BAB 2

TINJAUAN PUSTAKA

2.1. Landasan Teori

2.1.1. Perancangan

Menurut Pressman (2010:215) perancangan adalah membuat sebuah gambaran atau model dari sebuah perangkat lunak dengan menyediakan rincian arsitektur dari sebuah perangkat lunak, struktur data, tampilan, dan komponen-komponen yang dibutuhkan untuk mengimplementasikan sistem. Perancangan memiliki peran penting karena model ini dapat dinilai terlebih dahulu kualitasnya dan dikembangkan sebelum sistem dibangun.

2.1.2. Implementasi Sistem

Implementasi sistem adalah suatu proses untuk menempatkan dan menerapkan informasi baru kedalam operasi sistem. (*Whitten, Bentley & Barlow* : 1993)

2.1.3. Sistem

Menurut Bonnie Soeharman dan Marion Pinontoan (2008:3) dalam bukunya terbitan Elex Media Komputindo di Jakarta yang berjudul Designing Information System, sistem dapat diartikan sebagai serangkaian komponen-komponen yang saling berinteraksi dan bekerja sama untuk mencapai tujuan tertentu.

2.1.4. *Website*

Menurut Sibero (2011b:11) *website* adalah suatu sistem yang berkaitan dengan dokumen yang digunakan sebagai media untuk menampilkan tulisan, gambar, dan lainnya di jaringan *internet.*

2.1.5. Internship

Internship merupakan suatu proses pembelajaran yang mengandung unsur belajar sambil bekerja. Mahasiswa belajar sebagai pemagang akan membiasakan diri mengikuti proses pekerjaan yang diikuti oleh pemagang (pendidik).

Menurut Sudjana, D. (2000) mengemukakan lebih lanjut bahwa melalui magang seseorang yang memiliki pengalaman tertentu menyampaikan pengetahuan dan keterampilan yang telah ia miliki kepada orang lain yang belum berpengalaman dan lebih dahulu memiliki pengalaman dan keahlian tertentu sehingga pemagang memiliki pengalaman dan keahlian dalam bekerja.

2.1.6. *Software*

Menurut Roger S. Pressman (2001:6), *software* atau disebut juga perangkat lunak merupakan (1) sekumpulan perintah (program komputer) yang apabila dieksekusi dapat memberikan fungsi dan juga kinerja yang diinginkan oleh pengguna, (2) struktur data yang memungkinkan program untuk memanipulasi informasi, dan (3) dokumen yang mendeskripsikan operasi dan kegunaan dari program.

Untuk lebih memahami mengenai *software*, sangat penting bagi kita untuk memeriksa karakteristik *software*, antara lain:

- *Software* dikembangkan atau dibangun, bukan diproduksi
- *Software* tidak akan "rusak" meski berkali-kali dipakai
- Meski industri tengah bergerak menuju perakita berbasis komponen, software tetap dibuat sesuai pesanan

2.1.7. *Software Engineer*(belom ada daftar pustakanya)

Pressman & Bruce (2014:14) *software engineering* mencakup sebuah proses, kumpulan metode (praktek) dan berbagai *tools* yang memungkinkan profesional untuk membuat sebuah *software computer* berkualitas tinggi.

2.1.8. *Database*

Menurut Connolly & Begg (2015:15), database adalah sekumpulan data yang saling berhubungan secara logis, dan sebuah deskripsi dari data tersebut, yang di desain untuk memenuhi informasi yang dibutuhkan oleh suatu organisasi.

2.1.9. *Database Management System* (DBMS)

Menurut Connolly & Begg (2005:16), DBMS adalah sistem *software* yang memungkinkan pengguna untuk mendefinisikan, membuat, mengurus, dan mengontrol akses ke database.

2.1.10. *Internet*

Menurut Robert J. Verzello (1998:23), internet adalah suatu jaringan komputer global yang terbentuk dari jaringan-jaringan komputer *local* dan *regional*, dengan adanya jaringan ini memungkinkan komunikasi data antar komputer-komputer yang terhubung ke jaringan tersebut.

2.1.11. HTML

Menurut Williams & Sawyer (2007:68), *Hypertext Markup Language* (HTML) adalah sekumpulan instruksi khusus yang digunakan untuk menetapkan struktur dokumen , format, dan link-link untuk menuju ke dokumen multimedia dalam web.

2.1.12. *World Wide Web*

Menurut Rainer & Cegielski (2011:522), *World Wide Web* adalah sistem yang terdiri dari standar yang diterima secara universal untuk menyimpan, mendapatkan, memformat, dan menampilkan informasi melalui arsitektur *client*/*server*. WWW menangani semua tipe informasi digital, termasuk diantaranya: teks, *hypermedia*, gambar, dan suara.

2.2. Penelitian Sebelumnya

2.2.1. Analisa Kekurangan pada aplikasi lowongan *internship* pada www.anyintern.com

Penelitian Perancangan dan Implementasi Sistem Pencarian Lowongan Internship Berbasis Web yang telah ada yaitu www.anyintern.com. Berikut ini penulis uraikan beberapa penelitian pada website pencarian lowongan *internship* berbasis website (www.anyintern.com) mengenai fungsionalitas sistem nya:

Tabel 2.2.1 Penelitian Sebelumnya

Nama *website*	Penjelasan *website*	Kekurangan
1. www.anyintern.com	*website* anyintern merupakan *website* daftar lowongan *intership* yang mengumpulkan semua informasi *internship* yang tersedia untuk mahasiswa dalam berbagai industri seperti teknologi, musik, film, *fashion, startup*, dan lain-lain. *website* anyintern ini gratis untuk semua mahasiswa.	1. Tampilan kurang menarik dan monoton 2. Tidak tersedia fitur *filter*, untuk mencari lowongan *intership* yang lebih spesifik sesuai dengan kategori calon pelamar pada *website*

Berdasarkan penelitian di atas menjelaskan bahwa fungsionalitas sistem khususnya pada tampilan *website* dan fitur yang dimiliki harus memiliki fungsionalitas yang baik dan kompetitif dengan *website* lainnya, hal ini bertujuan agar mendapatkan kepuasan dengan *website* yang digunakannya.

2.2.2. Ulasan konten dari 3 jurnal yang telah didefinisikan di bab 1

2.2.2.1 Sistem Magang di Indonesia

Masalah *internship* telah diatur dalam Undang-Undang No. 13 tahun 2013 tentang Ketenagakerjaan khususnya pasal 21 – 30 dan lebih spesifiknya diatur dalam Peraturan Menteri Tenaga Kerja dan

Transmigrasi No. Per. 22/Men/IX/2009 tentang Penyelenggaraan Pemagangan di Dalam Negeri (Gajimu, 2011, Sistem Magang di Indonesia, http://www.gajimu.com/main/tips-karir/sistem-magang-di-indonesia, diakses pada tanggal 18 September 2016).

Internship merupakan bagian dari pelatihan kerja, khususnya di Indonesia biasanya *internship* dilakukan oleh mahasiswa tingkat akhir yang menjadi salah satu syarat utama untuk menyelesaikan proses pendidikannya. Mahasiswa juga dapat menambah wawasan mengenai dunia industri dan meningkatkan keterampilan serta keahlian praktek kerja.

Internship memberikan keuntungan bagi pesertanya yaitu:

- Mendapatkan sertifikat dari lembaga pelatihan kerja apabila yang bersangkutan telah menyelesaikan proses *internship*
- Mendapatkan pengalaman dunia industri
- Mendapatkan fasilitas keselamatan dan kesehatan kerja selama peserta melakukan *internship* di beberapa perusahaan tertentu
- Mendapatkan uang saku dan transportasi sesuai perjanjian antara peserta *internship* dengan lembaga pelatihan *internship* dan penyelenggara program *internship* di beberapa perusahaan tertentu.

Penulis juga berpendapat sistem magang di Indonesia masih terlalu rumit yaitu keterbatasan daftar perusahaan yang jumlahnya kurang mendukung dan keterbatasan relasi yang dimiliki oleh calon peserta *internship*, terkadang dari beberapa universitas mahasiswanya diharuskan mencari tempat *internship* sendiri.

2.2.2.2 Alasan mahasiswa perlu kerja magang

Alasan mahasiswa perlu kerja magang atau *internship*, ada 5 alasan mahasiswa perlu internship:

i. Mengaplikasikan Ilmu

Kerja magang atau *internship* mahasiswa bisa langsung mempraktikan ilmu dan teori yang sudah di pelajarinya. Mahasiswa juga mendapat kesempatan mengetahui keadaan dunia kerja yang sesungguhnya.

ii. Mengukur Kemampuan Diri

Dengan mengikuti program *internship*, mahasiswa akan memiliki kesempatan untuk mengukur kemampuan yang dimilikinya, selain itu bidang apa saja yang bisa mahasiswa tersebut kuasai dengan baik, serta hal apa yang harus mahasiswa tersebut perbaiki dibidang yang ia kurang paham.

iii. Mengenali Lingkungan Bekerja

Mahasiswa berkesempatan mengenali lingkungan kerjanya lebih dekat.

iv. Memahami Proses Bekerja

Dunia kerja dan kuliah sesuatu yang berbeda, proses-proses bekerja yang sering diceritakan di kuliah berbeda dengan kenyataan yang ada di lapangan kerja. Mahasiswa dengan mengikuti *internship* dapat mengenal proses-proses yang dilakukan di dunia kerja dengan lebih baik lagi, mulai dari hambatan yang akan di hadapi mahasiswa tersebut, sehingga mahasiswa tersebut bisa belajar untuk mengantisipasinya.

v. Nilai Tambah Bagi Perusahaan

Rata-rata perusahaan menganggap bahwa *internship* adalah nilai tambah bagi seorang kandidat saat perusahaan tersebut melakukan perekrutan pegawai baru. Perusahaan lebih suka jika pegawai baru tersebut sudah pernah terjun ke dunia kerja sebelumnya, walau dalam bentuk *internship*.

(Qerja, 2015, 5 Alasan Mahasiswa Perlu Kerja Magang, http://en-id.qerja.com/journal/view/283-5-alasan-mahasiswa-perlu-kerja-magang/, diakses pada tanggal 1 Oktober 2016).

2.2.2.3 Jumlah pemagang di Indonesia melalui media *online* masih kecil

Tren *internship* di beberapa negara Eropa, khususnya Inggris sedang mengalami peningkatan. Sebuah survei yang dirilis lembaga NAS di Inggris menyebutkan bahwa terjadi peningkatan pelamar *internship* melalui *online* sekitar 41% dibandingkan tahun 2013 lalu. Namun di Indonesia, angka pelamar *internship* melalui *online* masih relative kecil (Portalhr, 2013, Jumlah Pelamar Magang *Via Online* di Indonesia Masih Kecil, http://portalhr.com/berita/jumlah-pelamar-magang-via-online-di-indonesia-masih-kecil/, diakses pada tanggal 1 Oktober 2016).

Penulis juga berpendapat sama dengan artikel tersebut, hal ini memang jumlah pemagang di Indonesia melalui media *online* masih sangat dikit peminatnya, permasalahan ini dikarenakan :

i. Permasalahan pertama, komitmen saat *internship* membutuhkan waktu yang lebih dalam bekerja, mahasiswa layaknya bekerja seperti karyawan lainnya. Hal ini tidak sesuai dengan aturan

mahasiswa sebagai calon pegawai yang sedang belajar langsung di dunia kerja.

ii. Permasalahan kedua, jumlah pemagang di Indonesia melalui media *online* masih sangat sedikit peminatnya dikarenakan jumlah calon mahasiswa yang ingin *internship* belum bisa tertampung oleh univertsitas nya, selama ini universitas sudah memfasilitasi kemudahan bagi mahasiswa untuk *internship*, namun belum maksimal karena jumlah mahasiswa yang ingin *internship* lebih banyak jika dibandingkan dengan daftar lowongan *internship* yang disediakan.

2.3. Topik Khusus

2.3.1. jQuery

Berdasarkan website resmi jQuery (https://jquery.com/, diakses pada 20 Oktober 2016), jQuery merupakan sebuah *library* JavaScript yang cepat, kecil, dan kaya fungsi. jQuery membuat berbagai hal pemrograman website, seperti mengatur animasi, transisi, *event handling*, dan Ajax menjadi lebih mudah.

2.3.2. PHP

Menurut Welling & Thomson (2009:2), PHP Hypertext Preprocessor (PHP) merupakan sebuah bahasa *server-side scripting* yang dirancang secara khusus untuk *web*. Dalam halaman HTML, kita bisa menyisipkan kode PHP yang akan dieksekusi setiap kali halaman tersebut diakses. Kode PHP akan diterjemahkan oleh *web server* dan menghasilkan HTML atau *output* lainnya yang akan dilihat oleh pengunjung.

PHP dibuat pada tahun 1994 oleh Rasmus Lerdorf, yang kemudian diadopsi oleh orang-orang bertalenta lainnya dan telah mengalami empat kali penulisan ulang besar, yang menghasilkan produk yang luas dan matang seperti sekarang. PHP merupakan produk *open source*, sehingga kita bisa mengakses *source code*-nya, serta menggunakan, mengubah, dan mendistribusikan ulang tanpa dikenai biaya.

Selain PHP, terdapat berbagai macam bahasa *server-side scripting* lainnya, seperti Perl, Microsoft ASP.NET, Java Server Pages (JSP), dan ColdFusion. Namun, dibandingkan dengan bahasa-bahasa tersebut, PHP mempunyai banyak kelebihan, diantaranya:

- Performa tinggi

PHP sangatlah efisien. Hanya dengan satu *server* yang tidak mahal, Anda bisa melayani satu juta pengunjung per hari. Dengan *server* komoditas yang banyak, kapasitas tersebut bisa menjadi tidak terbatas secara efektif.

- *Interface* ke berbagai sistem database

  PHP mempunyai koneksi bawaan ke berbagai sistem database. Selain MySQL, Anda bisa juga mengkoneksikan PHP secara langsung ke PostgreSQL, mSQL, Oracle, dbm, FilePro, Hyperwave, Informix, Interbase, dan database Sybase. Bahkan, dengan menggunakan *Open Database Connectivity Standard* (ODBC), Anda bisa mengakses database manapun yang menggunakan *driver* ODBC, termasuk diantaranya Microsoft SQL Server.

- *Built-in libraries* untuk berbagai tugas web

  Karena PHP dirancang untuk penggunaan di *web*, PHP mempunyai banyak fungsi bawaan untuk melakukan berbagai pekerjaan, seperti menghasilkan gambar GIF dengan cepat, mengkoneksikan PHP ke *web services*, XML *parsing*, mengirim email, bekerja dengan *cookies*, dan menghasilkan dokumen PDF, semuanya hanya dengan beberapa baris kode.

- Gratis

  Untuk menggunakan PHP, Anda sama sekali tidak perlu membayar. Anda cukup mendownload PHP dan Anda bisa langsung menggunakannya.

- Kemudahan dalam pembelajaran dan penggunaan

  Sintaks PHP dibuat berdasarkan bahasa pemrograman lainnya, terutama C dan Perl. Jika Anda sudah mengetahui bahasa C atau Perl, atau bahasa yang mirip dengan bahasa C, seperti C++ atau Java, maka Anda bisa langsung menjadi produktif menggunakan PHP.

- Dukungan berbasis objek yang kuat

  PHP versi 5 mempunyai fitur berbasis objek yang didesain bagus. Jika Anda sudah mempelajari Java atau C++, Anda akan menemukan berbagai fitur yang Anda harapkan, seperti *inheritance*, *private* dan *protected attributes* dan *methods*, *abstract class* dan *method*, *interface*, *constructor* dan *destructor*.

- Portabilitas

  PHP tersedia untuk berbagai macam sistem operasi. Anda bisa menulis kode PHP di sistem operasi berbasis Unix yang gratis seperti Linux dan FreeBSD, versi Unix yang komersil seperti Solaris dan IRIX, atau juga berbagai versi berbeda dari Microsoft Windows. Portabilitas PHP membuatnya bisa berjalan lancar di sistem operasi yang berbeda-beda tanpa perubahan.

- Ketersediaan *source code*

  PHP dilisensi dalam *open source*, sehingga Anda bisa mengakses *source code* PHP secara bebas, dan bahkan mengubah atau menambah fitur baru ke dalam bahasa tersebut.

- Ketersediaan dukungan

  ZAND Technologies, perusahaan yang mengembangkan PHP, membiayai pengembangannya dengan menawarkan dukungan dan perangkat lunak terkait secara komersil.

2.3.3. MySQL

Menurut Welling & Thomson (2009:3), MySQL merupakan sebuah *relational database management system* berbasis SQL yang cepat dan kuat. *Server*

MySQL mengontrol akses ke data kita untuk memastikan banyak pengguna yang bisa menggunakannya bersama-sama, untuk menyediakan akses yang cepat, dan untuk memastikan hanya pengguna yang berwenang saja yang bisa menggunakannya. Jadi, MySQL merupakan *server* yang *multiuser* dan *multithreaded*.

MySQL mempunyai berbagai kompetitor, seperti PostgreSQL, Microsoft SQL Server, serta Oracle. Namun, yang membedakan MySQL adalah berbagai kelebihan yang dimilikinya:

- Performa tinggi

  Pada tahun 2002, majalah *eWeek* mempublikasikan *benchmark* yang menunjukkan MySQL beberapa kali lipat lebih cepat daripada sebagian kompetitornya. Bahkan, hasil terbaik pada tes tersebut hanya didapat oleh MySQL dan Oracle yang jauh lebih mahal.

- Harga yang relatif murah

  MySQL tersedia secara gratis dengan lisensi *open source* untuk penggunaan pribadi dan dengan harga yang murah untuk lisensi komersil.

- Kemudahan dalam penggunaan

  Sebagian besar database modern menggunakan SQL. Jika Anda pernah menggunakan RDBMS lainnya, Anda tidak akan kesulitan untuk beradaptasi menggunakan MySQL.

- Portabilitas

  MySQL bisa digunakan di berbagai sistem UNIX dan juga di Microsoft Windows.

- *Source Code*

Seperti juga PHP, Anda bisa mendapatkan dan mengubah *source code* dari MySQL karena kedua produk tersebut dilisensi dalam *open source*.

- Ketersediaan dukungan

  Tidak semua produk *open source* mempunyai perusahaan yang menyediakan dukungan, pelatihan, konsultasi, dan sertifikasi, tetapi MySQL menyediakan semua hal tersebut sehingga Anda tidak perlu lagi khawatir akan sulit mencari dukungan.

2.3.4. Laravel

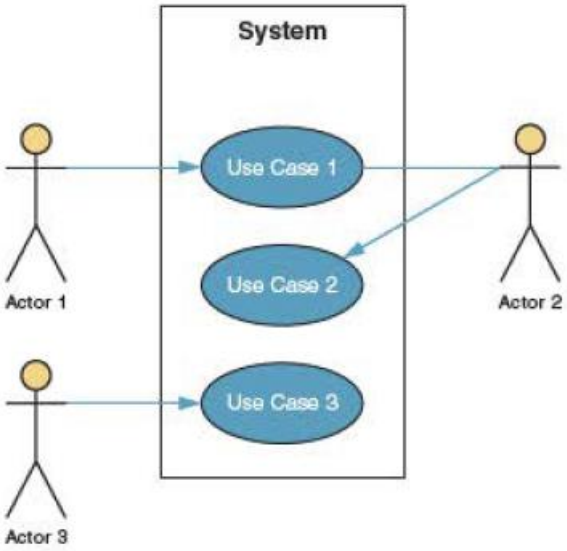
Menurut Shawn McCool (2012:2), Laravel merupakan *framework* MVC untuk pengembangan *web* yang ditulis dalam bahasa PHP. Laravel didesain untuk meningkatkan kualitas *software* dengan cara mengurangi biaya pengembangan awal dan biaya pemeliharaan terus-menerus, dan juga untuk meningkatkan pengalaman mengerjakan aplikasi dengan menyediakan sintaks yang jelas dan ekpresif serta sekumpulan fungsionalitas yang akan membantu mengurangi waktu implementasi.

2.3.5. UML

Menurut Grady Booch (1999:13), Unified Modeling Language (UML) adalah bahasa grafis standar untuk menulis blueprint perangkat lunak. UML bisa digunakan untuk memvisualisasikan, menspesifikasikan, membentuk, dan mendokumentasikan artifak dari sistem perangkat lunak, sehingga bisa memberi gambaran akan hal-hal konseptual, seperti proses bisnis dan fungsi sistem, dan juga hal-hal konkret, seperti skema database dan komponen perangkat lunak.

2.3.4.1 Use Case Diagram

Whitten & Bentley (2007:262), use case diagram secara grafis menggambarkan sebuah sistem sebagai kumpulan dari *use cases*, *actors* (*users*), dan relasinya. Rincian dari setiap *business event* dan bagaimana *users* berinteraksi dengan sistem dijelaskan pada artefak kedua, yang disebut *use-case narrative*, yang dijelaskan secara tertulis mengenai *business event* dan bagaimana *user* berinteraksi dengan sistem untuk menyelesaikan sebuah tugas.

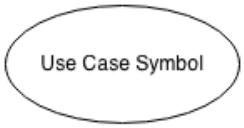


Gambar 1.1 Model diagram *Use Case*

(Sumber: Whitten & Bentley, 2007:246)

a. Use Case

Use case mendeskripsikan fungsi sistem dari perspektif *user* dengan menggunakan kata-kata dan terminologi yang mereka pahami (Whitten & Bentley, 2007:246). *Use case* dilambangkan dengan simbol:



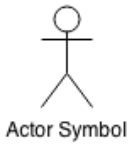
Gambar 1.2 Simbol *Use-Case*

Use case hanya menyatakan satu tujuan dari sistem dan mendeskripsikan rentetan aktivitas dan interaksi pengguna dalam mencapai tujuan tersebut.

b. *Actor*

Menurut Whitten & Bentley (2007:247), *use case* di inisiasi atau dipicu oleh *external users* yang disebut *actors*. *Actor* merupakan segala sesuatu yang berinteraksi dengan sistem untuk bertukar informasi. *Actor* tidak harus harus manusia, *actor* dapat berupa organisasi, peralatan external (seperti : sensor panas), atau sistem informasi yang lainnya.

Actor digambarkan sebagai *stick figure* yang diberi nama sesuai dengan peran yang dilakukan oleh *actor* tersebut. Berikut contoh simbol *actor* :



Gambar 1.3 Simbol *Actor*

Actor dibagi menjadi 4 tipe :

- ***Primary business actor***, *stakeholder* diuntungkan dari eksekusi sebuah *use case* dengan menerima sesuatu yang dapat diukur atau nilai yang dapat diamati.

  Contohnya : seorang pegawai menerima gaji berupa memiliki nilai.

- ***Primary system actor***, *stakeholder* yang berinteraksi secara langsung dengan sistem untuk memulai sebuah *business event* atau *system event*.

  Contohnya : seorang pegawai bank (*bank teller*), yang memproses transaksi bank.

- ***External server actor***, *stakeholder* yang merespon sebuah permintaan dari *use case*.

  Contohnya : biro kredit yang mengotorisasi pembayaran dengan kartu kredit.

- ***External receiver actor***, *stakeholder* yang bukan sebagai *primary actor* tapi menerima sesuatu yang dapat diukur atau nilai yang dapat diamati dari *use case*.

  Contohnya : Gudang mempersiapkan pengiriman pesanan setelah pelanggan memesan suatu barang.

c. *Relationship*

Menurut Whitten & Bentley (2007:248), *relationship* digambarkan sebagai garis yang menghubungkan antara dua *use case*. Berikut 5 tipe *relationship* pada *use case* :

- ***Association*** adalah *relationship* antara *actor* dan *use case* dimana interaksi terjadi diantara mereka berdua.

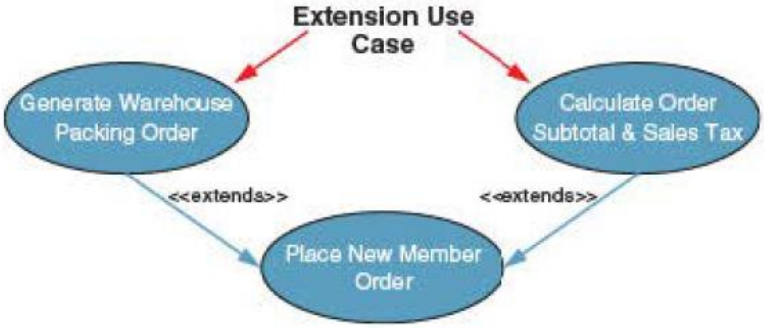
  Association dengan tanda panah mengindikasikan *use case* diimitasi oleh *actor* pada ujung garis yang lainnya.

  Association tanpa tanda panah mengindikasikan sebuah interaksi antara *use case* dengan sebuah *external server* atau *receiver actor*.

Gambar 1.4 Contoh *Association Relationship*

(Sumber: Whitten & Bentley, 2007:248)

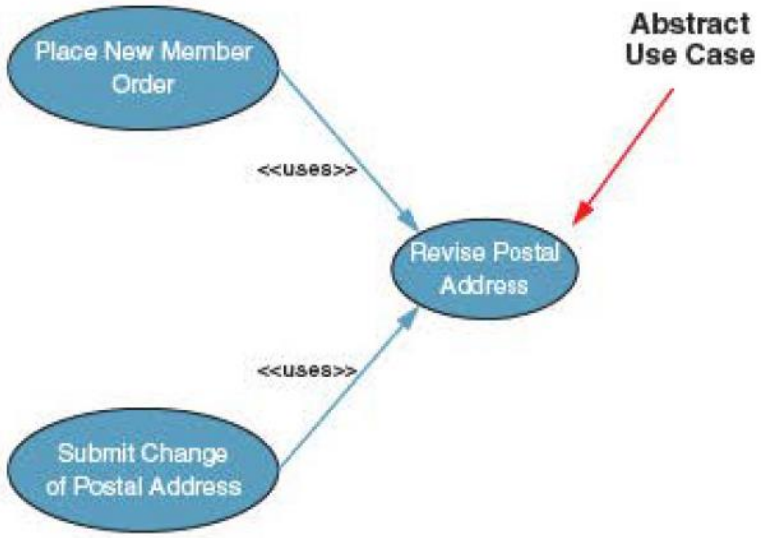
- ***Extends***, sebuah *relationship* untuk menambahkan bagian pada *use case* yang ada serta untuk pemodelan sistem layanan opsional dan digambarkan dengan simbol "<<extends>>". Contohnya : seorang pelanggan ingin melihat barang di suatu situs, maka untuk melihat barang tersebut pelanggan tidak perlu melewati proses login dan pesan barang.



Gambar 1.5 Contoh *Extends Relationship*

(Sumber: Whitten & Bentley, 2007:249)

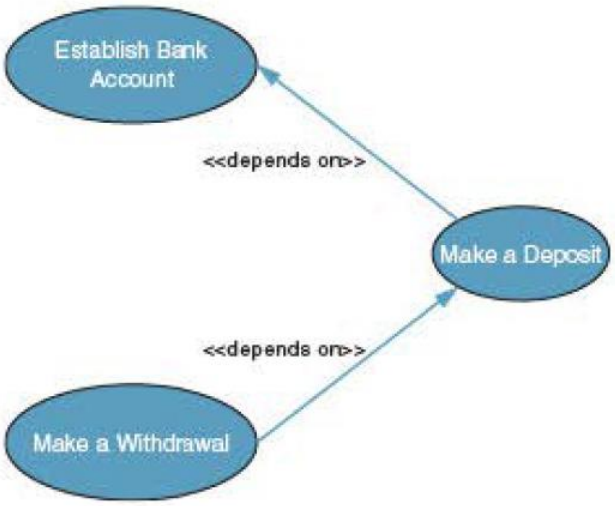
- ***Uses atau Includes***, *relationship* antara *abstract use case* dan *use case* disebut *uses relationship*. *Abstract use case* merupakan *use case* yang mengurangi redundansi diantara dua atau lebih *use case* dengan cara menggabungkan langkah-langkah yang sama yang terdapat pada banyak *use case* tersebut. *Abstract use case me*representasikan sebuah bentuk "*reuse*" (penggunaan ulang) dan merupakan alat yang bagus untuk mengurangi redundansi pada *use case*. *Uses relationship* digambarkan dengan simbol "<<uses>>" atau "<<indclude>>".



Gambar 1.6 Contoh *Uses Relationship*

(Sumber: Whitten & Bentley, 2007:249)

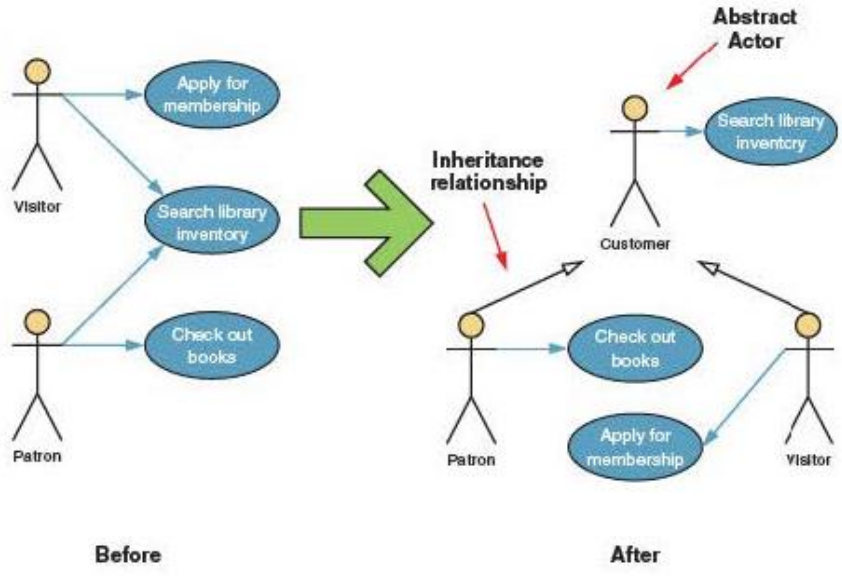
- ***Depends On,*** sebuah *relationship* antara *use cases* yang mengindikasikan satu *use case* tidak dapat dilakukan sampai *use case* lain telah dilakukan.



Gambar 1.7 Contoh *Depends On Relationship*

(Sumber: Whitten & Bentley, 2007:250)

- ***Inheritance***, sebuah *relationship* antara *actors* yang dibuat untuk menyederhanakan gambar ketika sebuah *abstract actor* mewarisi beberapa peran atau *role* dari *real actors*.



Gambar 1.8 Contoh *Inheritance Relationship*

2.3.4.2 Use Case Narrative

Menurut Whitten & Bentley (2007;246), *use case narrative* merupakan deskripsi tertulis dari *business event* dan bagaimana *user* akan berinteraksi dengan system untuk menyelesaikan sebuah *task* atau tujuan. Untuk membuat *use case narrative* harus mengikuti format dalam bentuk berikut :

Tabel 2.2.2 Format *Use Case Narrative*

Elemen	Keterangan
Use case name	Nama *use case* harus merepresentasikan sebuah tujuan yang akan dicapai *use case*. Nama harus diawali dengan kata kerja (contoh : *Enter New Member Order*).
Use case id	Penanda unik mendefinisikan *use case.*
Priority	Mengkomunikasikan tingkat kepentingan *use case* dari tingkat *low*, *medium*, atau *high.*
Primary business actor	*Stakeholder* akan mendapatkan keuntungan langsung dari eksekusi *use case* dengan menerima sesuatu yang bisa diukur maupun dinilai.
Description	Deskripsi singkat yang berisi beberapa kalimat yang menguraikan tujuan dan aktivitas dari sebuah *use case.*
Precondition	*Use case* lain harus dijalankan terlebih dahulu sebelum *use case* ini dieksekusi.
Trigger	*Event* yang memulai eksekusi sebuah *use case*. Biasanya berupa *physical action* atau waktu.
Typical Course of Events	Rentetan aktivitas yang dilakukan oleh *actor* dan *system* dengan maksud untuk memenuhi sasaran dari *use case*.

2.3.4.3 Class Diagram

Class Diagram merupakan diagram yang paling sering ditemukan pada modeling sistem berbasi objek. Class Diagram menunjukan sekumpulan *class*, *interface*, dan *colaborations*, dan diantaranya. Class Diagram penting, tidak hanya untuk visualisasi, spesifikasi, dan dokumentasi model struktural, tetapi

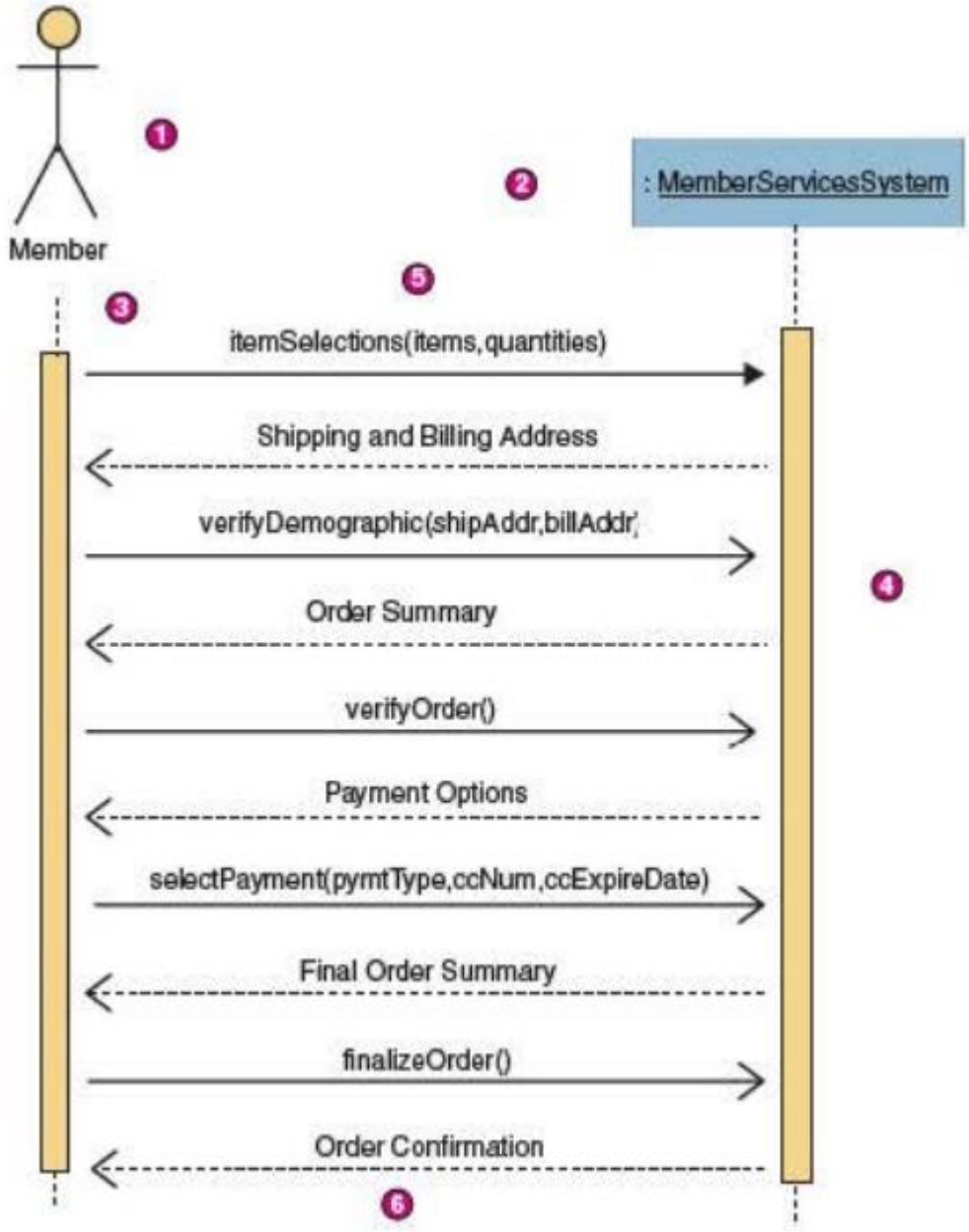
juga untuk kontruksi sistem yang dapat dieksekusi kedepannya dan rekayasa terbalik.

2.3.4.4 Sequence Diagram

Menurut Whitten & Bentley (2007:394), *sequence diagram* merupakan diagram yang menampilkan bagaimana objek saling berinteraksi satu sama lain melalui pesan dalam eksekusi sebuah *use case* atau operasi.

Sequence diagram membantu kita dalam mengenali pesan tingkat tinggi yang masuk dan keluar sistem. Selanjutnya, pesan tersebut akan menjadi tanggung jawab dari objek-objek individu, yang nantinya akan memenuhi tanggung jawab tersebut melalui komunikasi dengan objek-objek lain.

Sequence diagram tidak memuat satu pun alur alternatif dari *use case*, melainkan hanya menampilkan satu skenario, satu jalur yang melalui suatu *use case*. Jadi, sekumpulan *sequence diagram* yang utuh bisa saja mempunyai beberapa diagram untuk satu *use case*.



Gambar 1.4

Contoh *Sequence Diagram* untuk *Use Case* Membuat *Order* Baru

(Sumber: Whitten & Bentley, 2007:395)

Berikut ini notasi penggambaran *sequence diagram*:

1. *Actor*

   Orang yang melakukan *use case*, digambarkan dalam bentuk *stick figure*

2. *System*

   Sebuah kotak yang mengindikasikan sistem sebagai "*black box*" atau sebagai keseluruhan. Titik dua (:) merupakan notasi standar *sequence diagram* untuk mengindikasikan sebuah "instansi" sistem yang sedang berjalan.

3. *Lifelines*

   Garis vertikal putus-putus yang memanjang kebawah dari *actor* dan *system*, yang menandakan masa aktif dari *sequence*.

4. *Activation bars*

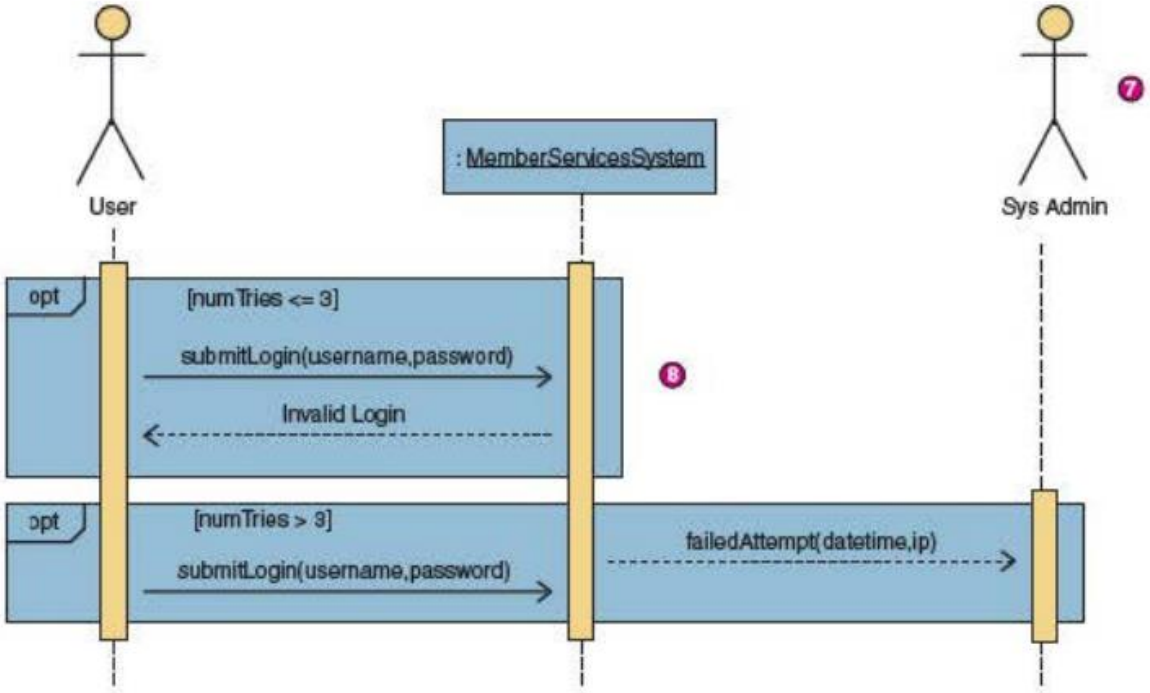
   Batang yang menutupi *lifelines* mengindikasikan periode waktu saat *actor* atau *system* aktif dalam interaksi.

5. *Input messages*

   Panah horizontal dari *actor* menuju *system* yang menandakan data input. Konvensi UML untuk *input messages* yaitu memulai kata pertama dengan huruf kecil dan menambahkan tambahan kata lainnya dengan awalan huruf kapital dan tanpa spasi. Pada kurung, masukkan parameter yang sudah kita ketahui, diikuti dengan konvensi nama yang sama dan memisahkan masing-masing parameter dengan koma.

6. *Output messages*

   Panah horizontal dari *system* menuju *actor* yang menandakan data output, bentuknya berupa garis putus-putus.



Gambar 1.5

Contoh *Sequence Diagram* untuk validasi *login*

(Sumber: Whitten & Bentley, 2007:396)

7. *Receiver Actor*

   Actor lain atau *external system* yang menerima data dari *system*.

8. *Frame*

   Kotak yang melingkupi satu atau lebih data untuk membagi suatu pecahan *sequence*. *Frame* ini bisa untuk menunjukkan *loop*, pecahan *sequence* alternatif, atau *sequence* tambahan. Untuk *sequence* tambahan, kondisinya bisa ditampilkan di kurung siku yang menandakan kondisi dimana *sequence* tersebut akan dijalankan.

2.3.4.5 Activity Diagram

Activity Diagram adalah diagram yang dapat digunakan untuk menggambarkan grafik dari aliran proses bisnis, langkah-langkah dari *use case*, atau logika dari perilaku objek.

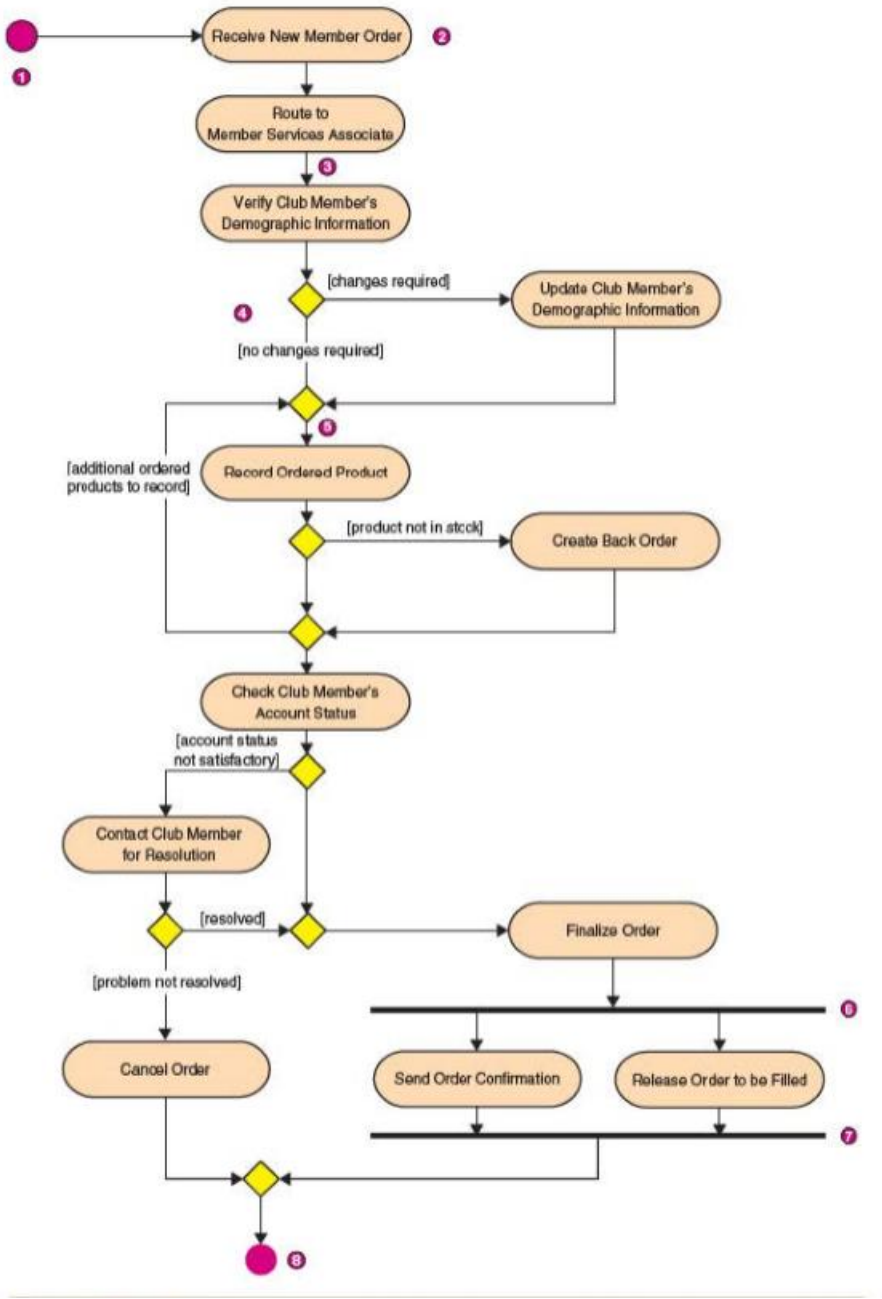
Notasi penggambaran *activity diagram* :

1. *Initial node,* lingkaran padat yang mewakili awal dari proses.
2. *Actions,* persegi panjang bulat merepresentasikan langkah-langkah invidu. Urutan *action* membentuk aktivitas secara total yang ditunjukan oleh diagram.
3. *Flow,* panah pada diagram menunjukakan perkembangan dari *action*. Kebanyakan *flow* tidak memerlukan kata-kata untuk mengindetifikasikan mereka kecuali keluar dari keputusan.
4. *Decision,* bentuk berlian dengan satu *flow* masuk dan dua atau lebih *flow* keluar. *Flow* yang keluar ditandai untuk menunjukkan kondisi
5. *Merge,* bentuk berlian dengan dua atau lebih *flows* masuk dan satu *flow* keluar. Ini menggabungkan *flows* yang sebelumnya dipisahkan oleh *Decisions.* Proses berlanjut dengan salah satu *flow* yang dating kearah *merge*
6. *Fork* , bar hitam dengan satu *flow* dating dan dua atau lebih *flow* keluar. *Actions* pada *parallel flows* dibawah *fork* dapat terjadi dalam urutan apapun atau bersamaan
7. *Join,* bar hitam dengan dua atau lebih *flows* masuk dan satu *flow* keluar, mencatat akhir pemrosesan secara bersamaan. Semua *actions* yang masuk kedalam *join* harus selesai sebelum proses berlanjut.

8. *Activity final*, lingkaran padat dalam lingkaran berlubang merepresentasikan akhir proses

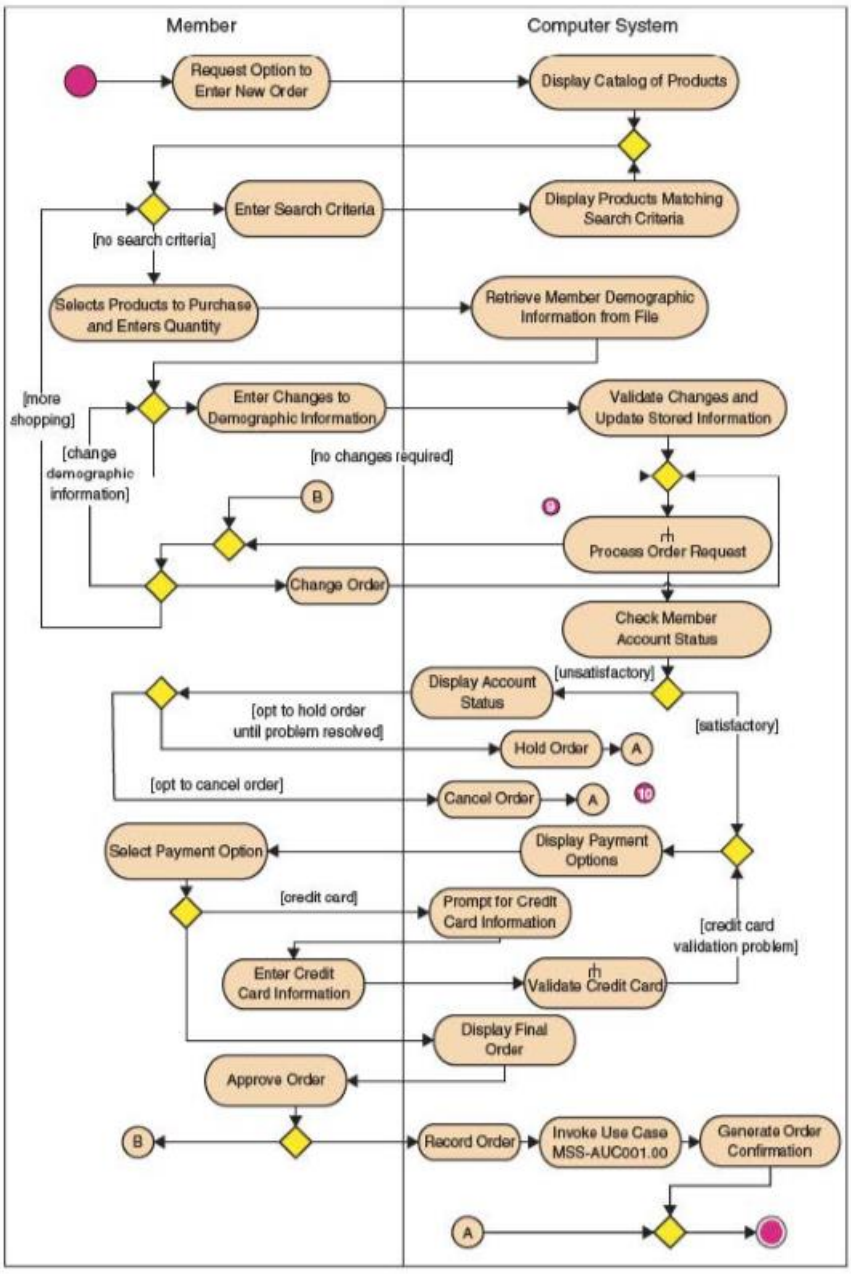
Fitur tambahan *activity diagrams*:

9. *Subactivity indicator*, simbol trisula pada *action* menunjukkan bahwa *action* ini dipisahkan di *activity diagram* yang lain. Ini membantu anda agar tidak membuat *activity diagram* menjadi terlalu rumit.
10. *Connector*, sebuah surat dalam lingkaran yang memberikan alat lain untuk mengelola kompleksitas. Sebuah *flow* yang menuju ke *connector* akan berpindah ke *flow* yang keluar dari *connector* dengan huruf yang sama.



Gambar 1.6 Contoh Activity *Diagram* untuk memasukkan anggota baru

hal 393



Gambar 1.7 Contoh *Activity Diagram* untuk menempatkan *order* baru dengan partisi hal 394

Pedoman membangun *Activity Diagram*

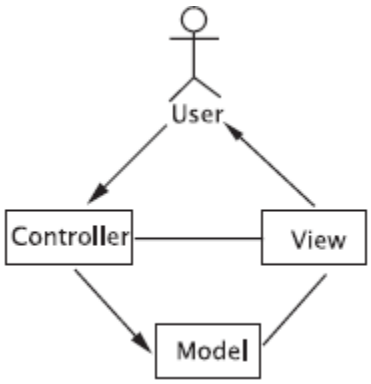
Daftar berikut merupakan proses yang sangat baik untuk membangun *activity diagram*

- Dimulai dengan *node* awal sebagai titik awal

- Menambahkan partisi jika mereka relevan dengan analisi mereka
- Menambahkan *action* untuk setiap langkah utama dari *use case*
- Menambahkan *flows* dari setiap *action* menuju ke *action* lain, *decision point*, atau *end point.* Untuk ketepatan makna yang maksimal, setiap *action* harus memiliki hanya satu *flow* yang datang dan satu *flow* yang keluar, dengan semua *forks, joins, decisions,* dan *merges* yang ditampilkan secara tersurat
- Menambahkan *decisions* dimana *flows terpisah dengan rute* alternative. Pastikan untuk membawa mereka kembali bersama dengan sebuah *merge.*
- Menambahkan *forks* dan *joins* dimana aktivitas-aktivitas dilakukan secara parallel
- Mengakhiri dengan sebuah notasi tunggal

2.3.5 MVC (*Model-View-Controller*)

Sarnath Ramnath & Brahma Dathan (2010:240) mengatakan bahwa *model-view-controller* adalah pola yang relaitf lama yang diperkenalkan di *Smalltalk programming language*. *Model-view-controller* merupakan pola membagi sebuah aplikasi menjadi 3 subsistem: *model, view,* dan *controller*.



Gambar 1.8 *Model-view-controller architecture*

(Sumber: Sarnath Ramnath & Brahma Dathan, 2010:240)

Model, merupakan *object passive* yang relative, yaitu menyimpan data. Setiap object memiliki aturan tersendiri dalam *model*. *View* menerjemahkan *model* kedalam bentuk format yang ditentukan, biasanya yang cocok untuk interaksi dengan *end user*. Misalnya jika *model* menyimpan informasi tentang rekening bank, pada *view* tertentu saja yang dapat menampilkan jumlah rekening dan total saldo rekening. Sedangkan, *controller* mendapatkan input dari penggunan dan jika diperlukan, lalu *method calls* pada *model* akan mengubah data yang tersimpan. Ketika *model* diubah, maka *view* akan respon dengan apa yang diubah lalu ditampilkan.