BAB II
TINJAUAN PUSTAKA

2.1 Sistem Informasi

Sistem informasi adalah pengaturan orang, data, proses, dan teknologi informasi yang berinteraksi untuk mengumpulkan, memproses, menyimpan, dan menyediakan sebagai output informasi yang diperlukan untuk mendukung sebuah organisasi. [4]

Sistem informasi terdiri dari komponen-komponen yaitu enam blok bangunan (*building block*), yang keenam blok tersebut masing-masing saling berinteraksi satu sama lain membentuk suatu kesatuan untuk mencapai sasaran.

a. Blok masukan (*input block*)

   Input mewakili data yang masuk ke dalam sistem informasi. Input yang dimaksud adalah metode dan media untuk menangkap data yang akan dimasukkan, yang dapat berupa dokumen-dokumen dasar.

b. Blok model (*model block*)

   Blok ini terdiri dari kombinasi prosedur, logika, dan model matematik yang akan memanipulasi data input dan data yang tersimpan di basis data dengan cara tertentu untuk menghasilkan keluaran yang diinginkan.

c. Blok keluaran (*output block*)

   Produk dari sistem informasi adalah keluaran yang merupakan informasi yang berkualitas dan dokumentasi yang berguna untuk semua tingkatan manajemen serta semua pemakai sistem.

d. Blok teknologi *(technology block*)

   Teknologi merupakan *tool box* dalam sistem informasi. Teknologi digunakan untuk menerima masukan, menjalankan model, menyimpan dan mengakses data, menghasilkan dan mengirimkan keluaran, dan membantu pengendalian dari sistem secara keseluruhan. Teknologi terdiri dari 3 bagian utama yaitu, teknisi (*brainware*), perangkat lunak (*software*), dan perangkat keras (*hardware*).

e. Blok basis data (*database block*)

   Basis data (*database*) merupakan komponen data yang saling berkaitan dan berhubungan satu sama lain, tersimpan di perangkat keras komputer dan

menggunakan perangkat lunak untuk memanipulasinya. Data perlu disimpan dalam basis data untuk keperluan penyediaan informasi lebih lanjut. Data di dalam basis data perlu diorganisasikan sedemikian rupa supaya informasi yang dihasilkan berkualitas. Organisasi basis data yang baik juga berguna untuk efisiensi kapasitas penyimpanannya. Basis data diakses atau dimanipulasi menggunakan perangkat lunak yang disebut DBMS (*Database Management System*).

f. Blok kendali (*control block*)

Banyak hal yang dapat merusak sistem informasi, seperti bencana alam, api, temperatur, air, debu, kecurangan, kegagalan sistem itu sendiri, ketidakefisienan, sabotase, dan lain sebagainya. Beberapa pengendendalian perlu dirancang dan diterapkan untuk meyakinkan bahwa hal-hal yang dapat merusak sistem dapat dicegah ataupun bila terlanjur terjadi kesalahan-kesalahan dapat langsung cepat diatasi. [5]

2.2 Diary

Istilah *diary* berasal dari bahasa Latin yaitu *diarium* yang berarti catatan harian [6]. Makna *diary* sendiri adalah sebuah buku tempat seseorang menyimpan catatan harian dari suatu peristiwa dan pengalaman [7]. Dalam bahasa Indonesia, *diary* disebut juga sebagai catatan harian yaitu catatan mengenai kegiatan sehari-hari [8]. Dapat disimpulkan, *diary*/catatan harian berarti catatan yang dibuat seseorang mengenai peristiwa atau pengalaman sehari-hari.

Penggunaan *diary* sendiri mulai berkembang pada akhir zaman Renaisans, ketika pentingnya individu mulai ditekankan. Selain mengungkapkan tentang kepribadian penulis *diary*, *diary* juga sangat penting sebagai pencatatan sejarah sosial dan politik. Minat dalam *diary* meningkat pesat pada awal abad ke-19, dimana periode tersebut banyak *diary* milik tokoh terkemuka di masanya, pertama kali diterbitkan. Contohnya ialah pada abad ke-20, *diary* milik penjelajah Robert F. Scott (1910-1912), *Journal of Katherine Mansfield* (1927), dua volume *Journal of Andre Gide* (1939, 1954), *The Diary of a Young Girl Anne Frank* (1947) dan lima volume *Diary of Virginia Woolf* (1977-1984) merupakan diantaranya yang terkenal dan diterbitkan. [9]

Entri *diary* adalah sebuah bagian dari penulisan yang diorganisasikan berdasarkan tanggal. Entri dalam *diary* ialah bagaimana penulis *diary* (*diarist*) mengorganisasikan pikiran, perasaan, dan pendapat yang dituangkannya. Entri *diary* memecah *diary* kedalam bagian-bagian yang lebih kecil, dan dapat diibaratkan seperti bab-bab dalam sebuah buku. Entri *diary* dapat bersifat pendek atau panjang tergantung dari keinginan *diarist*. [10]

2.2.1 Teknik Menulis *Diary*

Entri *diary* ialah seperti bab-bab dalam sebuah buku, karena itu penulis *diary* perlu menentukan akan seperti apa buku atau *diary* nantinya. Pikirkan ide-ide untuk diary yang akan ditulis, seperti apakah *diary* nantinya hanya bercerita tentang diri sendiri atau mengenai topik spesifik lainnya. Setelah mengetahui tema yang diinginkan, maka langkah-langkah dalam menulis berikut akan lebih mudah:

1. Tentukan tanggal entri

   Karena *diary* akan digunakan dan disimpan dalam waktu yang lama, setiap entri *diary* sebaiknya diberi penanggalan berdasarkan waktu ketika sebuah entri ditulis.

2. Pilih sebuah topik

   Seperti bab-bab dalam buku, entri dalam *diary* dapat memiliki jenis-jenis tema tertentu atau tujuan. Apakah itu mengenai apa aktivitas yang dilakukan selama satu hari, suatu peristiwa yang akan datang, ataupun peristiwa yang telah berlalu. Coba dan tetap berfokus ketika menulisnya.

3. Tulislah secara alami

   Penulis *diary* tidak perlu menulis *diary* untuk mengesankan siapapun. Entri *diary* ditulis hanya untuk diri sendiri, sehingga menulis dengan santai ialah hal yang diperlukan.

4. Ceritakan secara jujur

   Saat menulis *diary*, hal yang paling buruk ialah berbohong kepada diri sendiri, sehingga lebih baik untuk mencoba menulis *diary* sejujur mungkin.

5. Ceritakan dengan ramah

   Saat menulis, cobalah untuk menulis seperti menulis untuk seorang teman. Menuliskan mengenai hal seperti apa yang akan diceritakan kepada seorang teman baik.

6. Teruslah menulis

   Jika menemui hambatan saat menulis dan merasa tidak mengetahui kalimat apa yang perlu ditulis, teruslah tetap menulis. Penulis *diary* juga dapat menuliskan barisan-barisan pikirannya dengan memikirkan apa yang membuat hambatan tersebut muncul. Selain itu, pikirkan juga apa yang akan kamu tulis dan bagaimana hal tersebut dapat mempengaruhi entri *diary* dan tuliskan proses pemikiran yang sedang berlangsung tersebut.

7. Baca kembali entri tersebut

   Setelah selesai menulis sebuah entri, baca kembali entri tersebut dan buat sebuah daftar mengenai hal apa yang diharapkan untuk difokuskan pada entri selanjutnya. Refleksikan kembali hal apa yang telah ditulis. Ide-ide baru yang muncul saat proses membaca kembali tersebut juga dapat dituliskan ke dalam entri.

8. Jadikan sebagai suatu kebiasaan

   Semakin sering seseorang menulis entri dalam *diary*, semakin seseorang tersebut dapat menikmati untuk menulisnya dan juga semakin banyak hal yang dapat dipelajari dalam menulis tersebut. Tentukan waktu tertentu dalam satu hari yang digunakan untuk menulis *diary* dan pastikan terjadwal. [10]

2.3 Metodologi FAST RAD

2.3.1 FAST

FAST (*Framework for the Application of Systems Thinking*/Kerangka untuk Penerapan Pemikiran Sistem) adalah sebuah metodologi hipotetis yang digunakan untuk mendemonstrasikan pengembangan sistem perwakilan.

FAST memiliki fase-fase seperti kebanyakan metodologi dan tiap fase menghasilkan produk jadi yang dilewatkan ke fase berikutnya. Adapun fase-fase dari metodologi FAST ialah:

1. Definisi Lingkup

   Tujuan dari fase ini ialah untuk menjawab pertanyaan seberapa pantas suatu proyek diperhatikan dan mengasumsikan bahwa masalah tersebut memang pantas diperhatikan. Definisi lingkup menentukan ukuran dan batas-batas proyek, visi proyek, semua batasan, partisipan proyek yang dibutuhkan, serta anggaran dan jadwal.

2. Analisis Masalah

   Fase analisis masalah mempelajari sistem yang ada dan menganalisa penemuan-penemuan untuk menyediakan tim proyek dengan pemahaman yang lebih mendalam akan masalah-masalah yang memicu proyek. Dalam fase ini, para pengguna sistem mulai secara aktif dilibatkan. Prasyarat untuk fase analisis masalah adalah lingkup dan pernyataan masalah yang telah disetujui dalam fase definisi lingkup. Produk jadi dari analisis masalah adalah satu set tujuan perbaikan sistem yang diperoleh dari pemahaman menyeluruh terhadap masalah-masalah bisnis.

3. Analisis Persyaratan

   Fase analisis persyaratan mendefinisikan dan memprioritaskan persyaratan-persyaratan bisnis. Analis sistem mendekati para pengguna untuk mengetahui apa yang diperlukan dari sistem yang baru. Tujuan perbaikan sistem dari analisis masalah adalah prasyarat pada fase ini. Produk jadinya adalah pernyataan persyaratan bisnis termasuk persyaratan data bisnis. Untuk menghasilkan pernyataan persyaratan bisnis, analis sistem bekerja secara dekat dengan pengguna sistem untuk mengidentifikasi kebutuhan dan prioritas. Informasi ini dikumpulkan dengan cara wawancara, kuesioner, dan pertemuan terfasilitasi.

4. Desain Logis

   Persyaratan bisnis dalam bentuk kata-kata diterjemahkan ke dalam gambar yang disebut model sistem untuk memvalidasi persyaratan-persyaratan untuk kelengkapan dan konsistensi. Desain logis yang dapat digambarkan ialah model data logis yang menggambarkan persyaratan-persyaratan data dan informasi, model-model proses yang menggambarkan persyaratan-persyaratan proses bisnis, dan model antarmuka logis yang menggambarkan persyaratan antarmuka bisnis dan sistem. Prasyarat desain logis adalah pernyataan persyaratan bisnis dari fase sebelumnya. Dengan kata lain, pernyataan persyaratan bisnis diidentifikasi dan didokumentasikan untuk kemudian dimodelkan. Produk jadi fase ini ialah model dan spesifikasi sistem logis.

5. Analisis Keputusan

   Dengan diberikan persyaratan-persyaratan bisnis dan model-model sistem logis, tujuan fase ini ialah mengidentifikasikan solusi teknis calon, menganalisis solusi-

solusi calon untuk kepraktisan, dan merekomendasikan sistem calon sebagai target untuk didesain. Produk jadi kunci fase analisis keputusan adalah proposal sistem. Proposal ini dapat tertulis atau dipresentasikan secara lisan.

6. Desain dan Integrasi Fisik

   Dengan adanya persetujuan proposal sistem dari fase analisis keputusan, setelah itu dapat dilakukan desain sistem baru. Tujuan fase ini adalah mentransformasi persyaratan-persyaratan bisnis (diwakilkan sebagian oleh model sistem logis) ke dalam spesifikasi desain fisik yang akan memandu konstruksi sistem. Desain fisik mengurus rincian lebih besar mengenai teknologi yang akan digunakan dalam sistem baru. Desain fisik adalah kebalikan dari desain logis, dimana desain fisik mewakili solusi teknis spesifik sedangkan desain logis berurusan secara eksklusif dengan persyaratan-persyaratan bisnis yang terpisah dari semua solusi teknis. Produk jadi fase desain dan integrasi fisik adalah kombinasi model dan spesifikasi desain fisik, *prototipe* desain, dan desain ulang proses bisnis

7. Konstruksi dan Pengujian

   Setelah adanya tingkat model dan spesifikasi desain fisik (prototipe desain), dapat dilakukan konstruksi dan pengujian komponen-komponen sistem untuk desain tersebut. Produk jadi fase ini ialah sistem fungsional yang siap untuk diimplementasikan, Dua tujuan fase konstruksi dan pengujian adalah membangun dan menguji sebuah sistem yang memenuhi persyaratan bisnis dan spesifikasi desain fisik, dan mengimplementasikan antarmuka sistem. Sebagai tambahan, dokumentasi final akan dikembangkan sebagai persiapan pelatihan dan operasi sistem.

8. Instalasi dan Pengiriman

   Tujuan fase ini ialah mengirimkan sistem ke dalam operasi (produksi). Sistem fungsional dari fase sebelumnya adalah input kunci pada fase instalasi dan pengiriman. Produk jadinya adalah sistem operasional. Pembangun sistem meng-*install* sistem dari lingkungan pengembangan ke dalam lingkungan produksi. Para analis sistem harus melatih para pengguna sistem, menuliskan berbagai macam manual pengguna dan kontrol produksi, mengkonversi file dan basis data yang ada menjadi basis data baru, dan melakukan pengujian sistem final. Masalah yang muncul di fase ini dapat memicu pengulangan dalam fase-fase sebelumnya. Para

pengguna sistem menyediakan umpan balik yang berkesinambungan saat masalah dan isu baru muncul. [11]

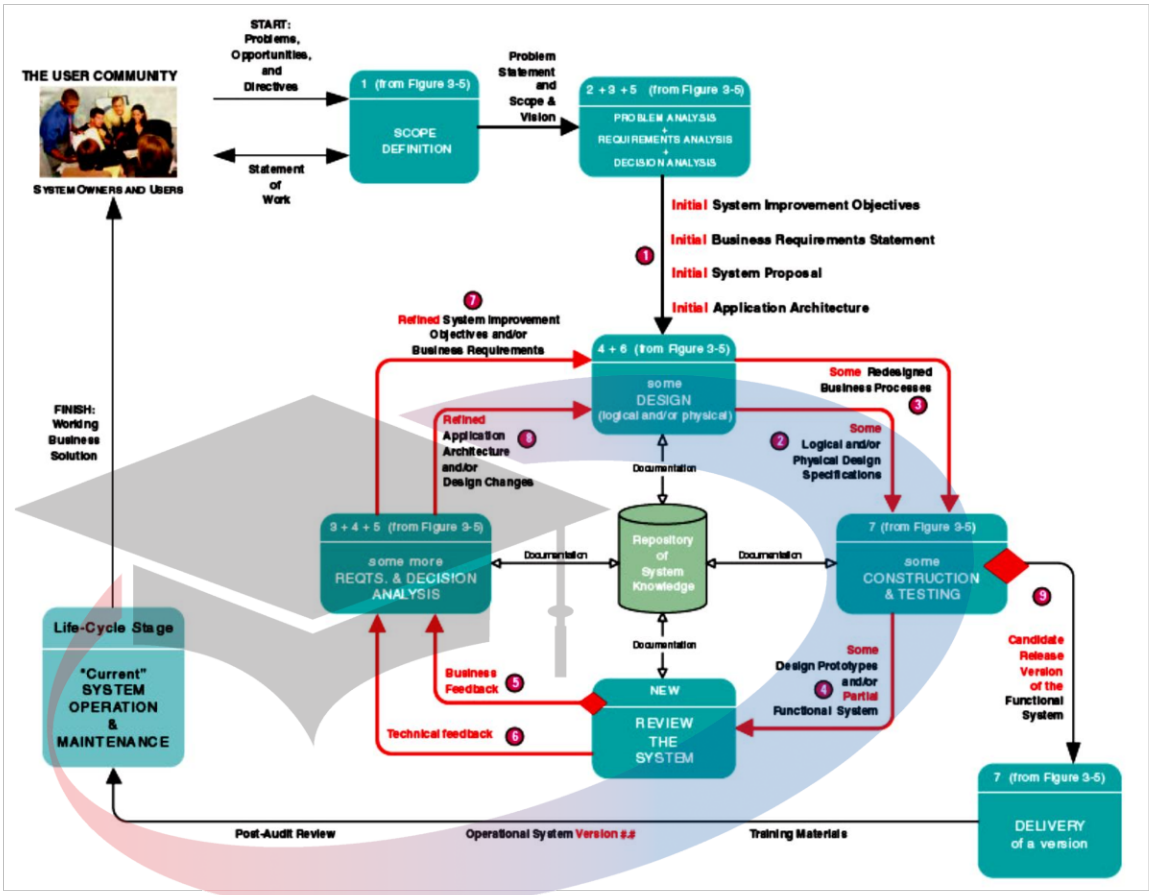
2.3.2 Rapid Application Development

Rapid Application Development (RAD) adalah sebuah strategi pengembangan sistem yang menekankan kecepatan pengembangan melalui keterlibatan pengguna yang ekstensif dalam konstruksi, cepat, berulang, dan bertambah serangkaian *prototipe* bekerja sebuah sistem yang pada akhirnya berkembang ke dalam sistem final atau sebuah versi.

Gagasan dasar RAD adalah:

1. Lebih aktif melibatkan para pengguna sistem dalam aktivitas analisis, desain, konstruksi.
2. Mengorganisasikan pengembangan sistem ke dalam rangkaian seminar yang intensif dan terfokus bersama dengan para pemilik, pengguna, analis, desainer, dan pengembang sistem.
3. Mengakselarasi fase-fase analisis dan desain persyaratan melalui pendekatan konstruksi berulang.
4. Memperpendek waktu yang diperlukan sebelum para pengguna mulai melihat sebuah sistem yang bekerja.

Dalam RAD, *prototipe* digunakan agar pengguna dapat melihat bagaimana desain yang diusulkan dari sebuah sistem yang akan dibangun.Sebuah *prototipe* kemudian akhirnya akan berkembang menjadi sistem informasi final. Tahapan RAD untuk FAST diilustrasikan pada gambar 2.1. [4]



Gambar 2.1 Fase Rapid Application Development (RAD)

1. Fase analisis masalah awal, analisis persyaratan, dan analisis keputusan dikonsolidasikan dan diakselerasi untuk mengurangi waktu dalam mengembangkan aplikasi dan sistem. Produk jadi dikatakan sebagai "awal" karena diharapkan untuk berubah semasa proyek dijalankan.

   Setelah analisis dilakukan, pendekatan RAD berulang melalui siklus fase-fase yang dideskripsikan sebagai "lakukan desain, konstruksi, analisis, desain lagi, konstruksi lagi, analisis lagi" dan seterusnya, sampai sebuah versi sistem final siap untuk diimplementasikan. Tiap pengulangan hanya menekankan fungsionalitas untuk diselesaikan dalam beberapa minggu.
2. Spesifikasi desain fisik dan logis biasanya secara signifikan dipendekkan dan diakselerasi. Pada tiap pengulangan dalam siklus, hanya beberapa spesifikasi desain yang akan dipertimbangkan. Saat beberapa model sistem digambar, akan secara selektif dipilih dan penekanannya berlanjut pada pengembangan cepat.

Asumsinya adalah bahwa *error* dapat ditemukan dan diperbaiki dalam pengulangan berikutnya.

3. Dalam beberapa kasus, pengulangan beberapa proses bisnis mungkin harus didesain ulang untuk merefleksikan aplikasi perangkat lunak yang berkembang.
4. Pada tiap pengulangan dalam siklus, sebagian elemen *prototipe* desain atau sebagian sistem fungsional dikonstruksi dan diuji. Pada akhirnya, aplikasi yang telah selesai akan berasal dari pengulangan final dalam siklus.
5. Setelah tipe *prototipe* atau subsistem fungsional sebagian dikonstruksi dan diuji, para pengguna sistem diberi kesempatan untuk mengalami bekerja degan *prototipe* tersebut. Harapannya adalah para pengguna akan mengklarifikasi persyaratan, mengidentifikasi persyaratan baru, dan menyediakan umpan balik bisnis pada desain untuk pengulangan selanjutnya dalam siklus RAD.
6. Setelah tipe *prototipe* atau subsistem yang berfungsi dikonstruksi dan diuji, para analis dan desainer sistem akan meninjau kembali arsitektur dan desain aplikasi untuk menyediakan umpan balik teknis dan pengarahan untuk pengulangan selanjutnya dalam siklus RAD.
7. Berdasarkan umpan balik tersebut, analis sistem akan mengidentifikasi tujuan perbaikan sistem disempurnakan dan/atau persyaratan bisnis. Analis cenderung fokus merevisi atau memperluas tujuan dan persyaratan serta mengidentifikasi kepedulian pengguna pada desain.
8. Berdasarkan umpan balik tersebut, analis sistem dan desainer sistem akan mengidentifikasi arsitektur aplikasi disempurnakan dan/atau perubahan desain.
9. Terakhir, sistem atau versi sistem tersebut akan dianggap bernilai untuk diimplementasikan. Versi rilis calon sistem fungsional ini adalah sistem yang diuji dan ditempatkan dalam operasi. Versi selanjutnya dari sistem ini mungkin terus berulang dalam siklus RAD.

Beberapa keunggulan dalam menggunakan metodologi RAD yaitu:

1. Berguna untuk proyek yang kebutuhan penggunanya tidak pasti dan tidak tepat.
2. Mendorong pengguna aktif dan partisipasi manajemen, dimana meningkatkan antusiasme pengguna pada akhir proyek.

3. Proyek-proyek memiliki visibilitas dan dukungan lebih tinggi karena keterlibatan pengguna yang ekstensif selama proses.
4. Para pengguna dan manajemen melihat solusi-solusi yang berbasis perangkat lunak dan bekerja lebih cepat daripada pengembangan *model-driven*.
5. *Error* dan penghilangan cenderung untuk dideteksi lebih awal dalam *prototipe* daripada dalam model sistem.
6. Pengujian dan pelatihan adalah produk tambahan alami dari pendekatan *prototyping* yang mendasar.
7. Pendekatan berulang adalah proses yang lebih alami karena perubahan adalah faktor yang diharapkan selama pengembangan.

Sedangkan, kelemahan dalam menggunakan metodologi RAD yaitu

1. Adanya pendapat bahwa RAD mendorong mentalitas "mengkode, mengimplementasi, dan memperbaiki" yang meningkatkan biaya seumur hidup yang diperlukan untuk mengoperasikan, mendukung, dan merawat sistem.
2. *Prototipe-prototipe* RAD mungkin dapat memecahkan masalah yang salah karena analisis masalah disingkat atau diabaikan.
3. *Prototipe* berbasis RAD mungkin membuat para analis merasa tidak percaya diri untuk mempertimbangkan alternatif-alternatif teknis lain yang lebih bernilai.
4. Ada kalanya sebuah *prototipe* lebih baik dibuang, tetapi para *stakeholder* sering enggan melakukannya karena mereka menganggapnya sebagai hilangnya waktu dan usaha dalam produk saat ini.
5. Penekanan pada kecepatan dapat berdampak buruk terhadap kualitas yang disebabkan jalan-jalan pintas yang disarankan dengan buruk dalam metodologi tersebut. [4]

2.4 Teknik Perancangan Sistem

2.4.1 *Use-Case Diagram*

Diagram *use-case* merupakan diagram yang menggambarkan interaksi antara sistem dengan sistem eksternal dan pengguna, dengan kata lain, menggambarkan siapa yang akan menggunakan sistem dan dengan cara apa pengguna mengharapkan untuk berinteraksi dengan sistem.

Pemodelan *use-case* mengidentifikasi dan menggambarkan fungsi sistem dengan menggunakan alat yang disebut *use-case*. *Use-case* menggambarkan fungsi-fungsi sistem dari sudut pandang pengguna eksternal dan dalam sebuah cara dan terminologi yang mereka pahami. *Use-case* ialah hasil penyusunan kembali lingkup fungsionalitas sistem menjadi banyak pernyataan fungsionalitas sistem yang lebih kecil.

Pemodelan *use-case* memberikan manfaat sebagai berikut:

1. Menyediakan *tool* untuk meng-*capture* persyaratan fungsional
2. Membantu menyususn ulang lingkup sistem menjadi bagian-bagian yang lebih dapat dikelola.
3. Menyediakan alat komunikasi dengan para pengguna dan *stakeholder* yang berhubungan dengan fungsionalitas sistem. Use-case menyajikan bahasa umum yang dapat dipahami oleh berbagai macam *stakeholder*.
4. Memberikan cara bagaimana mengidentifikasi, menetapkan, melacak, mengontrol, dan mengelola kegiatan pengembangan sistem, terutama pengembangan *incremental* dan iteratif.
5. Menyajikan panduan untuk mengestimasi lingkup, usaha, dan jadwal proyek.
6. Menyajikan garis pokok pengujian, khususnya menentukan rencana tes dan *test case*.
7. Menyajikan garis pokok bagi *help system* dan manual pengguna, dan juga dokumentasi pengembangan sistem.
8. Menyajikan *tool* untuk melacak persyaratan.
9. Menyajikan titik mulai/awal untuk identifikasi objek data atau entitas.
10. Menyajikan spesifikasi fungsional untuk mendesain antarmuka pengguna dan sistem.
11. Menyajikan alat untuk menentukan persyaratan akses *database* dalam hal menambah, mengubah, menghapus, dan membaca.
12. Menyajikan kerangka kerja untuk mengarahkan proyek pengembangan sistem.

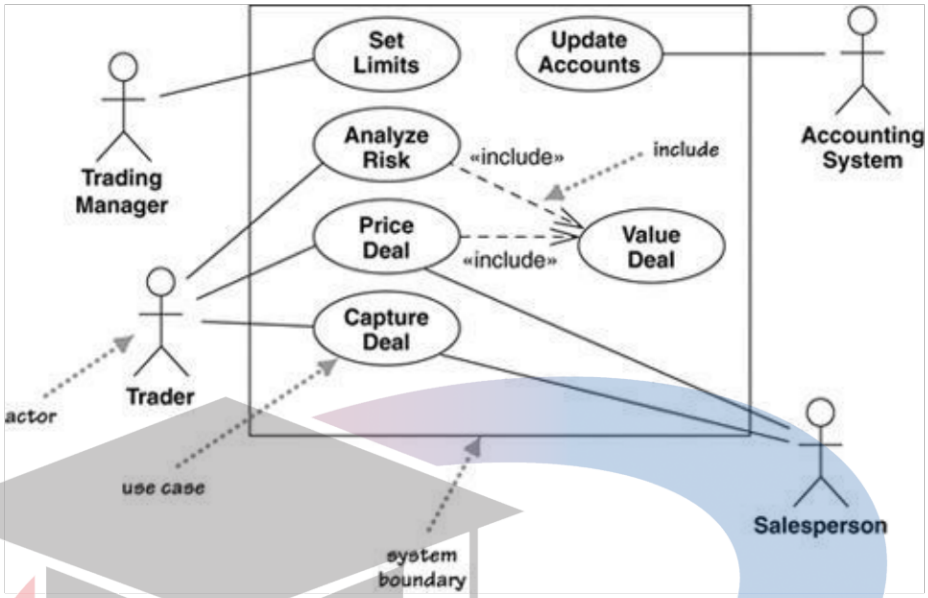
Elemen-elemen yang digunakan dalam diagram *use-case* adalah: [4]

Tabel 2.1 Elemen dalam *Use-case* Diagram

Nama elemen	**Simbol**	**Keterangan**

Aktor/Pelaku		Berupa segala sesuatu yang perlu berinteraksi dengan sistem untuk pertukaran informasi. Aktor tidak selalu menggambarkan individu ataupun nama posisi, melainkan dapat berupa perusahaan, sistem informasi, alat eksternal, atau konsep waktu.
Use-case		Merepresentasikan satu tujuan tunggal dari sistem dan menggambarkan rangkaian dan interaksi pengguna untuk mencapai tujuan
Association		Merupakan hubungan antara seorang aktor dengan satu use-case di mana terjadi interaksi di antara keduanya
Extend		Hubungan antara *extension* use-case, yaitu use-case yang dihasilkan dari use-case yang lebih kompleks, dengan use-case yang diperluas
Include/Use		Hubungan antara *abstract* use-case, yaitu use-case yang menggambarkan fungsionalitas yang umum, dengan use-case yang menggunakannya.
Depends on		Merupakan hubungan antara use-case yang memiliki ketergantungan terhadap use-case yang lain.
Generalization		Merupakan pembagian atribut yang umum ke beberapa tipe entitas dikelompokkan dalam entitasnya sendiri

Berikut ini adalah contoh dari diagram *use case*.



Gambar 2.2 Contoh Diagram *Use Case*

Gambar 2.2 merupakan contoh sederhana dari penggunaan elemen *use case* pada sebuah sistem dalam perusahaan, dimana terdapat 4 aktor yang masing-masing menginisiasi *use case* atau fungsi dalam sistem yang sesuai dengan peran aktor tersebut. Terdapat juga *use case* yang memiliki hubungan *include* atau yang termasuk ke dalam *use case abstract* yaitu *use case Value Deal.*

2.4.2 *Sequence Diagram*

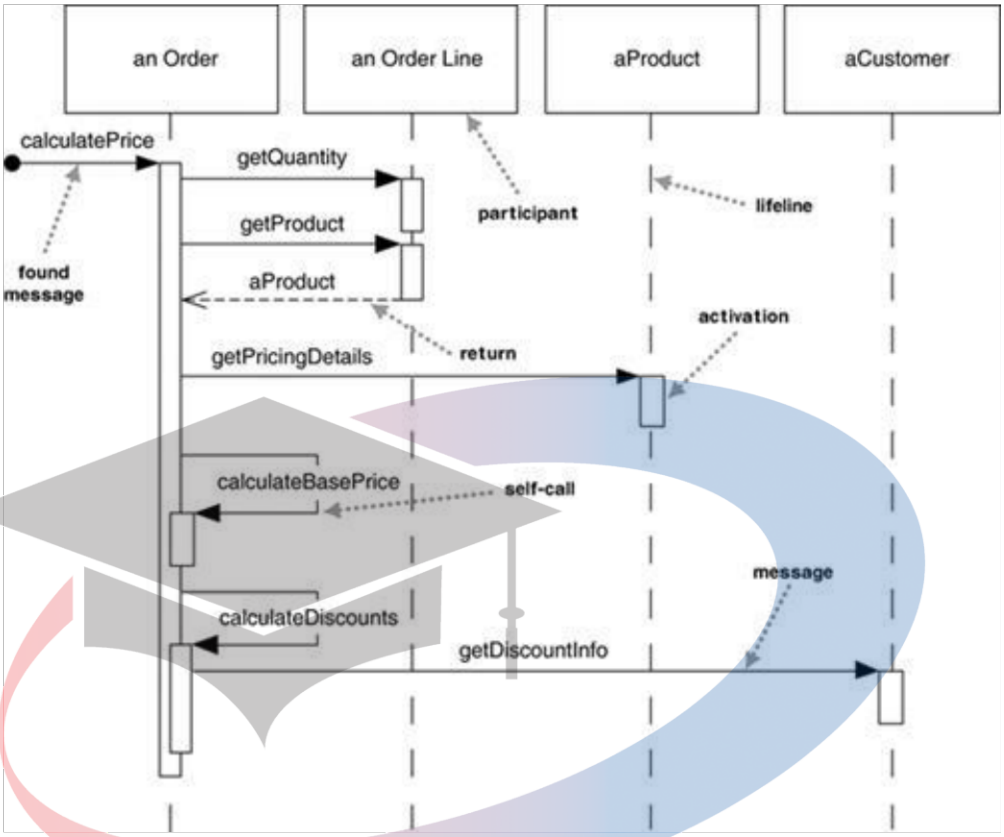
Sebuah diagram *sequence* menggambarkan rangkaian interaksi antara kelas atau objek selama operasi. Diagram *sequence* sering digunakan untuk menggambarkan pemrosesan yang dideskripsikan dalam skenario use case. Dalam prakteknya, diagram *sequence* berasal dari analisis use case dan digunakan dalam perancangan sistem untuk memperoleh interaksi, hubungan, dan metode objek dalam sebuah sistem. Diagram *sequence* digunakan untuk menampilkan pola keseluruhan dari aktivitas atau interaksi dalam sebuah use case. [9]

Tabel 2.2 Komponen dalam Diagram *Sequence*

Komponen	**Simbol**	**Keterangan**
Aktor		Aktor yang berinteraksi dengan user interface ditampilkan menggunakan simbol aktor use case.

Interface class	<<interface>> :Order Window	Melambangkan kode class user interface, ditandai dengan <<interface>>.
Controller class	<<controller>> :Place New Order	Setiap use case akan memiliki satu/lebih class controller, ditandai dengan <<controller>>
Entity class	:Member Order	Melambangkan setiap entitas yang perlu berkolaborasi dalam urutan langkah-langkah.
Message		Melambangkan input pesan yang dikirim ke class.
Activation Bar		Ditempatkan pada garis alir (*lifeline*) dan melambangkan periode waktu selama setiap perintah objek berlangsung
Return Message		Melambangkan respon terhadap pesan yang dikirim.
Self-call		Sebuah objek dapat memanggil method-nya sendiri
Frame		Dapat menyertai satu atau lebih pesan untuk membagi sebuah fragmen *sequence*. Operator *loop* pada frame menandakan bahwa controller perlu berulang kali mengeksekusi semua item.

Berikut ini adalah contoh dari diagram *sequence*.



Gambar 2.3 Contoh Diagram *Sequence*

Diagram *sequence* pada Gambar 2.3 menampilkan implementasi dari skenario kalkulasi harga pesanan. Entitas *anOrder* menggambarkan sebuah pesanan yang diterima dan langkah selanjutnya setelah menerima pesanan ialah memanggil perintah untuk mengkalkulasi harga pesanan. Untuk melakukannya, *anOrder* membutuhkan data mengenai daftar item yang ada pada pesanan dan menentukan harganya, yang mana berdasarkan harga yang telah ditentukan pada produk dalam antrian pesanan. Setelah melakuka kalkulasi pada semua daftar item pesanan, maka *anOrder* harus menghitung diskon keseluruhan yang berdasarkan aturan yang mengikat pada entitas *Customer*.

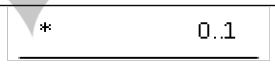
2.4.3 *Class Diagram*

Class diagram mendeskripsikan jenis-jenis objek dalam sistem dan berbagai macam hubungan statis yang terdapat di antara mereka. Selain itu juga menunjukkan properti dan operasi sebuah *class* dan batasan-batasan yang terdapat dalam hubungan-

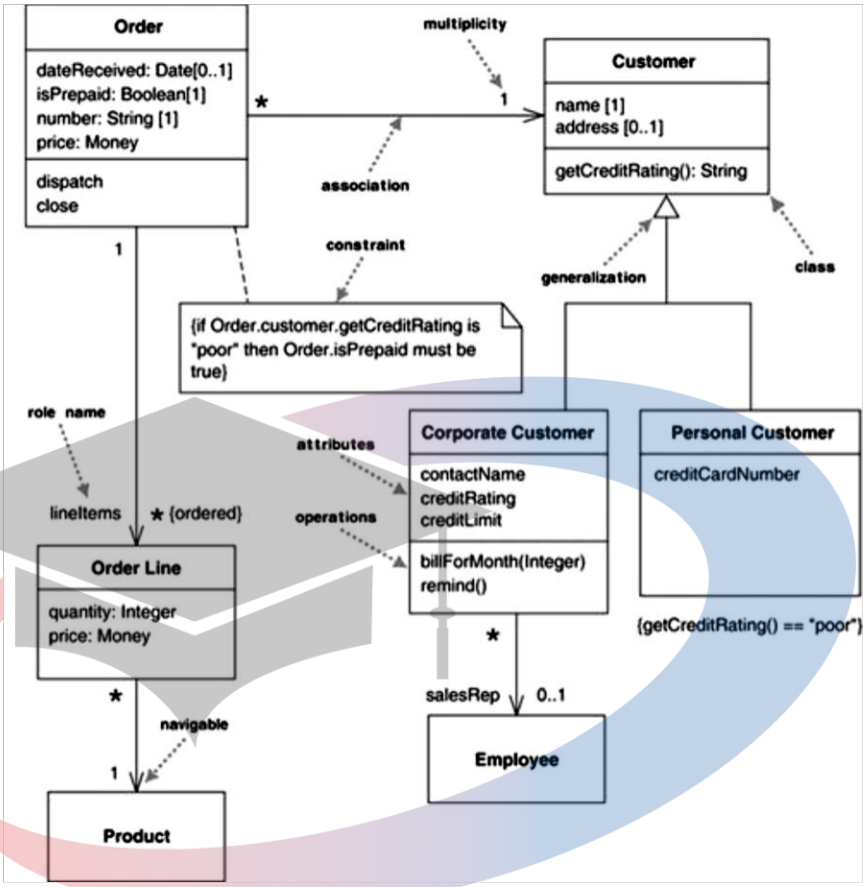
hubungan objek tersebut. UML menggunakan istilah fitur sebagai istilah yang umum meliputi properti dan operasi sebuah *class*. [10]

Adapun elemen-elemen yang terdapat dalam *class* diagram dapat dilihat dalam tabel di bawah ini: [11]

Tabel 2.3 Elemen dalam Diagram *Class*

Nama elemen	Simbol	Keterangan
Class		Sebuah *class* mewakili sebuah konsep yang relevan dari domain, sekelompok orang, objek, atau ide yang digambarkan dalam sistem
Atribut		Atribut sebuah *class* mewakili sebuah karakteristik dari sebuah *class* dalam sistem
Association		Merupakan hubungan antara satu class dengan *class* yang lain
Multiplicity		Merupakan indikasi tentang berapa banyak objek yang akan terlibat dalam sebuah asosiasi
Aggregation		Merupakan kasus spesial dari sebuah asosiasi, yang berarti "terdiri dari"
Dependency		Hubungan yang muncul antara dua elemen jika perubahan definisi sebuah elemen dapat menyebabkan perubahan pada elemen lainnya.
Generalization		Merupakan hubungan antara *class* yang umum dengan *class* yang lebih spesifik.
Catatan		Merupakan komentar dalam diagram, dapat berdiri sendiri atau dihubungkan dengan garis hubung dengan elemen yang dikomentarinya

Berikut ini adalah contoh dari diagram *class*.



Gambar 2.4 Contoh Diagram *Class*

Gambar 2.4 menggambarkan model *class* sederhana dari sistem pembelian dalam suatu perusahaan. Kotak-kotak dalam diagram ialah *class*, yang dibagi kedalam tiga bagian, yaitu nama *class*, atribut *class*, dan operasinya. Dalam diagram tersebut digunakan dua jenis hubungan antara *class*, yaitu asosiasi dan generalisasi.

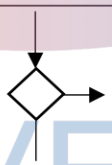
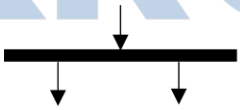
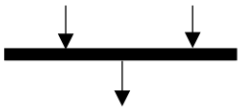
2.4.4 Activity Diagram

Diagram *activity* digunakan untuk menggambarkan secara grafis aliran proses bisnis, langkah-langkah dalam sebuah use case, atau logika *object behavior* (*method*). Diagram ini memodelkan langkah proses atau aktivitas dari sistem. Diagram *activity* memiliki kesamaan dengan *flowchart* dalam hal menggambarkan urutan alur dari aktivitas dalam proses bisnis ataupun sebuah *use case*. Perbedaan diagram *activity* dengan *flowchart* terletak dalam bagaimana diagram *activity* menyediakan sebuah mekanisme untuk menggambarkan aktivitas yang terjadi secara paralel. Oleh karenanya, diagram ini sangat berguna untuk memodelkan tindakan yang akan

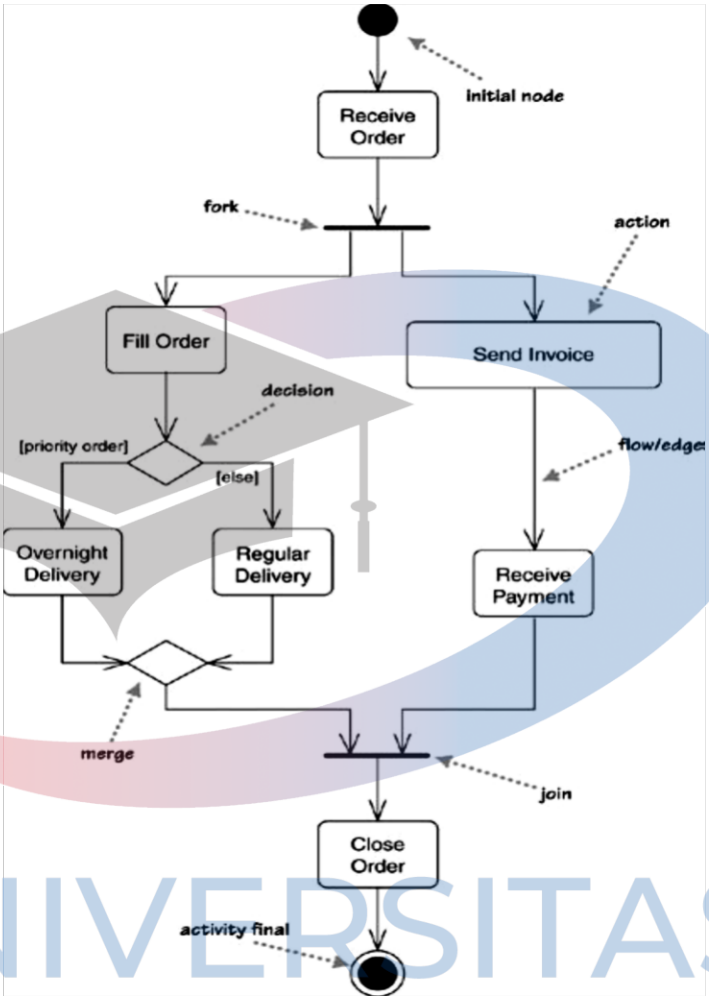
dilakukan saat sebuah operasi sedang berjalan sekaligus hasil dari tindakan tersebut. Diagram *activity* bersifat fleksibel sehingga dapat digunakan selama analisis dan desain. Setidaknya satu diagram *activity* dapat dikonstruksi untuk setiap use case. Lebih dari satu diagram *activity* dapat dikonstruksi jika use case memiliki logika yang rumit. Analis sistem menggunakan activity diagram untuk mengetahui lebih baik alur dan perurutan dari langkah use case.

Berikut ialah penjelasan terhadap notasi-notasi yang digunakan dalam diagram *activity*: [2]

Tabel 2.4 Elemen dalam Diagram *Activity*

Nama notasi	Simbol	Keterangan
Initial node		Melambangkan awal mulai proses
Actions		Melambangkan langkah individual. Urutan dalam tindakan membentuk jumlah aktivitas yang diperlihatkan dalam diagram
Flow		Merupakan proses dari aktivitas yang satu ke aktivitas yang lain
Decision		Terdapat satu aliran masuk dan dua/lebih aliran keluar. Arah aliran keluar ditandai untuk mengindikasi kondisi.
Merge		Terdapat dua/lebih aliran masuk dan satu aliran keluar. Menggabungkan aliran yang sebelumnya dipisahkan oleh *decision*. Pemrosesan berlanjut dengan aliran apapun yang masuk ke merge.
Fork		Terdapat satu aliran masuk dan dua/lebih aliran keluar. Aksi dalam aliran paralel dibawah simbol *fork* dapat terjadi dalam urutan apapun atau secara bersamaan.
Join		Terdapat dua/lebih aliran masuk dan satu aliran keluar, merupakan akhir dari pemrosesan yang bersamaan. Semua aksi yang masuk ke *join* harus diselesaikan sebelum sebelum pemrosesan berlanjut
Activity final		Melambangkan akhir dari proses.

Berikut ini adalah contoh dari diagram *acivity*.



Gambar 2.5 Contoh Diagram *Activity*

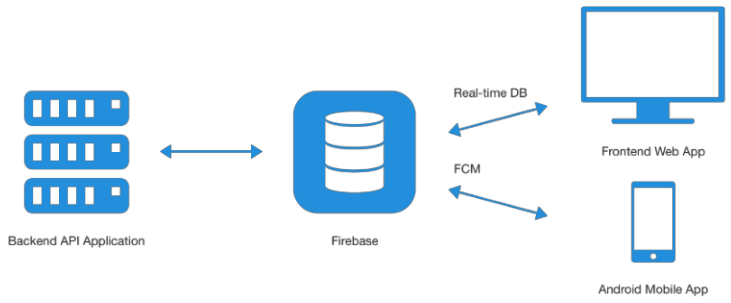
Gambar 2.5 merupakan contoh sederhana dari sebuah diagram activity yang menggambarkan langkah-langkah serta proses yang terdapat dalam aktivitas penerimaan pesanan (*receive order*). Setelah menerima pesanan, terdapat dua aksi berikut yang paralel yaitu membuat pesanan (*fill order*) dan mengirim faktur (*send invoice*), yang berarti urutan diantara keduanya tidak berhubungan. Urutan aksi membuat pesanan, mengirim faktur, pengiriman, dan menerima pembayaran dapat dilakukan ataupun memulai aksi dengan mengirim faktur terlebih dahulu, menerima pembayaran, pembuatan pesanan, dan kemudian melakukan pengiriman juga dapat diterima.

2.5 *Firebase Realtime Database*

Firebase Realtime Database adalah layanan basis data berbasis cloud-hosted. Data yang disimpan ialah sebagai JSON (*JavaScript Object Notation*) dan akan disinkronisasi secara realtime ke setiap aplikasi klien yang terkoneksi. Cara kerja *Realtime Database* ini ialah dengan mengizinkan akses yang aman ke *database* langsung dari kode sisi klien. Data disimpan di *drive* lokal, bahkan saat *offline* sekalipun, perisitiwa realtime terus berlangsung, sehingga pengguna akhir akan merasakan pengalaman yang responsif. Ketika koneksi perangkat tersambung, *Realtime Database* akan menyinkronkan perubahan data lokal dengan *update* jarak jauh yang terjadi selama klien *offline*, sehingga setiap perbedaan akan otomatis digabungkan.

Realtime Database menyediakan bahasa aturan berbasis ekspresi yang fleksibel dengan sintaks mirip *JavaScript*, disebut *Firebase Realtime Database Security Rules*, agar pengembang dapat dengan mudah menentukan cara strukturisasi data, penyusunan indeks, serta kapan data dapat dibaca dan ditulisi. Dan dikombinasikan dengan layanan Autentikasi Firebase, pengembang dapat menentukan siapa yang boleh mengakses data dan melindungi informasi pribadi pengguna dari akses tanpa izin.

Firebase Realtime Database merupakan database *NoSQL* sehingga memiliki pengoptimalan dan fungsionalitas yang berbeda dengan *database* relasional. API *Realtime Database* dirancang agar hanya mengizinkan operasi yang dapat dijalankan dengan cepat, sehingga data dapat disinkronisasi secara *realtime* antara jutaan pengguna tanpa mengorbankan kemampuan respons. [14]



Gambar 2.6 Cara Kerja *Firebase Realtime Database*