagian atau pemain dalam sebuah sistem.

Perancangan berorientasi objek memodelkan objek-objek yang di identifikasi pada proses analisis dan interaksi antar objek ke dalam model *software*. Proses selanjutnya bisa mengidentifikasi atribut dan operasi objek-objek Lainnya.

### 3.2.3.2 Metode Pengembangan Sistem

#### 3.2.3.2.1 *Agile*

Metode pengembangan sistem yang digunakan adalah penggabungan *Agile* dan *Unified Process* (UP). Pada salah satu kamus bahasa Inggris, *agile* diterjemahkan sebagai "pergerakan yang cepat dan ringan" yang bersinonim dengan kata "*nimble*". Dengan demikian metode yang "*agile*" bisa didefinisikan sebagai metode yang berkarakter "cepat dan ringan".

**[Lar04]** mengemukakan selain itu metode yang *agile* juga mengutamakan kesederhanaan, komunikasi yang intensif antar anggota tim, organisasi tim yang fleksible dan menyesuaikan dengan kebutuhan. Metode *agile* biasanya menerapkan iterasi yang dibatasi oleh waktu dan menghasilkan produk yang dihasilkan secara inkremental. Ciri lain dari metode *agile* adalah adaptasi dan respons yang cepat terhadap adanya perubahan.

Secara spesifik, praktek dari metode *agile* ini bermacam-macam di antaranya adalah *eXtreme Programming* (XP) yang dikonsep oleh *Kent Beck* yang cukup populer dan fenomenal di awal kemunculannya. Selain XP, metode lain yang berkarakter *agile* adalah *Scrum*, *Lean Development*, *DSDM*, *FeatureDriven Development*, dan tentu saja *Unified Process* (UP).

Manifesto *Agile*:

1. Mengutamakan orang-orang dan interaksi (daripada proses dan tools)
2. Mengutamakan software yang berjalan (daripada dokumentasi yang lengkap)
3. Mengutamakan kerjasama pelanggan (daripada negosiasi kontrak)

4. Mengutamakan respons terhadap perubahan (daripada mengikuti rencana)

Prinsip *Agile*:

1. Prioritas utama adalah memuaskan pelanggan dengan menghasilkan produk dengan cepat dan berkala.
2. Menerima dengan baik *requirement* yang berubah, walaupun pengerjaan software sudah masuk ke fase pengembangan. Prinsip *agile* mengutamakan kepentingan kompetitif pelanggan yang sekali waktu memerlukan perubahan *requirement* tersebut.
3. Menghasilkan produk berkala dengan cepat, dalam hitungan beberapa minggu sampai beberapa bulan, di mana berlaku prinsip lebih cepat lebih baik.
4. Bisnis analis dan pengembang software harus bekerja sama setiap hari selama proyek berlangsung.
5. Menjalankan proyek dengan orang-orang yang bermotivasi tinggi. Manajemen percaya, mendukung mereka, dan menjaga motivasi tinggi mereka dengan memberikan fasilitas yang mereka perlukan.
6. Mengutamakan mekanisme komunikasi secara langsung antar anggota tim melalui pertemuan-pertemuan yang diadakan berkala dalam sebuah ruangan dengan alatalat yang mendukung seperti papan *whiteboard* yang lebar.
7. Ukuran utama keberhasilan adalah aplikasi yang bisa berjalan.
8. Proses *agile* mendukung proses pengembangan software yang terprediksi, dan memiliki kesempatan keberhasilan yang tinggi (*sustainable*).

9. Manajemen, sponsor, pengembang, dan pelanggan harus bisa memelihara semangat yang konstan selama proyek berlangsung.
10. Perhatian secara terus menerus terhadap kemampuan teknis dan perancangan yang baik akan menambah karakter *agility*.
11. Kesederhanaan didefinisikan ebagai seni untuk memperkecil jumlah pekerjaan adalah sangat penting.
12. Arsitektur terbaik, analisis, dan perancangan berasal dari tim yang terorganisasi dengan sendirinya.
13. Dalam interval tertentu, tim selalu berorientasi untuk menjadi lebih baik, dan mengubah langkahlangkahnya untuk penyesuaian.

**3.2.3.2.2 *Unified Process* (UP)**

*Unified Process* (UP) juga biasa disebut RUP. Menurut **[Fow05]** RUP merupakan corak keseluruhan dan terminologi yang terdapat pada produk resmi *Rational Software* yang berdasarkan UP. UP pada dasarnya merupakan proses interatif dan inkremental.

Pendapat dari **[Lar04]** UP mendefinisikan keseluruhan proses, tahapan produk yang dihasilkan dalam setiap tahapan pembuatan *software*. Tahapan-tahapan ini biasanya terdiri atas: *requirement*, analisis, perancangan, implementasi, pengujian, dan penerapan. Menurut **[Lar04]** UP memiliki ciri-ciri utama sebagai berikut:

1. UP tergolong ke dalam metode yang berulang (iteratif). Metode yang berulang berarti membagi keseluruhan tahap dalam sebuah pembuatan software ke dalam beberapa tahap yang identik dan relatif singkat, di mana

proses dan hasil dari tahap berikutnya bergantung kepada hasil *review* dari tahap sebelumnya dan *feedback* dari pihakpihak yang berkepentingan.

2. UP mendefinisikan struktur untuk menjelaskan bagaimana tahapantahapan dalam keseluruhan proses dilakukan.
3. UP tergolong metode yang fleksibel yang cocok diterapkan terhadap pendekatan *agile* (ringan dan cepat). Pendekatan *agile* saat ini sedang popular karena kecepatan dan fleksibilitasnya.

Pendekatan iteratif dari metode UP ini kontras dengan pendekatan metode *waterfall* yang mencoba untuk mendefinisikan tahapan-tahapan pembuatan *software* ke dalam satu iterasi di mana tahapan *requirement*, analisis dan perancangan diselesaikan seluruhnya diawal sebelum tahapan implementasi dan pengujian dilakukan. Sebaliknya UP melihat keseluruhan proses sebagai iterasi dari sub proses yang terdiri atas tahapan yang lengkap di mana analisis, perancangan, dan implementasi dilakukan dalam setiap sub proses yang memiliki batas waktu tertentu. Hubungan satu sub proses dengan sub proses sesudahnya bersifat *evolutionary*. Artinya sub proses lanjutan dikembangkan berdasarkan hasil dari sub proses sebelumnya berdasarkan *feedback* dan *review* dari pihak-pihak yang berkepentingan. Karena itu metode pengembangan sistem model *waterfall* tidak sesuai dengan filosofi UP [**Fow05**].

Untuk menerapkan metode UP dapat menggunakan prinsip-prinsip *agile* *Agile* adalah sebuah pendekatan yang mengutamakan kecepatan dan keringanan. Ringan dalam arti tidak banyak hal yang dilakukan dan tidak banyak dokumentasi yang dihasilkan. UP memang sekilas tidak tergolong dalam metode yang cepat

dan ringan. Aktivitas yang didefinisikan dalam metode UP ini tergolong cukup banyak dengan dokumentasi yang juga tidak sedikit. Tetapi masih tetap bisa menerapkan pendekatan *agile* ini dalam metode UP. Pertama jangan anggap semua aktifitas dan dokumentasi dalam UP sebagai hal yang wajib dilakukan. Ambil sebagian saja yang memang perlu karena memiliki nilai tambah apabila dilakukan. UP juga tergolong metode yang iteratif dan inkremental. Analisis dan perancangan tidak sepenuhnya dikerjakan di awal sebelum pemrograman dimulai. Pendekatan *agile* juga memiliki ciri demikian.

Pemodelan UML dengan pendekatan *agile* yang lebih mementingkan pemodelan untuk kepentingan komunikasi daripada untuk dokumentasi. Pendekatan *agile* juga lebih banyak menggunakan UML sebagai sketsa dibandingkan sebagai *blueprint* maupun bahasa pemrograman.

Terdapat empat fase besar yang mencirikan metode UP. Keempatnya dijelaskan secara singkat sebagai berikut **[Lar04]**:

1. Insepsi (*Inception*)

   Fase *inception* lebih menitik beratkan kepada pembahasan visi bisnis, penentuan batasan-batasan sistem, dan estimasi-estimasi secara kasar. Pada fase ini kegiatan yang dilakukan lebih bersifat menentukan arah dan batasan-batasan tentang apa yang akan dilakukan, termasuk juga memperkirakan secara kasar besaran-besaran *project* seperti sumber daya dan waktu yang diperlukan.

2. Elaborasi (*Elaboration*)

   Pada fase *elaboration* mulai dibentuk arsitektur utama yang membentuk sistem secara iteratif. Identifikasi resiko tinggi juga dilakukan pada fase ini.

Identifikasi dari *requirement* inti dan memiliki resiko tinggi sudah dilakukan dalam fase ini. Implementasi (yang menghasilkan program yang bisa dijalankan) arsitektur dari aplikasi juga sudah mulai dilakukan. Perkiraan-perkiraan besaran yang didefinisikan dalam fase insepsi dibuat lebih realistis.

3. Konstruksi (*Construction*)

   Fase *construction* adalah fase pengembangan yang melengkapi sistem secara keseluruhan. Pada fase ini dilakukan implementasi secara iteratif terhadap semua *requirement* yang ada.

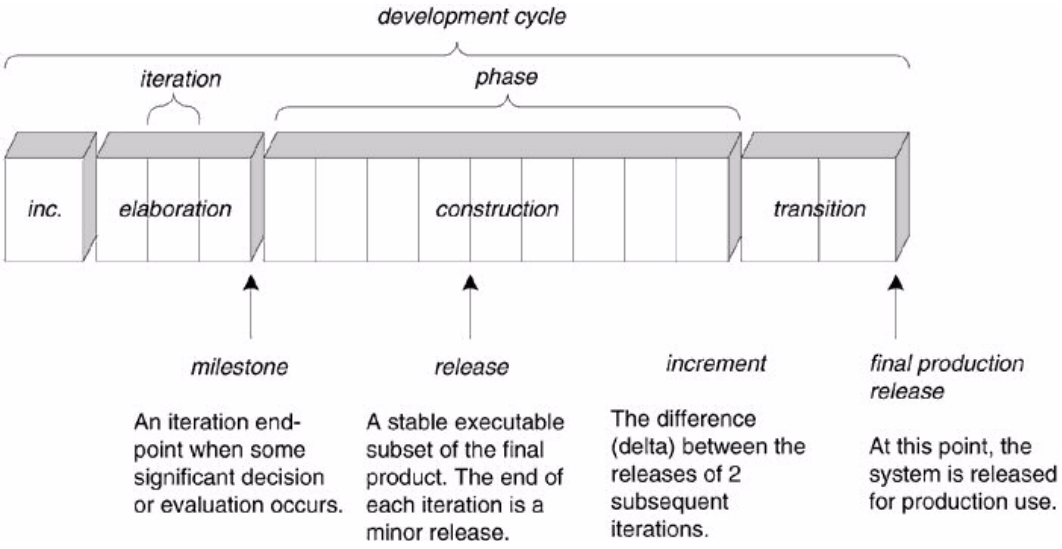
4. Transisi (*Transition*)

   Pada fase *transition*, pengujian sudah dilakukan sampai diserahkannya produk jadi ke pelanggan. *Beta test* dan persiapan penerapan aplikasi dilakukan dalam fase ini.

Walaupun urutan-urutan fase-fase UP mirip *waterfall*, secara karakteristik keduanya tentu saja berbeda. Karakter utama *waterfall* terdapat dalam iterasi tunggal di mana *requirement*, analisis, perancangan, konstruksi, dan pengujian software dilakukan dengan sekuensial. Sebaliknya, UP tergolong metode yang iteratif.

Fase *inception* UP tidak sama dengan tahap *requirement waterfall*. *Inception* lebih menekankan kepada kegiatan-kegiatan yang lebih mendukung *feasibilitas* dari sebuah projek. Fase *elaboration* juga tidak sama dengan fase analisis dan perancangan pada *waterfall*. Dalam fase ini, arsitektur inti dari aplikasi didefinisikan, dianalisis, dirancang, dibuat dan diuji. Aktifitas-aktifitas tersebut dilakukan secara iteratif dan inkremental dengan batasan waktu yang

jelas dalam satu iterasinya. Demikian juga dengan fase *construction* dan *transition*.




Sumber : Craig Larman. Applying UML and Patterns Edition 2004

**Gambar 3.2** Proses Iterasi Tiap-Tiap UP

Pendapat dari **[Lar04]** masing-masing fase dalam UP dapat dipecah ke dalam beberapa *iteration* / pengulangan. Suatu iterasi adalah suatu pengembangan mengulang yang akan menghasilkan rilis dari produk yang dapat di eksekusi, suatu *subset* dari produk akhir di bawah pengembangan, yang bertumbuh secara inkremental dari iterasi ke iterasi untuk menghasilkan sistem akhir.

Menurut [**Lar04**] beberapa manfaat pendekatan iteratif dalam rekayasa perangkat lunak dibandingkan dengan proses *waterfall* / air terjun yang tradisional, proses iteratif mempunyai keuntungan-keuntungan sebagai berikut:

1. Resiko-resiko sebelumnya yang dapat dikurangi
2. Perubahan lebih dapat dikendalikan
3. Tingkat penggunaan kembali yang lebih tinggi dan menguntungkan
4. Tim dalam proyek dapat belajar dalam perjalannya

5. Secara menyeluruh mutu lebih baik

### 3.2.3.3 Alat Bantu Pemodelan

Pemodelan adalah sebuah aktivitas yang telah berlangsung bertahun-tahun dalam pengembangan perangkat lunak. Sebuah model masih diperlukan untuk menulis sebuah aplikasi baik menggunakan bahasa pemograman yang paling sederhana sampai bahasa yang kompleks. Pemodelan dapat berupa menggambar diagram alir (*flowchart*) yang menunjukkan langkah-langkah yang dikerjakan sebuah aplikasi. Pemodelan diperlukan untuk mendefinisikan sebuah model membuat mudah untuk memecah aplikasi yang komplek atau sebuah sistem yang besar menjadi sederhana, menjadi bagian-bagian diskrit yang mudah untuk dipelajari.

Menurut **[SH02]** pemodelan harus memenuhi syarat berikut :

1. *Readability*

   *Readability* membawa kejelasan dan kemudahan dalam memahami. Memahami sistem adalah langkah pertama baik dalam membangun atau mengembangkan sebuah sistem yang telah ada. Hal ini melibatkan pengetahuan tentang bagaimana sistem dibuat, bagaimana sistem berperilaku dan sebagainya. Memodelkan sebuah sistem membuat sistem mudah dibaca dan yang paling penting, mudah dalam pendokumentasian. Penggambaran sebuah sistem agar mudah dibaca melibatkan pemahaman struktur sistem dan tingkah laku sistem.

2. *Reusability*

   *Reusability* adalah efek dari membuat sebuah sistem mudah dibaca. Bila sebuah sistem telah dimodelkan sedemikian rupa sehingga mudah dipahami, karena

para pengembang cenderung untuk mencari kesamaan atau pengulangan, baik dalam hal fungsionalitas, fitur ataupun struktur.

#### 3.2.3.3.1 *Unified Modeling Language* (UML)

Ada beberapa definisi dari*Unified Modeling Language* (UML) yang pada dasarnya memiliki maksud yang sama, antara lain:

**[DR03]** mendefinisikan:

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

**[SH02]**mendefinisikan:

*"Unified Modeling Language* (UML) adalah sebuah bahasa untuk menentukan, visualisasi, kontruksi, dan mendokumentasikan *artifacts* dari sistem *software*, untuk memodelkan bisnis, dan sistem non *software* lainnya. UML merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan sistem yang besar dan kompleks."

Kemudian **[Bah99]** juga mengemukakan:

*"The unified modeling language is a language for specifying, constructing, visualizing, and documenting the software system and its components. A UML is a graphical language with sets of rules and semantics. The rules and semantics are expressed in English, in a form known as object constraint language (OCL)."*

UML mengintegrasikan konsep dari *Booch*, OMT, OOSE dan juga *ClassRelation* dengan menggabungkan mereka menjadi suatu kesatuan bahasa

modeling yang bisa berguna bagi siapa saja. UML bertujuan untuk menjadi standar bahasa modeling yang mampu untuk memodelkan sistem yang konkuren dan juga terdistribusi. UML sudah menjadi standar industri yang dibuat dibawah pengawasan *Object Management Group* (OMG). OMG ditugaskan untuk memberi informasi mengenai *objectoriented methodology* yang mampu untuk membuat bahasa untuk *modeling software*. Dengan menggunakan UML para pengembang 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 ia lebih cocok untuk penulisan piranti lunak dalam bahasa pemrograman berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk *modeling* aplikasi prosedural dalam VisualBasic atau C.

UML bukan merupakan metode dan juga tidak berusaha untuk menjadi suatu metode, namun demikian ia merupakan kebutuhan untuk *Rational Unified Process* yang telah dibuat oleh *Rational Software Corporation*. Metode *development* lainnya seperti *Dynamic Systems Development Method* juga menggunakan teknik dari UML. UML menyediakan kemampuan untuk menangkap karakteristik sebuah sistem dengan menggunakan notasi-notasi tertentu.

UML menyediakan sederetan notasi sederhana yang mudah dipahami untuk mendokumentasikan sistem berdasarkan prinsip-prinsip perancangan berorientasi objek. UML sudah menjadi *standardized modelling language* yang

terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang sistem dan *software* agar bias menyelesaikan tugas-tugas seperti:

1. Spesifikasi
2. Visualisasi
3. Desain Arsitektur
4. Konstruksi
5. Simulasi dan Testing
6. Dokumentasi
7. Sketsa

UML dikembangkan sebagai ide dasar untuk mempromosikan hubunganan produktifitas antara para pengembang dari *objectoriented ystem*. Menurut **[Lar04]** ada tiga cara dalam memakai UML dalam melakukan pemodelan sistem:

1. UML sebagai sketsa

   UML digambarkan dalam sketsa coretan-coretan dalam kertas atau *whiteboard* secara tidak formal. Biasanya digunakan dalam sesi diskusi tim untuk membahas aspek tertentu dalam tahap analisis dan perancangan.

2. UML sebagai *blueprint* system

   Seperti diagram kelistrikan adalah blueprint dari komponen atau produk yang akan dihasilkan, UML juga bisa menggambarkan blueprint yang identik untuk sebuah sistem software.

3. UML sebagai bahasa pemrograman

UML berfungsi sebagai bahasa pemrograman mencoba melakukan semuanya dengan UML sampai kepada produk jadinya. Analisis dan perancangan dilakukan dengan diagram-diagram yang ada dalam UML, sementara sebuah tool atau generator bisa menghasilkan produk akhir dari diagram-diagram ini.

Saat ini UML paling banyak digunakan dengan cara pertama dan kedua. Khusus dalam metode *agile* (cepat dan ringan), UML digunakan dengan cara pertama.

#### 3.2.3.3.2 Konsep Dasar UML

Teknik dasar dalam membuat model dari sebuah ide atau dunia nyata adalah dengan menggunakan abstraksi. Setiap diagram UML yang digambar memiliki keterkaitan dengan dunia nyata. Abstraksi dikembangkan sebagai ketentuan untuk dipelajari dan sering digunakan. Analogi yang lebih mendekati adalah merupakan kumpulan dari sketsa atau *blueprint* yang menampilkan detil yang cukup dari suatu bangunan untuk memastikan tentang apa sebenarnya bangunan tersebut. Abstraksi model dan diagram juga berguna karena akan menjelaskan lebih rinci detildetil yang dibutuhkan.

#### 3.2.3.3.3 Model-Model Diagram UML

UML terdiri atas banyak elemen-elemen grafis yang digabungkan membentuk diagram. Tujuan representasi elemen-elemen grafis ke dalam diagram adalah untuk menyajikan beragam sudut pandang dari sebuah sistem berdasarkan fungsi masing-masing diagram tersebut. Kumpulan dari beragam sudut pandang inilah yang disebut sebuah model. Pada saat ini UML telah mencapai versi 2.1

memiliki banyak perubahan dari versi sebelumnya seperti diagram kolobarasi diganti dengan diagram komunikasi dan terdapat tiga tambahan diagram lainnya. Diagram di klasifikasikan berdasarkan tiga jenis diagram sebagai berikut.

**Tabel 3.1**

Klasifikasi Jenis Diagram UML

| | |
|---|---|
| *Structural Diagram* | *1. Class Diagram*<br>*2. Composite Structure Diagram*<br>*3. Object Diagram*<br>*4. Component Diagram*<br>*5. Deployment Diagram*<br>*6. Package Diagram* |
| *Behavioral Diagram* | *1. Use Case Diagram*<br>*2. Activity Diagram*<br>*3. State Machine Diagram* |
| *Interaction Diagram* | *1. Sequence Diagram*<br>*2. Communication Diagram*<br>*3. Interaction Overview Diagram*<br>*4. Timing Diagram*<br>(Sumber: Martin Fowler, 2005: 19) |

(Sumber: Martin Fowler, 2005:19)

#### 3.2.3.3.3.1 Diagram *Use Case*

Diagram *use case* adalah deskripsi fungsi dari sebuah sistem dari perspektif/sudut pandang para pengguna sistem. *Use case* mendefinisikan "apa" yang dilakukan oleh sistem dan elemen-elemennya, bukan "bagaimana" sistem dan elemen-elemennya saling berinteraksi. *Use case* bekerja dengan menggunakan "*scenario*", yaitu deskripsi urutan-urutan langkah yang menerangkan apa yang dilakukan penggunan terhadap sistem maupun sebaliknya.

Diagram *use case* mengidentifikasikan fungsionalitas yang dipunyai oleh sistem (*use case*), user yang berinteraksi dengan sistem (*actor*) dan asosiasi/keterhubungan antara user dengan fungsionalitas sistem. Komponen notasi dasar yang dipunyai oleh diagram *use case* adalah *actor*, *usecase*, dan

*association*. Berikut adalah notasi yang terdapat pada diagram *use case*. Diagram *use case* memiliki sebuah model khusus yang terbatas untuk kondisi tertentu yang disebut *stereotype*. Untuk menunjukkan *stereotype* digunakan symbol "<<" diawalnya dan ditutup dengan ">>" diakhirnya. Terdapat dua *stereotype* paling sering digunakan dalam *use case diagram* yaitu <<extend>> dan <<include>>. <<extend>> digunakan untuk menunjukkan bahwa satu *use case* merupakan tambahan fungsional dari *use case* yang lain jika kondisi atau syarat tertentu dipenuhi. Sedangkan <<include>> digunakan untuk menggambarkan bahwa suatu *use case* seluruhnya merupakan fungsionalitas dari *use case* lainnya.

#### 3.2.3.3.3.2 Diagram *Activity*

Diagram *activity* digunakan untuk mendokumentasikan alur kerja pada sebuah sistem, yang dimulai dari pandangan *business level* hingga ke *operational level*. Pada dasarnya, diagram *activity* merupakan variasi dari diagram *state machine*. Diagram *activity* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah diagram *activity* bisa mendukung perilaku *parallel* sedangkan *flowchart* tidak bisa.

#### 3.2.3.3.3.3 Diagram *Sequence*

Diagram *Sequence* mendokumentasikan komunikasi/interaksi antar kelas-kelas. Diagram ini menunjukkan sejumlah objek dan *message* (pesan) yang diletakkan diantara objek-objek didalam *use case*. Perlu diingat bahwa di dalam diagram ini, kelas-kelas dan aktoraktor diletakkan dibagian atas diagram dengan urutan dari kiri ke kanan dengan garis *lifeline* yang diletakkan secara vertical terhadap kelas dan aktor.

#### 3.2.3.3.3.4 Diagram *Class*

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

*Class diagram* menggambarkan struktur dan deskripsi *class, package* dan *object* beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. Sebuah *Class* memiliki tiga area pokok (Sumber : Modul *Workshop* UML Bab 2), yaitu:

1. Nama, merupakan nama dari sebuah kelas.
2. Atribut, merupakan peroperti dari sebuah kelas. Atribut melambangkan batas nilai yang mungkin ada pada objek dari *class*.
3. Operasi, adalah sesuatu yang bisa dilakukan oleh sebuah *class* atau yang dapat dilakukan oleh *class*lain terhadap sebuah *class*.

Atribut dan metoda dapat memiliki salah satu sifat berikut (Sumber : Modul *Workshop* UML Bab 2)

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.
4. *Package*, hanya dapat dipanggil oleh *instance* sebuah *class* pada paket yang sama.

*Class* juga 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 *runtime*.

#### 3.2.3.3.3.5 Relasi Pada Diagram *Class*

Relasi atau *relationship* merupakan keterhubungan antar kelas yang muncul pada saat sebuah kelas berinteraksi dengan kelas-kelas lainnya. Didalam diagram *class*, setiap kelas pasti akan berinteraksi dengan baik satu maupun lebih kelas. Relasi yang muncul pada setiap keterhubungan antar kelas juga akan memiliki atributatribut yang akan lebih menjelaskan definisi dari sebuah relasi yang terjadi. Berikut relasi yang terdapat pada diagram *Class* (Sumber : Modul *Workshop* UML Bab 2).

1. Asosiasi (*Association*)

   Para pemodel menggunakan pemahaman asosiasi adalah pada saat beberapa kelas saling terhubung satu sama lain secara konseptual. Asosiasi juga dapat menjadi lebih kompleks pada saat beberapa class terhubung ke satu class.

2. *Constraint* Pada Asosiasi

   Kadangkala sebuah asosiasi diantara dua *class* harus mengikuti sebuah aturan dan aturan ini bisa diletakkan dalam sebuah *constraint* pada garis asosiasi dan diletakkan dalam kurung kurawal. Bentuk lain dari tipe *constraint* adalah relasi OR yang ditulis dengan {or} dalam garis putusputus yang menghubungkan dua garis asosiasi. Kondisi OR ini menghadapkan kepada keadaan bahwa sebuah kelas terhubung dengan dua kelas tetapi pada saat mendefinisikan relasinya

kelas harus memilih salah satu dari kedua kelas tersebut, sedangkan kondisi untuk memilih keduanya adalah *invalid*.

3. Kelas Asosiasi (*Associations Class*)

   Sebuah asosiasi dapat memiliki atribut dan operasi seperti halnya sebuah *class* Sebuah *association class* sebenarnya diperlukan apabila salah satu dari kelas yang terhubung mempunyai sebuah atau beberapa atribut yang tidak layak dimiliki oleh kelas tersebut, karena secara logis atribut tersebut lebih layak dimiliki oleh asosiasi yang menghubungkan kedua kelas tersebut.

4. Multiplisitas (*Multiplicity*)

   Multiplisitas atau kardinalitas adalah jumlah banyaknya objek sebuah *class* yang berelasi dengan sebuah objek lain pada *class* lain yang berasosiasi dengan *class* tersebut. Untuk menyatakan multiplisitas anda dapat meletakkannya diatas garis asosiasi berdekatan dengan *class* yang sesuai. Ada banyak multiplisitas yang mungkin untuk dipakai. Tabel berikut menjabarkan multiplisitas yang dapat digunakan.

**Tabel 3.2**
Nilai Multiplisitas Potensial

| **Indicator** | **Meaning** |
|---|---|
| 0..1 | Zero on one |
| 1 | One only |
| 0..* | Zero or more |
| 1..* | One or more |
| N | Only n (where n > 1) |
| 0..N | Zero to n (where n > 1) |
| 1..N | One to n (where n > 1) |

(Sumber : Modul *Workshop* UML Bab 2)

#### 3.2.3.3.3.6 *Generalization & Inheritance*

Sebuah *class*( *child class* atau *subclass*) dapat mewarisi atribut-atribut dan operasioperasi dari *class* lainnya (*parent class* atau *super class*) dimana *parent class* bersifat lebih umum daripada *child class*

Generalisasi pada konsep *Object Oriented* digunakan untuk menjelaskan hubungan kesamaan diantara *class* Dengan menggunakan generalisasi bisa dibangun struktur logis yang bisa menampilkan derajat kesamaan atau perbedaan diantara *class class*. Manfaat lain dari struktur hirarkis juga memungkinkan untuk penambahan *subclass* (*child class*) baru tanpa harus merubah struktur yang sudah ada.

*Inheritance* adalah sebuah mekanisme pengimplementasian generalisasi dan spesialisasi. Aturan inheritance dapat secara umum bisa diklasifikasikan sebagai berikut :

1. *Subclass* selalu mewarisi semua sifat dari *superclass*nya.
2. Definisi *subclass* selalu mencakup paling tidak satu detil yang tidak diturunkan dari *superclass*nya. *Inheritance* sangat dekat dengan asosiasinya dengan generalisasi. Generalisasi menjelaskan hubungan logis antar elemenelemen yang mempunyai karakteristik yang sama. Sedangkan *inheritance* menerangkan mekanisme agar bagi pakai (*sharing*) bisa terjadi.

#### 3.2.3.3.3.7 Kelas Ketergantungan (*Dependant Class*)

Pada penggunaan relasi kadangkala satu *class* menggunakan *class* yang lain, hal ini disebut *dependency*. Umumnya penggunaan *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain.

Notasi untuk *dependency* pada UML dapat menggunakan garis putusputus dan tanda panah pada ujungnya.

#### 3.2.3.3.3.8 Agregasi

Didalam UML, ada relasi dengan perlakukan khusus yang disebut dengan "bagian dari (*part of*)" yang menangani antar objekobjek dimana salah satunya adalah bagian dari yang lain. Dengan kata lain sebuah objek terdiri atas objekobjek yang lain. Hal seperti ini di dalam UML disebut agregasi. Sebuah agregasi adalah kasus khusus dari asosiasi. Agregasi disimbolkan dengan jajaran genjang yang diletakkan pada *class* yang mengandung objek. Kadangkala relasi OR harus digunakan pada agregasi. Untuk memodelkan hal tersebut, sebuah constraint – dengan kalimat atau dalam tanda kurung kurawal pada garis putus-putus harus dibuat untuk menghubungkan dua objek.

Komposit (*composite*) adalah sebuah tipe agregasi yang kuat dimana bagian dari objek tergantung penuh/secara keseluruhan terhadap objeknya sehingga bila sebuah objek komposit dibuang maka bagian yang tergantung pada komponen tersebut akan terbuang juga pada saat yang bersamaan. Notasi komposit sama seperti agregasi hanya saja jajaran genjangnya terisi (solid).

#### 3.2.3.3.3.9 *Interface* & Realisasi

*Interface* adalah satu set *operation* yang memberikan spesifikasi beberapa aspek dari perilaku dan operasi disuatu *class* ke *class* yang lain. Contohnya, *keyboard* pada komputer sebenarnya merupakan *interface* yang bisa dipakai ulang karena tombol-tombol *keyboard* sebenarnya berasal dari mesin ketik, hanya saja mungkin ada beberapa operasionalisasi tomboltombol yang berbeda yang sudah

ditransfer ke sistem yang lain. Pemodelan *interface* sama dengan pemodelan *class* hanya saja pada *interface* tidak mempunya atribut dan pada penamaannya perlu ditambahkan *stereotype* <<interface>> atau ditambahkan huruf "I" di depan nama interface. Relasi antara *class* dan *interface* disebut *Realization*. Realisasi dituliskan dengan garis putusputus dengan segitiga yang mengarah ke *interface*.

#### 3.2.3.3.3.10 Diagram *Component*

Komponen perangkat lunak adalah bagian fisik dari sebuah sistem yang menetap di komputer. komponen merupakan implementasi software dari sebuah class. Komponen bisa berupa tabel, *file* data, *file* exe, *file* DLL, dokumen, dll.

Diagram c*omponent* mengandung komponen, *interface* dan *relationship*. Komponen diagram ini digunakan pada saat pengembang ingin memecah system menjadi komponenkomponen dan ingin menampilkan hubunganhubungan mereka dengan antarmuka atau pemecahan komponen menjadi struktur yang lebih rendah. Secara umum dapat disimpulkan bahwa component diagram yang digunakan untuk menjelaskan kebergantungan antar beragam komponenkomponen *software* seperti misalnya kebergantungan antara *filefile executable* dengan *filefile* sumbernya (*source file*) dll.

#### 3.2.3.3.3.11 Diagram *Deployment*

Diagram *deployment* menunjukkan tata letak sebuah sistem secara fisik, menampakkan bagianbagian *software* yang berjalan pada bagianbagian *hardware* yang digunakan untuk mengimplementasikan sebuah sistem dan keterhubungan antara komponenkomponen *hardware* tersebut. Diagram *deployment* dapat

digunakan pada bagianbagian awal proses perancangan system untuk mendokumentasikan arsitektur fisik sebuah sistem.

### 3.2.3.4 Perangkat Lunak Pendukung Pemodelan UML

Hasil-hasil pemodelan yang dikomunikasikan dengan pelanggan agar lebih baik dan lebih profesional jika diformat dengan baik menggunakan *software* (*case tool*) yang bisa menyajikan gambargambar hasil pemodelan dengan rapi. *Software* yang digunakan adalah *Enterprise Architect* 7.0 dari *Sparx System*.

Kegunaannya bernilai tambah lebih tinggi lagi, terutama untuk tim pengembang, apabila *software* tersebut memiliki fungsi *autogenerate* dari model ke kode sumber, dan fungsi *reverse engineering* dari kode sumber menjadi sebuah model. Sehingga hasil kerja *programmer* yang dituangkan dalam kode sumber bahasa pemrograman tertentu bisa divisualisasikan menggunakan model UML hasil *reverse engineer* dari software tersebut.

Nilai tambah lebih tinggi juga bisa diberikan oleh sebuah *case tool* apabila menyediakan integrasi dengan *Integrated Development Environment* (IDE) dari sebuah bahasa pemrograman seperti *Microsoft Visual Studio* (.NET) dan *Eclipse* (Java). IDE tertentu biasanya mempunyai kelebihan yang telah dioptimisasi untuk bahasa pemrograman dan *platform* tertentu, sehingga lebih menjadi pilihan *programmer* daripada membuat kode sumbernya langsung di sebuah *case tool*. Dengan adanya integrasi antara *case tool* dengan sebuah IDE, proses *auto generate* dan *reverse engineering* tentu bisa dilakukan dengan lebih mudah dan lebih cepat.

Sesuai dengan prinsip *agile* sehingga penulis tidak perlu menggunakan seluruh model yang ada dalam UML dalam melakukan pemodelan dengan *software* ini. Fitur-fitur yang cukup banyak dan menarik dipakai dari pandangan seorang *software architect* adalah *auto generate* dari sebuah *class diagram* menjadi *language specific diagram* dan kode sumber.

### 3.4 Faktor Pengujian Software

Pengujian perangkat lunak adalah proses pengecekan terhadap suatu perangkat lunak, apakah perangkat lunak yang dihasilkan sudah dapat dijalankan sesuai dengan standar tertentu. Standar yang menjadi acuan dapat berupa menurut instansi tertentu ataupun disesuaikan dengan keperluan costumer dan user.

Metode pengujian yang akan dilakukan penulis pada pengujian perangkat lunak yang akan dibangun adalah metode pengujian *black-box*, dimana pada pengujia *black-box* ini hanya berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian didapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Metode pengujian ini dilakukan untuk menemukan kesalahan yang mungkin terjadi pada sebuah perangkat lunak seperti :

1. Fungsi-fungsi tidak benar atau hilang
2. Kesalahan tatap muka
3. Kesalahan dalam struktur data atau akses basis data eksternal
4. Kesalahan kinerja
5. Kesalahan inisialisasi dan kesalahan terminasi