BAB II
TINJAUAN PUSTAKA

2.1 Kecerdasan Buatan

2.1.1 Definisi Kecerdasan Buatan /Artificial Intelligence (AI)

Kecerdasan buatan atau artificial intelligence merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan jaman, maka peran komputer semakin mendominasi kehidupan umat manusia. Komputer tidak lagi digunakan sebagai alat hitung, lebih dari itu, komputer diharapkan untuk dapat diberdayakan untuk mengerjakan segala sesuatu yang bisa dikerjakan oleh manusia.

Agar komputer bisa bertindak seperti dan sebaik manusia, maka komputer juga harus diberi bekal pengetahuan, dan mempunyai kemampuan untuk menalar. Untuk itu pada AI, akan mencoba untuk memberikan beberapa metoda untuk membekali komputer dengan kedua komponen tersebut agar komputer bisa menjadi mesin yang pintar.

Pengertian kecerdasan buatan dapat dipandang dari berbagai sudut pandang, antara lain:

1. Sudut pandang kecerdasan.

   Kecerdasan buatan akan membuat mesin menjadi cerdas (mampu berbuat seperti apa yang dilakukan oleh manusia).

2. Sudut pandang penelitian.

   Kecerdasan buatan adalah suatu studi bagaimana membuat agar komputer dapat melakukan sesuatu sebaik yang dikerjakan oleh manusia. Domain yang sering dibahas oleh para peneliti meliputi :

a. Mundane task

1. Persepsi (vision & speech)
2. Bahasa alami (understanding, generation, & translation)
3. Pemikiran yang bersifat commonsense.
4. Robot control.

b. Formal task

1. Permainan/games.
2. Matematika (geometri, logika, kalkulus integral, pembuktian)

c. Expert task

1. Analisis finansial
2. Analisis medikal
3. Analisis ilmu pengetahuan
4. Rekayasa (desain, pencarian kegagalan, perencanaan manfaktur)

3. Sudut pandang bisnis.

Kecerdasan buatan adalah kumpulan peralatan yang sangat powerful dan metodologis dalam menyelesaikan masalah-masalah bisnis.

4. Sudut pandang pemograman.

Kecerdasan buatan meliputi studi tentang pemograman simbolik, penyelesaian masalah (problem solving) dan pencarian (searching). Untuk melakukan aplikasi kecerdasan buatan ada 2 bagian utama yang sangat dibutuhkan, yaitu :

a. Basis pengetahuan (knowledge base), berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan lainnya.
b. Motor inferensi (inference engine), yaitu kemampuan menarik kesimpulan berdasarkan pengalaman.

2.1.2 Sejarah Kecerdasan Buatan

Kecerdasan buatan termasuk bidang ilmu yang relatif muda. Pada tahun 1950-an para ilmuwan dan peneliti mulai memikirkan bagaimana caranya agar mesin dapat melakukan pekerjaannya seperti yang bisa

dikerjakan oleh manusia. Alan turing, seorang matematikawan dari Inggris pertama kali mengusulkan adanya tes untuk bisa

melihat bisa tidaknya sebuah mesin dikatakan cerdas. Hasil tes tersebut kemudian dikenal dengan Turing test, dimana si mesin tersebut menyamar seolah-olah sebagai seseorang di dalam suatu permainan yang mampu memberikan respon terhadap serangkaian pertanyaan yang diajukan. Turing beranggapan bahwa, jika mesin dapat membuat seseorang percaya bahwa dirinya mampu berkomunikasi dengan orang lain, maka dapat dikatakan bahwa mesin tersebut cerdas (seperti layaknya manusia).

Kecerdasan buatan atau artificial intelligence itu sendiri dimunculkan oleh seorang professor dari Massachusetts Institute of Technology yang bernama John McCarthy pada tahun 1956 pada Dartmouth Conference yang dihadiri oleh para peneliti AI. Pada konferensi tersebut juga didefinisikan tujuan utama dari kecerdasan buatan, yaitu: mengetahui dan memodelkan proses-proses berfikir manusia dan mendesain mesin agar dapat menirukan kelakuan manusia tersebut.

Beberapa program AI yang mulai dibuat pada tahun 1956-1966, antara lain :

1. Logic Theorist, diperkenalkan pada Darmouth Conference, program ini dapat membuktikan teorema-teorema matematika.
2. Sad Sam, diprogram oleh Robert K. Lindsay (1960). Program ini dapat mengetahui kalimat-kalimat sederhana yang ditulis dalam bahasa inggris dan mampu memberikan jawaban dari fakta-fakta yang didengar dalam sebuah percakapan.
3. ELIZA, diprogram oleh Joseph Weizenbaum (1967). Program ini mampu melakukan terapi terhadap pasien dengan memberikan beberapa pertanyaan.

2.1.3 Lingkup Kecerdasan Buatan pada Aplikasi Komersial

Dewasa ini kecerdasan buatan juga memberikan kontribusi yang cukup besar di bidang manajemen. Adanya system pendukung keputusan, dan Sistem Informasi Manajemen juga tidak terlepas dari andil kecerdasan buatan.

Adanya irisan penggunaan kecerdasan buatan di berbagai disiplin ilmu tersebut menyebabkan cukup rumitnya untuk mengklasifikasikan kecerdasan buatan menurut disiplin ilmu yang menggunakannya. Untuk memudahkan hal tersebut, maka pengklasifikasian lingkup kecerdasan buatan didasarkan pada output yang diberikan yaitu pada aplikasi komersial (meskipun sebenarnya kecerdasan buatan itu sendiri bukan merupakan medan komersial).

Lingkup utama dalam kecerdasan buatan adalah:

1. Sistem Pakar (Expert System). Disini komputer digunakan sebagai sarana untuk menyimpan pengetahuan para pakar. Dengan demikian komputer akan memiliki keahlian untuk menyelesaikan masalah permasalahan dengan meniru keahlian yang dimiliki oleh pakar.
2. Pengolahan Bahasa alami (Nature Language Processing). Dengan pengolahan bahasa alami ini diharapkan user dapat berkomunikasi dengan komputer dengan menggunakan bahasa sehari-hari.
3. Pengenalan Ucapan (Speech Recognition). Melalui pengenalan ucapan diharapkan manusia dapat berkomnikasi dengan komputer dengan menggunakan suara.
4. Robotika dan Sistem Sensor (Robotics and Sensor Systems).
5. Computer Vision, mencoba untuk dapat mengintepretasikan gambar atau obyek-obyek tampak melalui komputer.
6. Intelligent Computer-aided Instruction. Komputer dapat digunakan sebagai tutor yang dapat melatih dan mengajar.
7. Game Playing.

   Beberapa karakteristik yang ada pada system yang menggunakan artificial intelligence adalah pemogramannya yang cenderung bersifat simbolik ketimbang algoritmik, bisa mengakomodasi input yang tidak lengkap, bisa melakukan inferensi, dan adanya pemisahan antara kontrol dengan pengetahuan.

2.2 *Natural Language Processing (NLP)*

Natural Language Processing (NLP) atau Pemrosesan Bahasa Alami (PBA) adalah cabang ilmu komputer dan linguistik yang mengkaji interaksi antara komputer dengan bahasa (alami) manusia. Pemrosesan bahasa alami sering dianggap sebagai cabang dari kecerdasan buatan dan bidang kajiannya bersinggungan dengan linguistik komputasional. Kajian pemrosesan bahasa alami antara lain mencakup segmentasi tuturan (speech segmentation), segmentasi teks (text segmentation), penandaan kelas kata (part-of-speech tagging), serta pengawataksaan makna (word sense disambiguation). Meskipun kajian yang mencakup teks dan tuturan, pemrosesan tuturan (speech processing) berkembang menjadi suatu bidang kajian terpisah. Tujuan dalam bidang natural language adalah melakukan proses pembuatan model komputasi dari bahasa, sehingga terjadi interaksi antara manusia dengan komputer dengan perantara bahasa alami. Model komputasi ini dapat berguna untuk keperluan ilmiah seperti meneliti sifat-sifat dari suatu bentuk bahasa alami maupun uttuk keperluan sehari-sehari, dalam hal ini memudahkan komunikasi antara manusia dengan komputer. Bidang-bidang pengetahuan yang berhubungan dengan pengolahan bahasa alami adalah sebagai berikut:

1. Fonetik dan fonologi : berhubungan dengan suara yang menghasilkan kata yang dapat dikenali. Bidang ini penting dalam aplikasi yang memakai metode *speech-based system*.
2. Morfologi : yaitu pengetahuan tentang kata dan bentuknya yang dimanfaatkan untuk membedakan satu kata dengan kata lainnya. Pada tingkat ini juga dapat dipisahkan antara kata dan elemen lain seperti tanda baca.
3. Sintaksis : yaitu pemahaman tentang urutan kata dan pembentukan kalimat dan hubungan antar kata tersebut dalam proses perubahan bentuk dari kalimat menjadi sesuatu yang sistematis.
4. Semantik : yaitu pemetaan bentuk struktur sintaksis dengan memanfaatkan tiap kata ke dalam bentuk yang lebih mendasar dan tidak tergantung dengan struktur kalimat.

5. Pragmatik : berkaitan dengan tingkatan pengetahuan masing-masing konteks yang berbeda tergantung pada situasi dan tujuan pembuatan sistem.
6. *Discourse knowledge* : melakukan pengenalan apakah suatu kalimat yang sudah dibaca dan dikenali sebelumnya dalam mempengaruhi arti dari kalimat selanjutnya. Informasi ini penting diketahui untuk melakukan pengolahan arti terhadap kata ganti orang dan untuk mengartikan aspek sementara dari informasi.
7. *World knowledge* : mencakup arti sebuah kata secara umum dan apakah ada arti khusus bagi suatu kata dalam suatu percakapan dengan konteks tertentu.

Jenis aplikasi yang terdapat pada bidang pengolahan bahasa alami antara lain adalah :

1. *Text-based application* : mencakup segala macam aplikasi yang melakukan terhadap teks tertulis seperti pada buku, berita di surat kabar, *e-mail* dan lain sebagainya. Contoh penggunaan aplikasi ini antara lain adalah mencari topik tertentu dari buku yang ada di perpustakaan, mencari isi dari surat atau *e-mail*, menerjemahkan dokumen dari satu bahasa ke bahasa yang lain. Akan tetapi, tidak semua sistem dapat melakukan hal yang demikian, misalnya pada pencarian topik dari suatu buku di perpustakaan dapat dilakukan dengan pendekatan sistem *database* yang cukup lengkap. Salah satu bentuk yang cukup menarik adalah jika sistem diminta mencari isi dari suatu buku atau artikel, dimana pendekatan yang dilakukan sama seperti pendekatan yang dilakukan oleh manusia jika menghadapi suatu tes *reading and comprehension*.
2. *Dialogue-based application* : merupakan pendekatan yang melibatkan bahasa lisan atau pengenalan suara. Akan tetapi, bidang ini juga memasukkan interaksi dengan cara memasukkan teks pertanyaan melalui *keyboard*. Aplikasi yang sering ditemui dalam bidang ini antara lain seperti sistem tanya jawab dimana *natural language* digunakan dalam mendapatkan informasi dari *database*, sistem otomasi pelayanan melalui telepon, kontrol suara pada peralatan elektronik, sistem *problem-solving*

yang membantu untuk melakukan penyelesaian masalah yang umum dihadapi dalam suatu pekerjaan. Perlu diketahui bahwa untuk sistem yang dapat melakukan interaksi melalui bahasa lisan ada pada bagian *speech recognition* yang merupakan bagian terpisah dari *natural language*.

2.3 Agen Cerdas (*Intellgent Agent*)

Agen cerdas adalah agen yang pada mulanya berasal dari suatu cabang ilmu *artificial intelligence* (AI) yang bernama *distributed artificial intelligence* (DAI), seiring dengan berjalannya waktu DAI memiliki perkembangan dalam hal pengertiannya yaitu agen berada pada lingkungan tertentu dan dapat berinteraksi dengan lingkungannya dengan menggunakan *sensor* dan *effector*. Sehingga terdapat proses yang terjadi secara berulang-ulang pada agen yaitu agen menerima masukan (*percept*) dari lingkungan dengan menggunakan *sensor*, lalu agen mencari dan memilih tindakan yang sesuai untuk masukan tadi, kemudian memberikan tindakan (*action*) yang telah dipilih tadi ke lingkungannya dengan menggunakan *effector*. Tindakan yang telah dilakukan oleh agen ke lingkungannya dapat mempengaruhi dan mengubah lingkungannya tersebut.

2.4 *Chatbot*

Chatbot adalah program komputer yang mensimulasikan percakapan cerdas. Proses *chatbot* dimulai dengan masukan dari pengguna menggunakan bahasa alami dan sistem akan menjawab dengan respon yang masuk akal atau bisa dikatakan cerdas untuk bahasa yang sebenarnya.

2.4.1 Sejarah singkat *chatbot*

Sejarah klasik dari *chatterbot* awal adalah ELIZA (1966) dan PARRY (1972). Program yang baru-baru saja dikembangkan yaitu A.L.I.C.E, Jabberwacky dan D.U.D.E. Pada masanya, ELIZA dan PARRY digunakan untuk menstimulasi percakapan tertulis, namun banyak *chatterbot* kini mendukung fitur fungsional seperti permainan dan kemampuan pencarian website. Tahun 1984, sebuah buku berjudul *The Policeman's Beard is Half Constructed* dipublikasikan. Buku ini diduga ditulis oleh

sebuah *chatterbot* Racter –walaupun program ini dirilis untuk tidak mampu melakukannya.

Salah satu penelitian penting di bidang kecerdasan buatan (AI) adalah pemrosesan bahasa alami(*Natural Procesing Language*). Biasanya bidang AI lemah memberdayakan perangkat lunak (atau *software*) khusus atau bahasa pemrograman yang dibuat secara spesifik dengan fungsi yang lebih sempit. Contohnya A.L.I.C.E menggunakan bahasa pemrograman yang disebut sebagai AIML yang fungsinya spesifik yaitu sebagai agen percakapan, selanjutnya banyak diadopsi oleh pengembang *Alicebots* lain. Meskipun demikian, A.L.I.C.E masih murni berdasarkan teknik pencocokan pola tanpa kemampuan penalaran –teknik yang sama yang digunakan ELIZA pada tahun 1966. Berbeda dengan AI kuat, yang membutuhkan cita rasa dan kemampuan penalaran logis.

Jabberwacky mempelajari respons baru dan berbasis pada konteks interaksi pengguna waktu nyata (atau *real-time*), bukan dengan digerakan basis data statis. Beberapa *chatterbot* terbaru juga mengkombinasikan pembelajaran waktu nyata dengan algoritma yang mengoptimalkan kemampuan berkomunikasi berbasis percakapannya, salah satu contoh yang sangat populernya yaitu Kyle, pemenang Penghargaan Leodis AI 2009. Meskipun, saat ini belum ada tujuan umum percakapan kecerdasan buatan, dan beberapa pengembang perangkat lunak berfokus pada aspek praktis, pengambilan informasi (atau *information retrieval*).

2.5 String

String dalam ilmu komputer dapat diartikan dengan sekuens dari karakter. Walaupun sering juga dianggap sebagai data abstrak yang menyimpan sekuens nilai data, atau biasanya berupa bytes yang mana merupakan elemen yang digunakan sebagai pembentuk karakter sesuai dengan encoding karakter yang disepakati seperti ASCII, ataupun EBCDIC. Hubungan string dengan penelitian ini adalah bahwa karakteristik dari informasi yang akan disimpan dalam database dapat dianggap serupa dengan string. Hal ini akan memudahkan programmer dalam membangun sistem

pencocokan karakter dari sampel yang akan dikonversi terlebih dahulu menjadi serupa dengan string ataupun deretan bytes. Konversi inilah yang nantinya akan dibandingkan langsung dengan informasi karakteristik yang disimpan dalam database.

2.6 *Fuzzy String Matching*

Logika *Fuzzy* adalah peningkatan dari logika *Boolean* yang berhadapan dengan konsep kebenaran sebagian. Di mana logika klasik menyatakan bahwa segala hal dapat diekspresikan dalam istilah *binary* (0 atau 1, hitam atau putih, ya atau tidak), logika *fuzzy* menggantikan kebenaran *boolean* dengan tingkat kebenaran. Logika *Fuzzy* memungkinkan nilai keanggotaan antara 0 dan 1, tingkat keabuan dan juga hitam dan putih, dan dalam bentuk linguistik, konsep tidak pasti seperti "sedikit", "lumayan", dan "sangat". Dia berhubungan dengan set *fuzzy* dan teori kemungkinan. Dia diperkenalkan oleh Dr. Lotfi Zadeh dari Universitas California, Berkeley pada 1965.

Fuzzy string matching adalah salah satu metode pencarian *string* yang menggunakan proses pendekatan terhadap pola dari *string* yang dicari. Metode ini termasuk dalam katagori *inexact matching* dimana konsep ini melakukan pencarian terhadap *string* yang sama dan juga *string* yang mendekati dengan *string* lain yang terkumpul dalam sebuah penampung atau kamus.

Kunci dari konsep pencarian ini adalah bagaimana memutuskan bahwa sebuah *string* yang dicari memiliki kesamaan dengan *string* tertampung di kamus, meskipun tidak sama persis dalam susunan karakternya. Untuk memutuskan ‘kesamaan’ ini dipergunakan sebuah fungsi yang diistilahkan sebagai *similarity function*. Berbagai metode sudah dikembangkan dalam menentukan *similarity function*. Fungsi ini akan bertugas memutuskan *string* hasil pencarian jika ditemukan *string* hasil pendekatan (aproksimasi).

Inexact string matching atau juga disebut *Fuzzy string matching*, merupakan pencocokan *string* secara samar, maksudnya pencocokan *string* dimana *string* yang dicocokkan memiliki kemiripan dimana keduanya

memiliki susunan karakter yang berbeda (mungkin jumlah atau urutannya) tetapi *string-string* tersebut memiliki kemiripan baik kemiripan tekstual/ penulisan (*approximate string matching*) atau kemiripan ucapan (*phonetic string matching*).

Inexact string matching masih dibagi menjadi dua yaitu:

a) Pencocokan *string* berdasarkan kemiripan penulisan (*approximate string matching*) merupakan pencocokan *string* dengan dasar kemiripan dari segi penulisannya (jumlah karakter, susunan karakter dalam dokumen). Tingkatkemiripan ditentukan dengan jauh tidaknya beda penulisan dua buah *string* yang dibandingkan tersebut dan nilai tingkat kemiripan ini ditentukan oleh pemrogram (*programmer*).

   Contoh: *computer* dengan *compiler*, memiliki jumlah karakter yang sama tetapi ada dua karakter yang berbeda. Jika perbedaan dua karakter ini dapat ditoleransi sebagai sebuah kesalahan penulisan maka dua *string* tersebut dikatakan cocok.

b) Pencocokan *string* berdasarkan kemiripan ucapan (*phonetic string matching*) merupakan pencocokan *string* dengan dasar kemiripan dari segi pengucapannya meskipun ada perbedaan penulisan dua *string* yang dibandingkan tersebut.

   Contoh: *step* dengan s*teb* dari tulisan berbeda tetapi dalam pengucapannya mirip sehingga dua *string* tersebut dianggap cocok. Contoh yang lain adalah *step*, dengan *steppe, sttep, stepp, stepe. Exact string matching* bermanfaat jika pengguna ingin mencari *string* dalam dokumen yang sama persis dengan *string* masukan. Tetapi jika pengguna menginginkan pencarian *string* yang mendekati dengan *string* masukan atau terjadi kesalahan penulisan *string* masukan maupun dokumen objek pencarian, maka *inexact string matching* yang bermanfaat. Beberapa algoritma *exact string matching* antara lain: algoritma Knuth-Morris Pratt, Bayer-Moore, dll.

2.7 Unified Modeling Languange (UML)

UML menawarkan diagram yang dikelompokkan menjadi lima perspektif berbeda untuk memodelkan suatu sistem. Berikut ini adalah berbagai diagram UML serta tujuannya.

2.7.1 Diagram Model Use-Case

Diagram Use-Case, secara grafis menggambarkan interaksi antara sistem, sistem eksternal, dan pengguna. Mendiskripsikan siapa yang akan menggunakan sistem dan dalam cara apa pengguna mengharapkan interaksi dengan sistem itu. Use-Case naratif digunakan untuk secara tesktual menggambarkan sekuensi langkah-langkah dari setiap interaksi.

2.7.2 Diagram Struktur Statis

UML menawarkan dua diagram untuk memodelkan struktur sistem informasi statis, yaitu :

1. Diagram Kelas, menggambarkan struktur Objek sistem, menunjukkan kelas objek yang menyusun sistem dan juga hubungan antara kelas objek tersebut.
2. Diagram Objek, serupa dengan diagram kelas, namum diagram objek memodelkan instance objek aktual dengan menunjukkan nilai-nilai saat ini dari atribut instance. Diagram objek menyajikan sebuah “snapshot” tentang objek sistem pada poin waktu tertentu.

2.7.3 Diagram Interaksi

Diagram interaksi memodelkan sebuah interaksi, terdiri dari satu set objek, hubungan-hubungannya, dan pesan yang terkirim di antara objek. Model diagram ini memodelkan behaviour sistem yang dinamis dan UML mempunyai dua diagram untuk tujuan ini, yaitu :

1. Diagram Rangkaian/Sekuensi, secara grafis menggambarkan bagaimana objek berinteraksi dengan satu sama lain melalui pesan pada eksekusi sebuah use-case atau operasi. Diagram ini mengilustrasikan bagaimana pesan terkirim dan diterima di antara objek dan dalam sekuensi apa.

2. Diagram Kolaborasi, serupa dengan diagram rangkaian/sekuensi, tetapi tidak fokus pada timing atau "sekuensi" pesan. Diagram ini menggambarkan interaksi (atau kolaborasi) antara objek dalam sebuah format jaringan.

2.7.4 Diagram *State* (*State Diagram*)

Diagram bagian juga memodelkan *behavior* dinamis dari sistem. *State diagram* terdiri dari :

1. Diagram *Statechart*, digunakan untuk memodelkan behaviour objek khusus yang dinamis. Diagram ini mengilustrasikan siklus hidup siklus hidup objek berbagai keadaan yang dapat diasumsikan oleh objek dan event-event yang menyebabkan objek beralih dari satu state ke state lain.
2. Diagram Aktivitas, secara grafis digunakan untuk menggambarkan rangkaian aliran aktivitas baik proses bisnis atau use-case. Diagram ini juga dapat digunakan untuk memodelkan action yang akan dilakukan saat sebuah operasi dieksekusi, dan memodelkan hasil dari action tersebut.

2.7.5 Diagram Implementasi

Diagram implementasi juga memodelkan struktur sistem informasi, yaitu :

1. Diagram Komponen, digunakan untuk mengambarkan organisasi dan ketergantungan komponen-komponen software sistem. Diagram ini dapat digunakan untuk menunjukkan bagaimana kode pemrograman dibagi menjadi modul-modul (atau komponen).
2. Diagram Penguraian/*Deployment*, mendeskripsikan arsitektur fisik dalam istilah node, untuk hardware dan software dalam sistem. Diagram ini menggambarkan konfigurasi komponen-komponen

software run-time, processor, dan peralatan yang membentuk arsitektur sistem.

2.8 *Rational Unified Proses* (RUP)

2.8.1 Definisi *Rational Unified Proses* (RUP)

Rational Unified Process (RUP) merupakan suatu metode rekayasa perangkat lunak yang dikembangkan dengan mengumpulkan berbagai *best practises* yang terdapat dalam industri pengembangan perangkat lunak. Ciri utama metode ini adalah menggunakan *use-case driven* dan pendekatan iteratif untuk siklus pengembangan perangkat lunak. RUP menggunakan konsep *object oriented*, dengan aktifitas yang berfokus pada pengembangan model dengan menggunakan *Unified Model Language* (UML).

2.8.2 Arsitektur Rational Unified Proses (RUP)

Dalam *Rational Unified Process* (RUP) memiliki dua dimensi , yaitu:

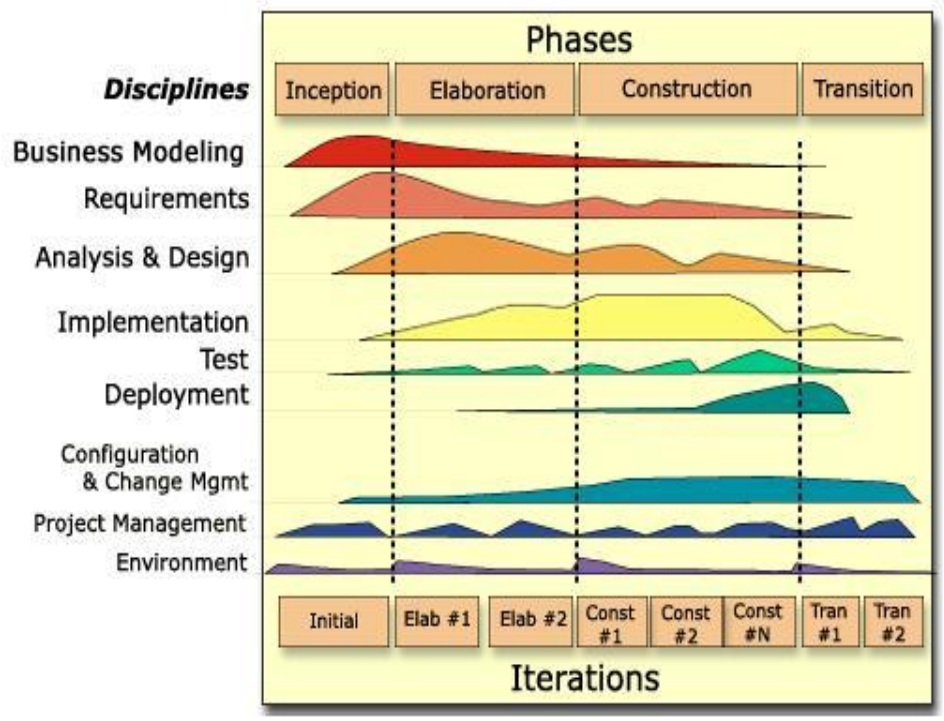
1. Dimensi pertama.

   Digambarkan secara horizontal. Dimensi ini mewakili aspek-aspek dinamis dari pengembangan perangkat lunak. Aspek ini dijabarkan dalam tahapan pengembangan atau fase. Setiap fase akan memiliki suatu major milestone (tonggak keberhasilan) yang menandakan akhir dari fase tersebut dan awal dari phase selanjutnya. Setiap fase dapat terdiri dari satu beberapa iterasi. Dimensi ini terdiri atas *Inception, Elaboration, Construction*, dan *Transition*.

2. Dimensi kedua.

   Digambarkan secara vertikal. Dimensi ini mewakili aspek-aspek statis dari proses pengembangan perangkat lunak yang dikelompokkan ke dalam beberapa disiplin. Proses pengembangan perangkat lunak yang dijelaskan kedalam beberapa disiplin terdiri dari empat elemen penting, yakni *who is doing, what, how* dan *when*. Dimensi ini terdiri atas *Business Modeling, Requirement, Analysis and Design*,

Implementation, Test, Deployment, Configuration and Change Manegement, Project Management, dan *Environtment*.



Gambar 2.1 Arsitektur Rational Unified Process

Adapun manfaat-manfaat penggunaan kedua dimensi tersebut diantaranya:

1. *Improve productivity*

   Standard ini dapat memanfaatkan kembali komponen-komponen yang telah tersedia/dibuat sehingga dapat meningkatkan produktifitas.

2. *Deliver high quality system*

   Kualitas sistem informasi dapat ditingkatkan sebagai sistem yang dibuat pada komponen-komponen yang telah teruji (*well-tested* dan *well-*

proven) sehingga dapat mempercepat delivery sistem informasi yang dibuat dengan kualitas yang tinggi.

3. *Lower maintenance cost*

   Standard ini dapat membantu untuk menyakinkan dampak perubahan yang terlokalisasi dan masalah dapat dengan mudah terdeteksi sehingga hasilnya biaya pemeliharaan dapat dioptimalkan atau lebih rendah dengan pengembangan informasi tanpa standard yang jelas.

4. *Facilitate reuse*

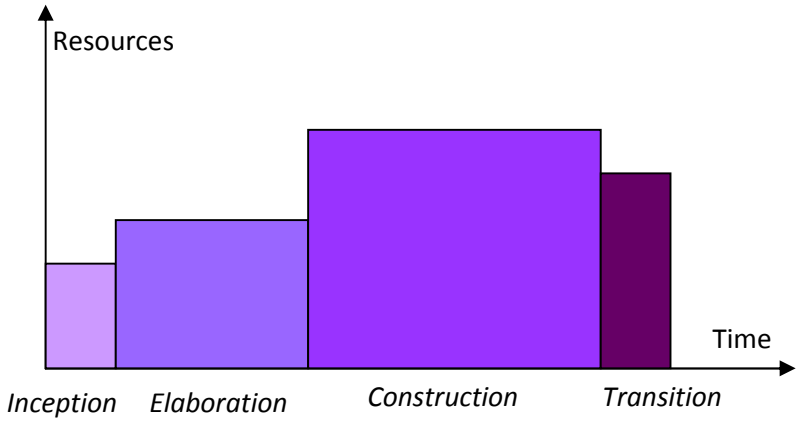
   Standard ini memiliki kemampuan yang mengembangkan komponen-komponen yang dapat digunakan kembali untuk pengembangan sistem aplikasi yang lainnya.

5. *Manage complexity*

   Standard ini mudah untuk mengatur dan memonitor semua proses dari semua tahapan yang ada sehingga suatu pengembangan sistem informasi yang amat kompleks dapat dilakukan dengan aman dan sesuai dengan harapan semua manajer proyek IT/IS yakni *deliver good quality software within cost* dan *schedule time* dan *the users accepted*.

2.8.3 Daur Hidup Rational Unified Proses (RUP)

Daur hidup RUP secara umum akan tampak seperti pada gambar 2.2 di bawah ini.



Gambar 2.2 Daur Hidup RUP

Bagian ini biasa disebut sebagai "*hump chart*". Pada bagian ini terlihat ada empat tahapan pengembangan yaitu *Inception, Elaboration, Construction,* dan *Transition*. Selain itu tampak pula sejumlah aktivitas (*dicipline*) yang harus dilakukan sepanjang pengembangan perangkat lunak (terdapat pada gambar 2.1), yaitu *business modeling, requirement, analysis and design, implementation, test.* Tahap dan aktivitas tersebut akan dilakukan secara iteratif.

2.8.3.1 *Inception*

Tahapan ini menjelaskan tentang *requirement* dari *software*, lebih terfokus pada *requirement* pengembangan *software* yang akan diapkai dan tujuan utamanya adalah ruang lingkup sistem yang memadai sebagai dasar untuk memvalidasi biaya awal dan anggaran. Untuk melengkapi kasus bisnis, kasus penggunaan model dasar, rencana proyek, penilaian resiko dan deskripsi awal proyek (persyaratan proyek inti, kendala dan fitur kunci) yang dihasilkan.

Outcome/hasil dari fase ini meliputi beberapa hal sebagai berikut :

1. Dokumen visi yang berisi : visi umum kebutuhan utama proyek, fitur-fitur kunci, dan kendala-kendala utama.
2. Model bisnis yang meliputi kontek bisnis, kriteria sukses (proyeksi pendapatan, pengakuan pasar, dll) dan perkiraan finansial.
3. Model use-case awal (10-20% lengkap).
4. Perkiraan resiko-resiko awal.
5. Perencanaan proyek dengan menunjukkan fase-fase umum dan iterasinya.

Milestone (tonggak keberhasilan) pertama yang ada akhir masa insepsi. Kriteria keberhasilan pada fase insepsi diukur dan dibuktikan dengan hal-hal sebagai berikut :

1. Didapatkannya persetujuan pihak pemangku kepentingan (stakeholder) tentang ruang lingkup proyek, anggaran dan estimasi jadwal proyek.
2. Pemahaman pemangku kepentingan tentang kebutuhan utama proyek yang dibuktikan dengan penerimaannya terhadap use-case utama.

3. Kredibilitas terhadap biaya, estimasi jadwal, prioritas, resiko, dan proses pengembangan.
4. Prototipe arsitektur sistem telah dikembangkan secara luas dan mendalam.
5. Membangun dasar yang digunakan untuk membandingkan biaya aktual dengan biaya yang direncanakan.

2.8.3.2 *Elaboration*

Tahapan ini adalah tahapan peninjauan kembali terhadap *requirement* bisnis untuk meminimalisir terjadinya perubahan pada tahap selanjutnya yaitu *Construction* dan tujuan utamanya adalah mengurangi item resiko utama diidentifikasi dengan analisis sampai dengan akhir fase ini. Fase elaborasi adalah tempat proyek mulai terbentuk. Dalam tahap analisis domain masalah yang dibuat dan proyek arsitektur mendapatkan bentuk dasar.

Outcome/hasil fase elaborasi antara lain diberikan sebagai berikut ini :

1. Model use-case (paling tidak 80% lengkap), setiap use-case dan aktor telah diidentifikasi dan sebagian besar diskripsi use-case telah dibuat.
2. Dokumen kebutuhan sistem tambahan yang tidak terkait langsung dengan fungsional sistem yang sedang dibangun.
3. Diskripsi arsitektur perangkat lunak.
4. Prototipe arsitektur executable.
5. Daftar resikoyang telah direvisi dan model bisnis yang telah direvisi.
6. Rencana pengembangan keseluruhan proyek, termasuk rencana global proyek, menunjukkan iterasi dan kriteria evaluasi keberhasilan untuk setiap iterasi.

Milestone (tonggak keberhasilan) kedua ada pada akhir fase elaborasi. Serupa dengan fase insepsi, kriteria evaluasi untuk fase elaborasi diukur dengan hal-hal sebagai berikut :

1. Apakah visi produk sudah stabil?
2. Apakah arsitektur sistem sudah stabil?

3. Apakah demonstrasi executable menunjukkan bahwa elemen-elemen resiko utama telah diidentifikasi dan dipecahkan?
4. Apakah semua pemangku kepentingan telah setuju dengan visi terbaru dengan sebuah keyakinan bahwa visi tersebut dapat dicapai jika rencana pengembangan dieksekusi?
5. Apakah biaya yang direncanakan dapat diterima?

2.8.3.3 *Construction*

Tahapan ini merupakan tahapan penerapan bisnis modeling yang telah terdefinisi dalam bentuk *coding* dari *software* yang dikembangkan beserta pengujian apakah *software* sudah memenuhi *requirement* awal. Tujuan utamanya adalah untuk membangun sistem perangkat lunak, pada tahap ini fokus utamanya adalah pada pengembangan komponen dan fitur lain dari sistem. Tahap ini menghasilkan rilis eksternal pertama dari perangkat lunak, kesimpulanya adalah ditandai dengan kemampuan operasional sebagai tonggak awal.

Outcome/hasil fase konstruksi sekurang-kurangnya menghasilkan beberapa hal antara lain diberikan dibawah ini :

1. Produk perangkat lunak yang telah diuji.
2. Buku petunjuk pemakaian untuk pemakai (manual *user* guide).
3. Penjelasan versi rilis terbaru.

Milestone (tonggak keberhasilan) ketiga ada pda akhir fase konstruksi. Kriteria evaluasi untuk fase konstruksi diukur dengan hal-hal sebagai berikut :

1. Apakah produk versi rilis terbaru cukup stabil dan mempunyai kematangan yang cukup untuk di-deploy ke komunitas pemakai?
2. Apakah semua pihak pemangku kepentingan siap mentransisikan produk ke komunitas pemakai?
3. Apakah biaya yang direncanakan dan biaya aktual masih berada pada toleransi yang masih dapat diterima?

2.8.3.4 *Transition*

Fase transisi dilakukan ketika perangkat lunak yang diproduksi telah selesi dan dideliveri ke pemakai dilakukan. Tugas-tugas di fase ini meliputi, melengkapi dan menyempurnakan perangkat lunak, melengkapi *acceptence testing*, melengkapi *manual user guide*, dan menyiapkan pelatihan pemakai. *Software Requirement Spesification (SRS)*, diagram *use-case*, diagram kelas, diagram komponen, dan diagram *deployment*, harus diperbaharui sesuai dengan perubahan terakhir.

Perhatikan sinkronisasi antara perangkat lunak versi terakhir yang telah dihasilkan dengan model, hal ini untuk kebutuhan pemeliharaan perangkat lunak dikemudian hari.

2.8.4 Bagian dan Cara Kerja *Rational Unified Proses* (RUP)

2.8.4.1 *Business Modeling* (Pemodelan Bisnis)

Tujuan utama bagian dalam *business modeling* disini adalah untuk memungkinkan adanya komunikasi dan pengertian yang lebih baik dari *business engineering* dan *software engineering*. Fase-fase yang terlibat dalam *business modeling* ini yaitu :

1. *Inception*

   Pertama kalinya business modeling dideklarasikan dan didefinisikan.

2. *Elaboration*

   Peninjauan kembali terhadap requirement bisnis untuk meminimalisasikan terjadinya perubahan pada tahap selanjutnya yaitu *Construction*.

3. *Construction*

   Penerapan dari business modeling yang telah terdefinisi dalam bentuk coding.

4. *Transition*

   Dimungkinkan apablia terjadi kesepakatan antara developer dengan end *user*s dalam perawatan software yang telah dibuat.

2.8.4.2 *Requirement* (Persyaratan/Kebutuhan)

Tujuan dari tahap ini adalah mendeskripsikan apa yang harus dilakukan oleh sistem dan memungkinkan terjadinya kesepakatan antara *customer* dan *developer* tentang hal itu. Fase-fase yang terlibat yaitu :

1. *Inception*

   Requirement dari software pertama kali dibahas. Lebih terfokus pada requirement pengembangan software yang akan dipakai.

2. *Elaboration*

   Mengurangi/meninjau kembali requirement dari software, dan dimungkinkan terjadi pergantian requirement dalam software yang akan dikembangkan.

3. *Construction*

   Perwujudan requirement yang ada dalam bentuk coding dari software yang dikembangkan beserta pengujian apakah software sudah memenuhi requirement awal.

4. *Transition*

   Fase ini bisa saja berupa requirement dari end *user*s untuk menambah aplikasi software, atau mungkin perawatan software.

2.8.4.3 *Analysis* dan *Design* (Analisis dan Desain)

Tujuan dalam tahap ini adalah untuk menunjukkan bagaimana project akan diwujudkan dalam fase implementasi kelak. Hasil dari tahap ini adalah *model design*. Fase-fase yang terlibat antara lain :

1. *Inception*

   Analysis dan design sudah mulai dibahas dengan adanya pembahasan tentang business modeling dan requirement.

2. *Elaboration*

   Fase inilah yang menjadi pusat perkembangan dari analysis dan design. Selain karena memegang segala macam domain, scope project, dan peninjauan kembali terjadi di fase ini. Hampir bisa depastikan bahwa perancangan dan analisa dilakukan pada fase ini.

3. *Construction*

   Dari design yang terbentuk dari tahap *Elaboration* tahapan ini dikembangkan menjadi bentuk coding.

2.8.4.4 *Implementation* (Implementasi)

Tujuan dari implementasi di sini adalah mendefinisikan pengkodean secara terorganisasi, mengimplimentasikan *classes* dan *objects* dalam bentuk komponen-komponen, menguji perkembangan komponen-komponen dalam bentuk kesatuan, dan mengintegrasikan hasil-hasil dari tiap-tiap kelompok yang mengerjakan *project*. Fase-fase yang terlibat antara lain :

1. *Inception*

   Pada tahap ini implementasi berlaku dengan terjadinya percakapan antara end *user*s dan developer mengenai software yang akan dikembangkan.

2. *Elaboration*

   Selain implementasi terhadap pembuatan use-case, tahap ini juga memuat implementasi dari perkembangan perencanaan arsitektural dan sebagainya.

3. *Construction*

   Dari nama fase ini, *Construction* alias konstruksi, tentu saja jelas dapat diambil kesimpulan, bahwa pada fase ini implementasi terhadap rancangan software dan sebagainya diterapkan.

4. *Transition*

   Implementasi yang terjadi pada tahap ini adalah penyerahan software terhadap end *user*s dan implementasi pada penerapan aplikasi software yang telah dikembangkan.

2.8.4.5 *Testing* (Pengujian)

Tujuan dari *testing* adalah memverifikasi interaksi antar *objects*, memverifikasi integrasi yang sesuai antar komponen, memverfikasi apakah semua *requirement* telah terpenuhi dengan benar, mengidentifikasi dan memastikan *defect* telah ditangani secara benar. Fase-fase yang terlibat :

1. *Inception*

   Dalam fase ini testing dilakukan apabila business modeling dan requirement telah teridentifikasi. Testing dilakukan dengan tujuan menghasilkan kesepakatan antara end *user*s dengan developer.

2. *Elaboration*

   Testing di sini merupakan testing setelah use case diimplementasikan, masih seputar tercapainya kesepakatan antara end *user*s dengan developer.

3. *Construction*

   Testing kebanyakan dilakukan di akhir fase *Construction*, karena setelah penyelesaian program-lah, testing baru dilaksanakan.

4. *Transition*

   Testing dilakukan sebelum penyerahan software kepada end *user*s dengan keadaan yang sebenarnya.

2.8.4.6 *Deployment* (Pengembangan)

Tujuan dari pengembangan di sini adalah suksesnya menghasilkan suatu project dan mengantarkan project tersebut pada end *user*s. Fase-fase yang terlibat :

1. *Elaboration*

   Mulailah pengembangan tentang realitas dari software itu akan seperti apa dalam fase ini.

2. *Construction*

   Dalam fase ini pengembangan software secara nyata terjadi dengan adanya coding.

3. *Transition*

   Fase yang paling berpengaruh karena adanya penyerahan software dari developer kepada end *user*s.

2.8.5 Kelebihan dan Kekurangan menggunakan Rational Unified Proses (RUP)

Ada beberapa keuntungan dengan menggunakan RUP diantaranya sebagai berikut :

1. Menyediakan akses yang mudah terhadap pengetahuan dasar bagi anggota tim.
2. Menyediakan petunjuk bagaimana menggunakan UML secara efektif.
3. Mendukung proses pengulangan dalam pengembangan software.
4. Memungkinkan adanya penambahan-penambahan pada proses.
5. Memungkinkan untuk secara sistematis mengontrol perubahan-perubahan yang terjadi pada software selama proses pengembangannya.
6. Memungkinkan untuk menjalankan test case dengan menggunakan Rational Test Manager Tool.

Meskipun demikian ada sedikit kekurangan terhadap model pengembangan perangkat lunak RUP ini, yakni metodologi ini hanya dapat digunakan pada pengembangan perangkat lunak yang berorientasi objek dengan berfokus pada UML (*Unified Modeling Language*).

2.9 Uji Penerimaan Pengguna (*User* Acceptance Test)

Teknik pengujian *software* adalah cara atau teknik untuk menguji perangkat lunak, mempunyai mekanisme untuk menentukan data uji yang dapat menguji perangkat lunak secara lengkap dan mempunyai kemungkinan tinggi untuk menentukan kesalahan.

User Acceptance Test (UAT) atau uji penerimaan pengguna adalah merupakan bagian kesepakatan antara pembeli/pengguna dan pengembang. Pengujian ini membentuk metode sederhana dan empiris untuk memutuskan apakah program memadai. Pengujian ini dapat dilihat sebagai bagian dari spesifikasi perangkat lunak. Program yang tidak lolos semua pengujian ini berarti tidak memenuhi spesifikasi (meskipun terdapat kriteria lain dispesifikasi yang tidak diuji, seperti percobaan pemakai, dokumentasi).

Pengujian ini umumnya dispesifikasikan sebelum program ditulis. Pengujian dapat menspesifikasikan aspek fungsional atau kinerja dari program. Pengujian ini dapat berlaku untuk potongan-potongan program, dimungkinkan untuk menguji suatu subsistem tertentu.

Manfaat dari pengujian ini dapat menyatakan apakah program memenuhi spesifikasi. Dalam hal ini, pengujian-pengujian dapat di pandang sebagai bukti bahwa pada kondisi-kondisi tertentu dan masukan-masukan tertentu, program yang di kembangkan bekerja secara benar.

3.0 Blackbox Testing

Metode ujicoba *blackbox* memfokuskan pada keperluan fungsional dari software. Karna itu ujicoba *blackbox* memungkinkan pengembang software untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba *blackbox* bukan merupakan alternatif dari ujicoba whitebox, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode whitebox.

Ujicoba *blackbox* berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya :

1. Fungsi-fungsi yang salah atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses *database* eksternal
4. Kesalahan performa
5. Kesalahan inisialisasi dan terminasi

Tidak seperti metode *whitebox* yang dilaksanakan diawal proses, ujicoba *blackbox* diaplikasikan dibeberapa tahapan berikutnya. Karena ujicoba *blackbox* dengan sengaja mengabaikan struktur kontrol, sehingga perhatiannya difokuskan pada informasi domain.

Ujicoba didesain untuk dapat menjawab pertanyaanpertanyaan berikut :

1. Bagaimana validitas fungsionalnya diuji?

2. Jenis input seperti apa yang akan menghasilkan kasus uji yang baik ?
3. Apakah sistem secara khusus sensitif terhadap nilai input tertentu ?
4. Bagaimana batasan-batasan kelas data diisolasi?
5. Berapa rasio data dan jumlah data yang dapat ditoleransi oleh sistem?
6. Apa akibat yang akan timbul dari kombinasi spesifik data pada operasi sistem?

Dengan mengaplikasikan ujicoba *blackbox*, diharapkan dapat menghasilkan sekumpulan kasus uji yang memenuhi kriteria berikut :

1. Kasus uji yang berkurang, jika jumlahnya lebih dari 1, maka jumlah dari ujikasus tambahan harus didesain untuk mencapai ujicoba yang cukup beralasan.
2. Kasus uji yang memberitahukan sesuatu tentang keberadaan atau tidaknya suatu jenis kesalahan, daripada kesalahan yang terhubung hanya dengan suatu ujicoba yang spesifik.