BAB II

LANDASAN TEORI

2.1 *System Development Life Cycle*

System development life cycle (SDLC) adalah proses untuk memahami bagaimana sebuah sistem informasi dapat mendukung kebutuhan bisnis dengan merancang suatu sistem, membangun sistem tersebut dan menyampaikan kepada pengguna (Tegarden, Dennis, Wixon, 2013).

SDLC memiliki empat fase dasar yaitu *planning*, *analysis*, *design* dan *implementation*. Setiap fase itu sendiri terdiri dari serangkaian langkah yang menggunakan cara tertentu dalam menghasilkan goal yang dicapai. Pada poin berikut akan dijelaskan secara singkat dari keempat fase tersebut.

a. *Planning*

Fase *planning* adalah proses dasar dalam memahami mengapa sistem informasi harus dibangun dan menentukan bagaimana tim proyek akan membangun *project* tersebut.

b. *Analysis*

Fase *analysis* adalah jawaban dari pertanyaan siapa yang akan menggunakan sistem, apa yang yang akan dilakukan oleh sistem, dan dimana serta kapan sistem tersebut akan digunakan. Pada fase ini pula tim proyek menginvestigasi sistem yang sudah ada sebelumnya, mengidentifikasi peluang untuk perbaikan dan mengembangkan konsep yang baru untuk sistem yang akan dibuat.

c. *Design*

Fase *design* yaitu menentukan bagaimana sistem akan beroperasi, dalam hal ini antara lain perangkat keras, perangkat lunak, infrastruktur jaringan (*user interface*), *forms* dan laporan (*database* dan *file* yang dibutuhkan aplikasi).

d. *Implementation*

Fase final pada SDLC ini adalah fase *implementation*, yaitu pada saat sistem telah selesai dibuat. Implementasi pada fase ini biasanya paling banyak mengambil perhatian karena dalam keseluruhan sistem, tahap implementasi adalah tahap yang paling banyak memakan waktu serta biaya karena mencoba keseluruhan sistem.

2.2 Metodologi Pengembangan Sistem

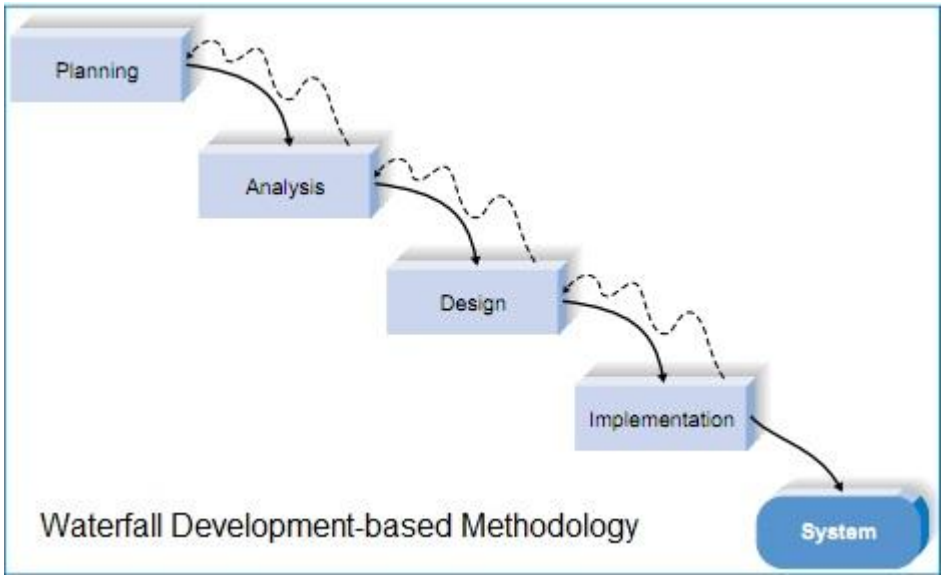
Metodologi merupakan pendekatan formal untuk mengimplementasikan SDLC. Terdapat banyak cara untuk mengkategorikan sebuah metodologi tergantung dengan fokus pada proses bisnis atau data yang mendukung bisnis. Namun dalam pengerjaan penelitian ini menggunakan metodologi yang berorientasi pada objek untuk mencoba menyeimbangkan fokus antara proses dan data dengan menggabungkan keduanya menjadi satu model.

2.2.1 Desain Terstruktur

Metode desain terstruktur mengadopsi langkah-langkah pendekatan terhadap SDLC yang bergerak dari satu tahap ke tahap selanjutnya. Berikut pengembangan dari desain terstruktur:

A. *Waterfall*

Struktur asli dari metode desain terstruktur adalah metode pengembangan *waterfall*. Dengan metode pengembangan *waterfall*, analis dan pengguna memproses melewati tahap demi tahap. Untuk lebih jelasnya dapat dilihat pada Gambar 2.1.



Gambar 2.1 *Waterfall Development*

Kelebihan dari menggunakan metode *waterfall* yaitu karena sistem ini mengidentifikasi kebutuhan sistem jauh sebelum melakukan pemrograman dan berguna untuk meminimalisir terjadinya perubahan pada kebutuhan sistem yang akan dibuat. Namun kekurangan dari sistem ini yaitu desain dari sistem yang akan dibuat harus selesai secara keseluruhan sebelum *programming* aplikasi berjalan dan hal tersebut membutuhkan waktu lama dalam tahap analisis sistem.

B. *Parallel*

Metode ini digunakan untuk mengatasi masalah penundaan yang lama antara analis dan pengiriman pada sistem. Metode ini melakukan desain umum untuk sistem secara keseluruhan kemudian membagi *project* menjadi serangkaian *sub project* kecil yang dapat dirancang dan dilaksanakan secara *parallel*. Ketika semua *sub project* telah selesai maka diintegrasikan menjadi sehingga menjadi kesatuan sistem *project* yang utuh. Untuk lebih jelasnya dapat dilihat Gambar 2.2.

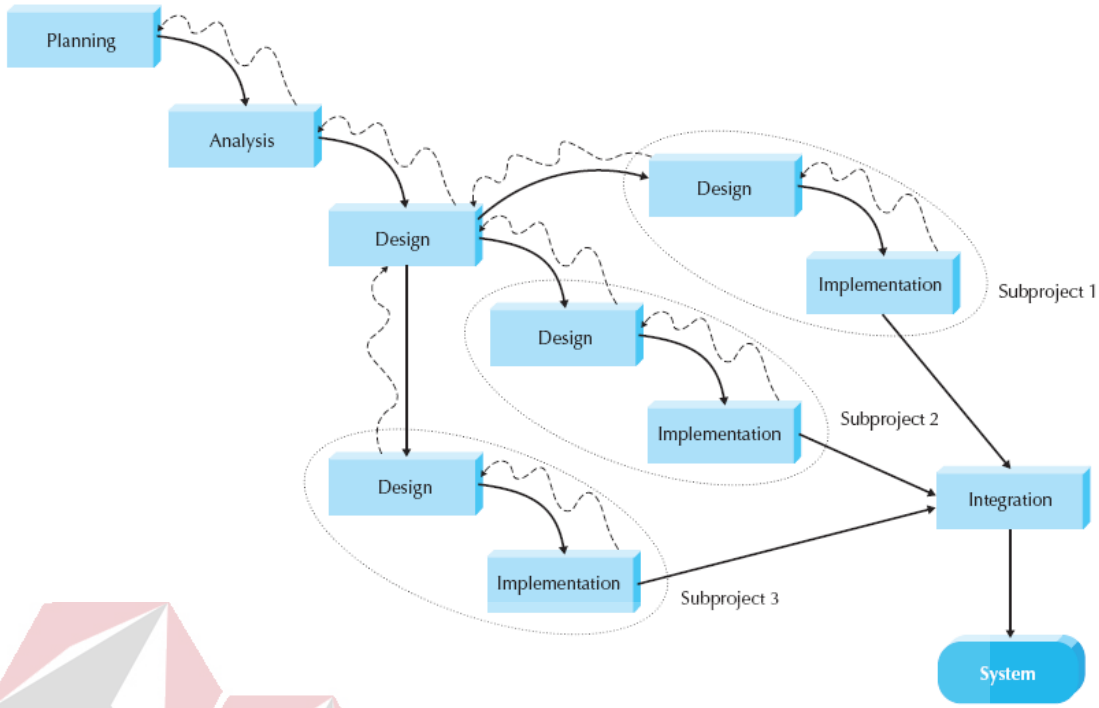


FIGURE 1-3 A Parallel Development–based Methodology

Gambar 2.2 *Parallel Development*

Kelebihan dari metode ini yaitu dapat mengurangi waktu dalam pembuatan sistem serta lebih sedikit kemungkinan terjadinya perubahan dalam lingkungan bisnis yang dibuat. Namun metode ini juga memiliki kekurangan yaitu terkadang *sub project* yang dikerjakan tidak sepenuhnya independen tetapi masih terhubung dengan *sub project* yang lainnya sehingga pada akhir *project* membutuhkan integrasi sistem yang besar.

2.2.2 *Rapid Application Development*

Metode *Rapid Application Development* (RAD) ini berupaya mengatasi kedua kelemahan metodologi desain terstruktur dengan menyesuaikan fase pada SDLC untuk menyelesaikan bagian dari sistem yang dikembangkan dengan cepat dan kembali ke tangan pengguna. Dengan menggunakan metode ini pengguna

dapat lebih memahami sistem dan dapat menyarankan revisi pada aplikasi yang bermasalah agar sistem dapat menjadi lebih dekat dengan tujuan yang diinginkan. Ada tiga pendekatan dasar dari RAD yaitu:

A. *Phased Development*

Metode *phased development* membagi keseluruhan sistem menjadi beberapa versi yang akan dikembangkan secara berurutan. Pada fase analisa mengidentifikasi konsep sistem secara keseluruhan, tim proyek, pengguna dan sistem pendukung lalu mengkategorikan kebutuhan menjadi beberapa versi berbeda. Kebutuhan yang paling penting dan mendasar terikat kedalam versi pertama dari sistem. Berikut metode *phased development* pada Gambar 2.3.

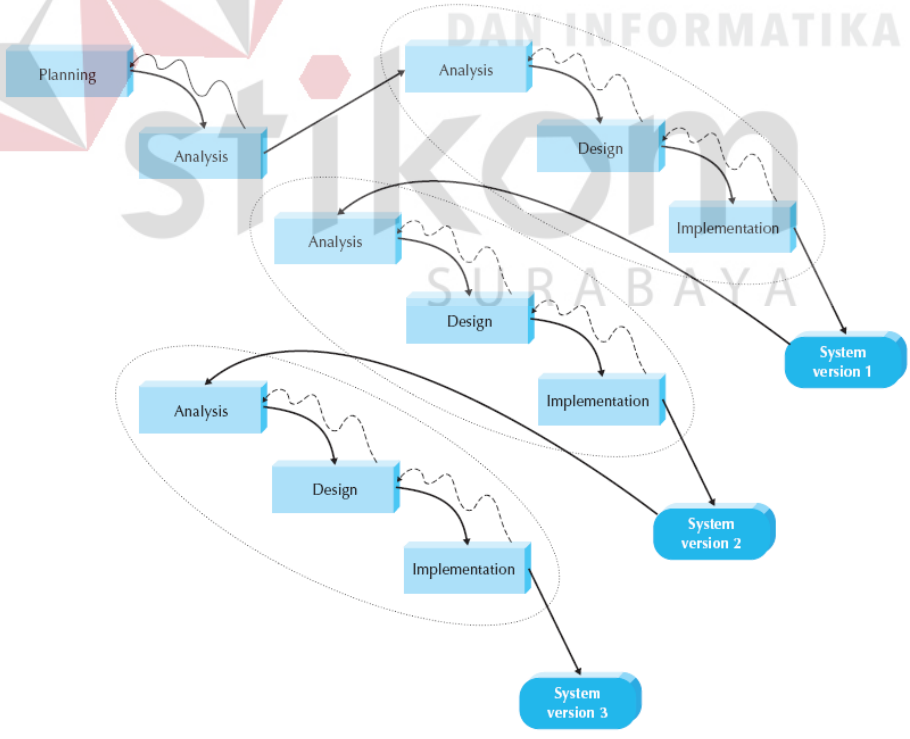


FIGURE 1-4 A Phased Development–based Methodology

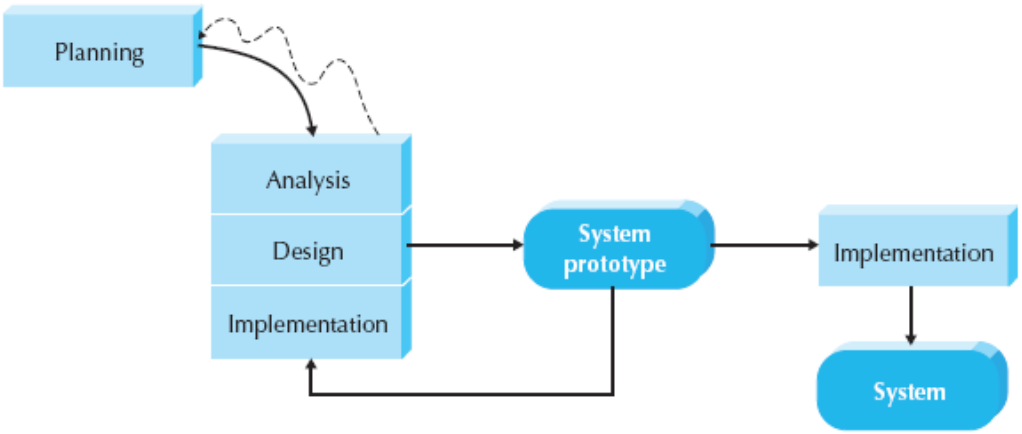
Gambar 2.3 *Phased Development*

Pada saat implementasi dilakukan pada versi pertama, maka akan dilanjutkan ke versi kedua. Analisis tambahan dilakukan berdasarkan dari identifikasi kebutuhan sebelumnya dan dikombinasikan dengan ide-ide baru serta isu yang muncul dari pengalaman pengguna pada versi pertama. Setelah versi yang dikerjakan selesai maka dilanjutkan ke versi berikutnya hingga selesai.

Kelebihan dari penggunaan metode *phase development* ini yaitu secara cepat mendapatkan sistem yang dapat digunakan oleh pengguna. Meskipun sistem masih belum bisa melakukan fungsi-fungsi secara keseluruhan, namun metode ini dapat melakukan proses bisnis yang dibutuhkan pengguna lebih cepat sebelum aplikasi selesai sepenuhnya.

B. *Prototyping*

Metode *prototyping* melakukan fase analisis, desain dan implementasi secara bersamaan, serta ketiga fase tersebut dilakukan berulang kali sampai sistem selesai dibangun. Pada metode ini, analisis dan desain dasar dari sistem dibangun dan langsung dijalankan pada sistem uji coba yang berisi program apa adanya serta fitur-fitur yang masih belum lengkap. Sistem uji coba yang pertama kali dijalankan umumnya berisi sistem yang telah berjalan sebelumnya kemudian ditunjukkan kepada pengguna atau perusahaan untuk mendapatkan masukan-masukan yang nantinya digunakan untuk menganalisis ulang, mendesain ulang dan mengimplementasi ulang pada sistem uji coba yang kedua. Proses ini berlanjut hingga sistem sudah memenuhi kebutuhan fungsionalitas dari pengguna atau perusahaan. Setelah sistem uji coba sudah sempurna maka akan diimplementasikan kedalam sistem yang sesungguhnya. Berikut metode *prototyping* pada Gambar 2.4.



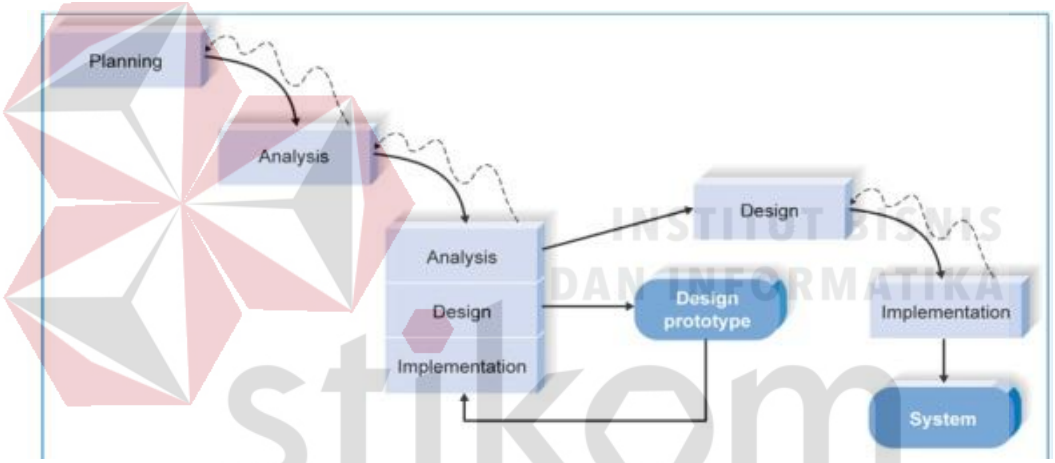
Gambar 2.4 Metode *Prototyping*

Kelebihan dari penggunaan metode ini yaitu dapat membuat sistem yang langsung bisa digunakan oleh pengguna meskipun sistem tersebut masih belum bisa diterapkan pada perusahaan yang bersangkutan. Sistem uji coba tersebut dapat meyakinkan pengguna bahwa tim proyek sudah melakukan pekerjaanya sejak awal (tidak perlu menunggu waktu lama dalam implementasi) dan uji coba tersebut membantu menyelesaikan kebutuhan dari perusahaan secara nyata. Dibandingkan tim proyek harus mempelajari segala aspek spesifikasi kebutuhan perusahaan secara keseluruhan, metode ini lebih baik karena pengguna dapat berinteraksi secara langsung pada sistem uji coba dan mengerti apa yang bisa dilakukan dan mana yang tidak bisa dilakukan.

C. *Throwaway Prototyping*

Metode *throwaway prototyping* hampir sama dengan metode *prototyping* yaitu keduanya melakukan pengembangan melalui sistem uji coba. Pada metode ini diharuskan melewati tahap analisis untuk medapatkan informasi dan

mengembangkan ide-ide untuk konsep dari sistem yang akan dibangun. Namun setelah membuat sistem uji coba, pengguna yang mencoba kemungkinan besar tidak memahami keseluruhan fitur-fitur yang mereka inginkan. Desain sistem uji coba yang dibuat bukanlah sebuah *working system* atau sistem yang dapat langsung digunakan oleh pengguna, namun cenderung sebuah produk yang masih membutuhkan perbaikan. Sistem uji coba ini juga hanya mengandung beberapa informasi yang bisa dipahami oleh pengguna. Berikut dibawah ini Gambar dari metode *throwaway prototyping*.



Gambar 2.5 Metode *Throwaway Prototyping*

Sebuah sistem yang dikembangkan menggunakan metode ini kemungkinan bergantung pada desain uji coba yang berbeda beda baik pada fase analisis maupun fase desain. Setiap uji coba yang digunakan berguna untuk meminimalisir resiko yang ada pada sistem dengan cara mengkorfirmasi bahwa masalah-masalah utama telah dapat diatasi sebelum sistem yang sesungguhnya dijalankan. Ketika semua masalah pada sistem dapat diselesaikan maka *project* menuju fase desain dan implementasi. Inilah yang membedakan metode ini dari metode *prototyping*.

2.2.3 *Agile Development*

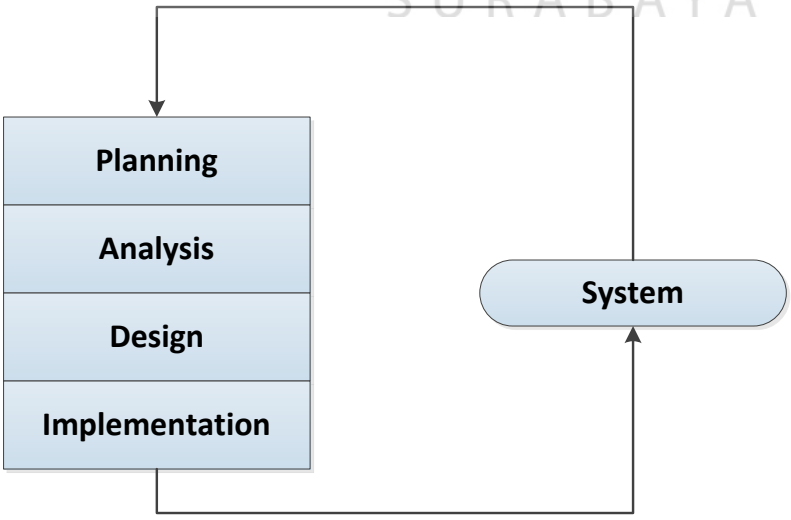
Kategori terakhir dari metodologi pengembangan sistem yaitu *agile development*. berdasarkan kenyataan, yang ditekankan pada metode ini yaitu kondisi kerja dari pengembang, aplikasi, *customer* dan mengatasi perubahan kebutuhan dari perusahaan dibandingkan dengan fokus pada proses pengembangan detail sistem, *tools*, dokumentasi ataupun detail perencanaan. Metode yang terpusat pada *programming* ini memiliki beberapa aturan dan praktek yang secara keseluruhan sangat mudah untuk dilakukan.

Agile development menganut dua belas prinsip dasar. Berikut ini kedua belas prinsip dasar yaitu:

1. *Software* yang dikerjakan diimplementasikan langsung melalui proses pengembangan aplikasi yang kontinyu untuk memuaskan *customer*.
2. Perubahan kebutuhan terjadi pada proses pengembangan.
3. *Update* aplikasi yang dibuat diperbarui secara berkala setiap harinya.
4. *Customer* dan pengembang aplikasi dapat bekerja sama untuk menyelesaikan permasalahan bisnis yang dihadapi.
5. Setiap individu dari tim termotivasi untuk menciptakan solusi dengan cara memberikan mereka *tools* dan lingkungan yang dibutuhkan serta mempercayai mereka untuk menyelesaikan pekerjaannya.
6. Berkomunikasi secara langsung dengan tim adalah metode yang paling efektif dan efisien dalam menentukan kebutuhan dari proyek.
7. Tolak ukur dari kinerja ditentukan dari sejauh mana perkembangan *software* yang dibuat.

8. Baik *customer* maupun pengembang aplikasi harus bekerja dengan ritme yang sama dan dilakukan secara berkelanjutan.
9. Kelincahan yang tinggi dan seimbang baik dalam segi keunggulan teknis dan desain.
10. Kesederhanaan adalah kunci utama. Menghindari pekerjaan yang tidak perlu adalah suatu hal yang penting.
11. Tim yang dijalankan secara terorganisir dapat mengembangkan arsitektur, kebutuhan dan desain yang terbaik.
12. Tim pengembang harus meningkatkan proses pembangunan mereka secara teratur.

Metode *agile* fokus pada pelurusan proses pengembangan sistem dengan menghilangkan pemodelan dan dokumentasi serta waktu yang digunakan untuk melakukan tugas tersebut. Pada metode *agile development* proses kerja yang dilakukan dibuat sesederhana mungkin dan melewati fase-fase tradisional pada proses pengembangan sistem. Berikut Gambar 2.6 metode *agile development*.



Gambar 2.6 Metode *Agile Development*

Keunggulan dari metode ini yaitu sangat cocok digunakan untuk pengembangan sistem yang berorietasi pada objek. Metode *agile development* terbagi menjadi dua yaitu *extreme programming* (XP) dan *scrum*. berikut dibawah ini adalah penjelasan dari masing-masing metode.

A. *Extreme Programming* (XP)

Extreme programming atau XP memiliki empat nilai inti yaitu komunikasi, kesederhanaan, timbal balik dan keberanian. Keempat nilai tersebut memberikan sebuah pondasi pada pengembang XP untuk membuat sebuah sistem. Pertama, pengembang harus memberikan timbal balik berkala yang cepat kepada *end user*. Kedua, metode XP membutuhkan pengembang yang mengikuti prinsip dari KISS. Ketiga, pengembang harus membuat perkembangan lebih baik secara berkala kedalam sistem. Keempat, pengembang harus memiliki mental yang berkualitas.

Testing dan praktek *coding* yang efisien adalah inti dari metode XP. *Coding* yang dibuat harus diuji berkala serta terintegrasi dengan lingkungan *testing*. Apabila terdapat *error*, maka *programmer* harus menyelesaikan program tersebut hingga sempurna.

B. *Scrum*

Dari keseluruhan metode pengembangan sistem yang ada, *scrum* adalah metode yang paling kacau dan untuk mengontrol beberapa kekacauan bawaan itu, metode ini berfokus pada beberapa praktek terapan. Tidak seperti pendekatan lain, pada tim ini tidak memiliki seorang pemimpin yang ditunjuk sebagai *project*

leader. Sebagai gantinya, tim dikelola sendiri secara simbiosis dengan anggota yang lain secara bersamaan.

Tipikal jumlah anggota tim ini berkisar tidak lebih dari tujuh orang. Model pengerjaan yang dilakukan yaitu pembagian kerja yang dilakukan selama perhari yang kemudian dilanjutkan pada hari berikutnya. Dengan metode seperti ini sangat tidak memungkinkan untuk diterapkan pada sistem dengan skala besar dikarenakan tidak ada struktur pembagian kerja yang jelas dan benar.

2.3 Waralaba / *Franchise*

Waralaba adalah perjanjian atau lisensi antara dua pihak yang independen secara hukum diantaranya, yaitu (IFA, 2010, p5):

a. Seseorang atau sekelompok orang yang berhak memiliki bisnis tertentu yang menggunakan merek dagang dari pihak lain.
b. Pengguna *franchise* memiliki hak untuk memasarkan produk atau jasa sesuai metode dan prosedur dari *franchisor* (pemilik *franchise*).
c. Pengguna *franchise* diwajibkan untuk membayar biaya ke *franchisor* untuk hak-hak tersebut.
d. *Franchisor* berhak untuk memberikan hak-hak dan dukungan untuk pengguna *franchise*.

Pada umumnya jenis waralaba (*franchise*) dapat dibedakan menjadi dua yaitu *franchise* lokal dan *franchise* asing. Sementara itu menurut *International Franchise Association* (IFA) yaitu organisasi *franchise* internasional yang beranggotakan Negara-negara di dunia yang berkedudukan di Washington DC, ada empat jenis *franchise* yang mendasar yang biasa digunakan di Amerika Serikat, yaitu:

2.3.1 Produk Franchise

Produsen memberikan hak kepada pemilik toko untuk mendistribusikan barang-barang milik pabrik dan mengijinkan pemilik toko untuk menggunakan nama dan merek dagang pabrik. Pemilik toko harus membayar sejumlah biaya atau membeli persediaan minimum sebagai timbal balik dari hak-hak ini.

2.3.2 Manufacturing Franchise

Jenis *franchise* ini memberikan hak pada suatu badan usaha untuk membuat suatu produk dan menjualnya pada masyarakat, dengan menggunakan merek dagang dan merek *franchisor*. Jenis *franchise* ini seringkali ditemukan dalam industri makanan dan minuman.

2.3.3 Business Oportunity Ventures

Bentuk ini mengharuskan pemilik bisnis untuk membeli dan mendistribusikan produk-produk dari suatu perusahaan tertentu. Perusahaan harus menyediakan pelanggan atau rekening bagi pemilik bisnis, dan sebagai timbal baliknya pemilik bisnis harus membayarkan suatu biaya atau prestasi sebagai kompensasi.

2.3.4 Business Format Franchising

Ini merupakan bentuk *franchising* yang paling populer di dalam prakteknya dimana perusahaan menyediakan suatu metode yang telah terbukti kesuksesannya untuk dioperasikan oleh pemilik bisnis dengan menggunakan nama dan merek dagang perusahaan. Dalam hal ini perusahaan menyediakan sejumlah bantuan tertentu kepada pemilik bisnis dengan membayar sejumlah biaya atau *royalty*. Hasil penelitian Hoffman and Preble (2004) menunjukkan

bahwa *business* format *franchising* yang banyak mengalami pertumbuhan adalah ritel dan restoran.

2.4 Sistem Pembayaran

Sistem pembayaran adalah sistem yang berkaitan dengan kegiatan pemindahan dana dari satu pihak kepada pihak lain yang melibatkan berbagai komponen sistem pembayaran, antara lain alat pembayaran, kliring, dan setelmen. Dalam prakteknya, kegiatan sistem pembayaran melibatkan berbagai lembaga yang berperan sebagai penyelenggara jasa sistem pembayaran maupun penyelenggara pendukung jasa sistem pembayaran seperti bank, lembaga keuangan selain bank, dan bahkan perorangan (Bank Indonesia, 2008, p2).

Dalam perkembangannya, sistem pembayaran yang merupakan salah satu pilar penopang stabilitas sistem keuangan telah berkembang dengan pesat seiring dengan perkembangan teknologi. Di sisi lain, perkembangan teknologi juga telah mendorong berkembangnya alat pembayaran dari yang semula *cash based* menjadi *non cash based*. Selanjutnya, *non cash based instrument* ini telah menjadi sedemikian canggih sehingga tidak lagi berbasis kertas (*paper based*) melainkan telah berevolusi ke bentuk *paperless*. Sudah barang tentu alat pembayaran yang *paperless* membutuhkan infrastruktur teknologi tinggi dan juga suatu *legal regime* yang berbeda dari alat pembayaran berbasis kertas (Bank Indonesia, 2008, p3).

2.5 Android

Android adalah platform *open source* yang dirancang untuk perangkat *mobile phone*. Android dikembangkan oleh Google dan dimiliki oleh perusahaan bernama Open Handset Alliance. Tujuan dari diciptakannya android ini yaitu

bagaimana mempercepat inovasi pada ponsel dan menawarkan kepada pengguna sebuah kendaraan untuk mengembangkan pengalaman menggunakan ponsel yang lebih mudah dan murah (Gargenta, 2011).

Pada aplikasi android dibutuhkan beberapa aplikasi pendukung dalam pembuatannya diantaranya yaitu SQLite sebagai media penyimpanan data transaksi pemesanan pelanggan, android SDK (*Software Developemnt Kit*) untuk pengembangan aplikasi yang akan dibuat dan Bluetooth *Module* untuk mengkoneksikan antara *smartphone* android dengan *wireless* Bluetooth *printer*. Masing-masing dari aplikasi pendukung diatas akan dijelaskan pada subbab dibawah ini.

2.5.1 SQLite

Database pada android diperlukan untuk menyimpan informasi berguna adari aplikasi yang perlu bertahan bahkan ketika pengguna menutup aplikasi atau pada saat mematikan dan menyalakan kembali *smartphone* yang digunakan (Gargenta, 2001).

Penggunaan database bisa melalui *cloud* atau lokal. Pada studi kasus yang diangkat lebih diutamakan penggunaan penyimpanan secara lokal untuk dapat diakses lebih cepat meskipun tanpa koneksi internet pada saat karyawan melakukan transaksi. Untuk penyimpanan melalui *cloud* diperlukan hanya pada saat *user* melakukan *login* dan *logout* pada aplikasi agar dapat diakses kedalam *web server*.

SQLite merupakan *database* yang bersifat *open source*. Awalnya dirilis pada tahun 2000 yang dirancang untuk menyediakan cara yang mudah dan nyaman untuk mengelola data pada aplikasi tanpa biaya berlebih seperti pada

database berbayar pada umumnya. SQLite memiliki reputasi baik yang dikenal karena sangat portabel, mudah digunakan, kompak , efisien, dan dapat diandalkan untuk penggunaan *mobile device* (Owens, 2006).

Berikut ini beberapa alasan mengapa SQLite sangat cocok untuk pengembangan aplikasi kasir berbasis android:

1. *Zero configuration database*. Sama sekali tidak ada konfigurasi *database* yang harus dilakukan oleh pengembang aplikasi. Hal ini membuatnya relatif lebih mudah digunakan.
2. Tidak memiliki *server*. Tidak ada *running process* yang terjadi pada *database* SQLite dikarenakan *libraries* pada SQLite sudah mendukung fungsionalitas *database*.
3. *Database* dengan tipe *file* tunggal dikarenakan setiap aplikasi dan data yang tersimpan didalamnya dibagi kedalam "kotak pasir" sehingga keamanan dari masing-masing aplikasi terjaga dengan baik.
4. Bersifat *open source* sehingga tanpa memerlukan biaya untuk pengembangannya.

2.5.2 Software Development Kit

Android *Software Development Kit* (SDK) adalah *tools* API *(Aplication Programming Interface)* yang digunakan untuk mulai mengembangkan aplikasi pada *platform* android menggunakan bahasa pemrograman Java. Android merupakan subset perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang di *release* oleh Google. Saat ini disediakan android SDK (*Software Development Kit)* sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada platform android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi netral, android memberi

kesempatan untuk membuat aplikasi yang bukan merupakan aplikasi bawaan *smartphone*. Beberapa fitur android yang paling penting dalam pembuatan aplikasi kasir adalah :

1. Sudah berisi SQLite untuk penyimpanan data.
2. *Media Support* yang mendukung *audio*, video dan Gambar (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF), GSM Telephony.
3. Bluetooth, EDGE, 3G dan WIFI untuk koneksi dengan internet dan *wireless* Bluetooth *printer*.
4. Lingkungan *development* yang lengkap dan kaya, termasuk perangkat *emulator*, *tools* untuk *debugging*, profil dan kinerja memori, dan *plug in* untuk IDE Eclipse.

2.5.3 Bluetooth *Module*

Bluetooth merupakan fitur penting dalam pembuatan aplikasi kasir ini. Teknologi *wireless* yang dikembangkan untuk menggantikan penggunaan kabel untuk menghubungkan dengan *smartphone* android. Bluetooth memberikan akses *wireless* untuk menghubungkan antara dua perangkat yang terpisah jarak tertentu. Dalam pengerjaan aplikasi ini dibutuhkan Bluetooth *adapter* pada perangkat android untuk melakukan pekerjaan yang mendasar seperti inisiasi penemuan perangkat, permintaan daftar perangkat terikat (*paired*) menggunakan alamat MAC *address* dari *smartphone* dan menciptakan *BluetoothServerSocket* untuk mendengarkan permintaan sambungan dari perangkat lain.

2.6 *Dashboard*

Dashboard adalah sebuah tampilan visual dari informasi terpenting yang dibutuhkan untuk mencapai satu atau lebih tujuan, digabungkan dan diatur pada

sebuah layar, menjadi informasi yang dibutuhkan dan dapat dilihat secara sekilas. *Dashboard* itu sebuah tampilan pada satu monitor komputer penuh yang berisi informasi yang bersifat kritis, agar kita dapat mengetahui hal-hal yang perlu diketahui. Biasanya kombinasi teks dan grafik, tetapi lebih ditekankan pada grafik (Few, 2006:34).

2.6.1 Tujuan Penggunaan *Dashboard*

Tujuan penggunaan *dashboard* menurut Eckerson (2006:5) yaitu:

1. Mengkomunikasikan Strategi

   Mengkomunikasikan strategi dan tujuan yang dibuat oleh eksekutif kepada semua pihak yang berkepentingan sesuai dengan peran dan levelnya dalam organisasi.

2. Memonitor dan Menyesuiakan Pelaksanaan Strategi

   Memonitor pelaksanaan dari rencana dan strategi yang telah dibuat. Memungkinkan eksekutif untuk mengidentifikasi permasalahan kritis dan membuat stategi untuk mengatasinya.

3. Menyampaikan Wawasan dan Informasi ke Semua Pihak

   Menyajikan informasi menggunakan grafik, simbol, bagan dan warna yang memudahkan pengguna dalam memahami dan mempersepsi informasi secara benar.

2.6.2 Jenis dashboard

Dashboard bisa dikelompokkan sesuai dengan level manajemen yang didukungnya menurut Eckerson dan Few (Hariyanti 2008:10) yaitu:

A. *Strategic Dashboard*

1. Mendukung manajemen level strategis.
2. Informasi untuk membuat keputusan bisnis, memprediksi peluang, dan memberikan arahan pencapaian tujuan strategis.
3. Fokus pada pengukuran kinerja *high-level* dan pencapaian tujuan strategis organisasi.
4. Mengadopsi konsep *Balance Score Card.*
5. Informasi yang disajikan tidak terlalu detail.
6. Konten informasi tidak terlalu banyak dan disajikan secara ringkas.
7. Informasi disajikan dengan mekanisme yang sederhana, melalui tampilan yang *unidirectional.*
8. Tidak di desain untuk berinteraksi dalam melakukan analisis yang lebih detail.
9. Tidak memerlukan data *real time*.

B. *Tactical Dashboard*

1. Mendukung manajemen *tactical.*
2. Memberikan informasi yang diperlukan oleh analisis untuk mengetahui penyebab suatu kejadian.
3. Fokus pada analisis untuk menemukan penyebab dari suatu kondisi atau kejadian tertentu.
4. Dengan fungsi *drill down* dan navigasi yang baik.
5. Memiliki konten informasi yang lebih banyak (Analisis perbandingan, pola/tren, evaluasi kerja).
6. Menggunakan media penyajian yang “cerdas” yang memungkinkan pengguna melakukan analisis terhadap data yang kompleks.

7. Didesain untuk berinteraksi dengan data.
8. Tidak memerlukan data *real time*.

C. *Operational Dashboard*

1. Mendukung manajemen level operasional.
2. Memberikan informasi tentang aktivitas yang sedang terjadi, beserta perubahannya secara *real time* untuk memberikan kewaspadaan terhadap hal-hal yang perlu direspon secara cepat.
3. Fokus pada *monitoring* aktifitas dan kejadian yang berubah secara konstan.
4. Informasi disajikan spesifik, tingkat kedetailan yang cukup dalam.
5. Media penyajian yang sederhana.
6. *Alert* disajikan dengan cara yang mudah dipahami dan mampu menarik perhatian pengguna.
7. Bersifat dinamis, sehingga memerlukan data *real time*.
8. Didesain untuk berinteraksi dengan data, untuk mendapatkan informasi yang lebih detail, maupun informasi pada level lebih atas (*Higher Level Data*).

Tipe *dashboard* yang akan digunakan pada penelitian ini yaitu tipe operational *dashboard*. Berikut ini beberapa kelebihan menggunakan *dashboard* pada aplikasi kasir berbasis android:

a. Dapat menampilkan ukuran kinerja perusahaan. Pada dashboard laporan yang akan ditampilkan nantinya pemilik dapat melihat kinerja dari tingkat penjualan karyawan, tingkat penjualan produk, gerai terlaris, produk terlaris dan lain sebagainya.
b. Memiliki kemampuan untuk mengidentifikasi dan memperbaiki tren negatif. Hal ini dapat dilakukan karena pemilik perusahaan dapat melihat informasi yang

disajikan apabila terjadi penurunan pendapatan pada lokasi gerai tertentu, karyawan tertentu dan lain sebagainya.

c. Memiliki kemampuan untuk menghasilkan laporan terbaru dan menghemat waktu dibandingkan dengan laporan konvensional. Setiap data yang digunakan selama gerai beroperasi (bahan baku, produk, jumlah transaksi, karyawan dan gerai) masuk kedalam *web server* untuk diolah secara *real-time. Real-time* karena tidak melalui rekap data secara manual oleh pegawai kantor yang membutuhkan waktu lebih lama.

d. Dapat mengukur tingkat efisiensi data. Karena pelaporan disimpan dalam *web server*, perusahaan tidak memerlukan penyimpanan data bersifat fisik tetapi dapat langsung melihat periode penjualan yang dibutuhkan kemudian dapat mencetaknya dalam bentuk *file* .PDF.

2.7 *Unified Modelling Language*

Menurut Nugroho (2005:16), pemodelan *visual* adalah proses penggambaran informasi-informasi secara grafis dengan notasi-notasi baku yang telah disepakati sebelumnya. Notasi-notasi baku sangat penting demi suatu alasan komunikasi. Dengan notasi-notasi pemodelan yang bersifat baku komunikasi yang baik akan terjalin dengan mudah antar anggota tim pengembang sistem/perangkat lunak dan antara anggota tim pengembang dengan para pengguna. UML dapat digambarkan sebagai bahasa pemodelan visual umum untuk memvisualiasasikan, menentukan dan membangun perangkat lunak dalam sebuah dokumen.

2.7.1 *Use Case Diagram*

Use-case diagram merupakan model diagram UML yang digunakan untuk menggambarkan *requirement* fungsional yang diharapkan dari sebuah sistem. *Use-case diagram* menekankan pada "siapa" melakukan "apa" dalam lingkungan sistem perangkat lunak akan dibangun. *Use-case diagram* sebenarnya terdiri dari dua bagian besar; yang pertama adalah *use case diagram* (termasuk Gambar *use case dependencies*) dan u*se case description.*

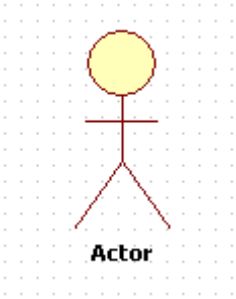
Use-case diagram adalah gambaran *graphical* dari beberapa atau semua *actor*, *use-case*, dan interaksi diantara komponen-komponen tersebut yang memperkenalkan suatu sistem yang akan dibangun. *Use-case diagram* menjelaskan manfaat suatu sistem jika dilihat menurut pandangan orang yang berada di luar sistem. Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar.

Komponen pembentuk *Use Case Diagram*:

A. ***Actor***

Pada dasarnya *actor* bukanlah bagian dari *use case diagram*, namun untuk dapat terciptanya suatu *use case diagram* diperlukan beberapa *actor*. *Actor* tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat, sistem lain) yang berinteraksi dengan sistem. Sebuah *actor* mungkin hanya memberikan informasi inputan pada sistem, hanya menerima informasi dari sistem atau keduanya menerima, dan memberi informasi pada sistem. *Actor* hanya berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case*. *Actor* digambarkan dengan *stick man*. *Actor* dapat digambarkan secara secara umum

atau spesifik, dimana untuk membedakannya kita dapat menggunakan *relationship*.

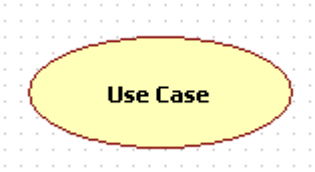


Gambar 2.7 *Actor* pada *Use Case Diagram*

B. *Use Case*

Use case adalah gambaran fungsionalitas dari suatu sistem, sehingga *customer* atau pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun.

Catatan : *Use case diagram* adalah penggambaran sistem dari sudut pandang pengguna sistem tersebut (*user*), sehingga pembuatan *use case* lebih dititik beratkan pada fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian.



Gambar 2.8 *Use Case* pada *Use Case Diagram*

2.7.2 *Activity Diagram*

Activity diagram memiliki pengertian yaitu lebih fokus kepada menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Dipakai pada *business modeling* untuk memperlihatkan urutan aktifitas proses bisnis.

Memiliki struktur diagram yang mirip *flowchart* atau *data flow diagram* pada perancangan terstruktur. Memiliki pula manfaat yaitu apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan. Dan *activity* dibuat berdasarkan sebuah atau beberapa *use case* pada *use case diagram* (Felici, 2004).

Terdapat beberapa hal penting yang harus diketahui, yaitu ;

Activity mengambarkan sebuah pekerjaan atau tugas dalam *workflow*, pada UML *activity* digambarkan dengan simbol kotak

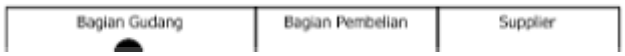
Start state dengan tegas menunjukan dimulainya suatu *workflow* pada sebuah *activity diagram*. Hanya terdapat satu *start state* dalam sebuah *workflow* serta pada UML, *start state* digambarkan dengan simbol lingkaran yang solid

End state menggambarkan akhir atau terminal dari pada sebuah *activity diagram*. Bisa terdapat lebih dari satu *end state* pada sebuah *activity diagram*. Pada UML, *end state* digambarkan dengan simbol sebuah *bull's eye*

State transition menunjukan kegiatan apa berikutnya setelah suatu kegiatan sebelumnya. Pada UML, *state transition* digambarkan oleh sebuah *solid line* dengan panah

Decision adalah suatu titik atau *point* pada *activity diagram* yang mengindikasikan suatu kondisi dimana ada kemungkinan perbedaan transisi. Pada UML, *decision* digambarkan dengan sebuah simbol *diamond*

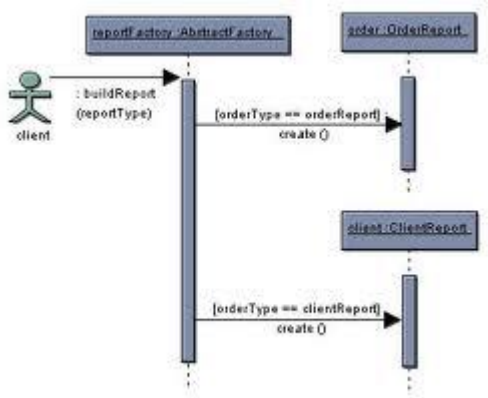
Obyek *swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu atau biasa disebut entitas pada *work flow*.



2.7.3 *Sequence Diagram*

Sequence diagram adalah interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. Gambar 2.9 merupakan contoh dari *sequence diagram*.



Gambar 2.9 *Sequence Diagram*

Sequence diagram biasanya digunakan untuk tujuan analisa dan desain, memfokuskan pada identifikasi *method* didalam sebuah *system*.

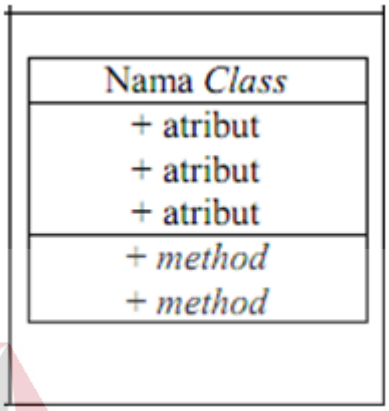
2.7.4 *Class Diagram*

kumpulan objek-objek dengan dan yang mempunyai struktur umum, *behavior* umum, relasi umum, dan *semantic*/kata yang umum. *Class-class* ditentukan/ditemukan dengan cara memeriksa objek-objek dalam *sequence diagram* dan *collaboration diagram*. Sebuah *class* digambarkan seperti sebuah bujur sangkar dengan tiga bagian ruangan. *Class* sebaiknya diberi nama menggunakan kata benda sesuai dengan domain/bagian/kelompoknya (Whitten L. Jeffery et al, 2004).

Elemen-eleman *class diagram* dalam pemodelan UML terdiri dari: *Class-class*, struktur *class*, sifat *class* (*class behavior*), perkumpulan/gabungan (*association*), pengumpulan/kesatuan (*agregation*), ketergantungan (*dependency*), relasi-relasi turunannya, keberagaman dan indikator navigasi, dan *role name* (peranan/tugas nama).

Simbol-simbol pada *class diagram*:

a. *Class* adalah blok-blok pembangun pada pemrograman berorientasi obyek. Sebuah *class* digambarkan sebagai sebuah kotak yang terbagi atas 3 bagian. Bagian atas adalah bagian nama dari *class*. Bagian tengah mendefinisikan *property*/atribut *class*. Bagian akhir mendefinisikan *method* dari sebuah *class*.



Gambar 2.10 *Class* yang berisi atribut dan *method*

b. *Association* : Sebuah asosiasi merupakan sebuah relationship paling umum antara 2 *class* dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 *class*. Garis ini bisa melambangkan tipe-tipe *relationship* dan juga dapat menampilkan hukum-hukum multiplisitas pada sebuah *relationship*.(Contoh: *One-to-one, one-to-many,many-to-many*).

c. *composition*: Jika sebuah *class* tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *Composition* terhadap *class* tempat dia bergantung tersebut. Sebuah *relationship composition* digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/solid.

d. *Dependency* : Kadangkala sebuah *class* menggunakan *class* yang lain. Hal ini disebut *dependency*. Umumnya penggunaan *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain. Sebuah *dependency* dilambangkan sebagai sebuah panah bertitik-titik.

e. *Aggregation* : *Aggregation* mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut sebagai relasi.

2.8 Konsep Dasar Sistem Penjualan

Menurut Mulyadi (2008:202), Penjualan merupakan kegiatan yang dilakukan oleh penjual dalam menjual barang atau jasa dengan harapan akan memperoleh laba dari adanya transaksi-transaksi tersebut dan penjualan dapat diartikan sebagai pengalihan atau pemindahan hak kepemilikan atas barang atau jasa dari pihak penjual ke pembeli.

Penjualan merupakan suatu seni untuk melaksanakan suatu pekerjaan melalui orang lain. Beberapa ahli menyatakan sebagai ilmu dan sebagai seni, adapula yang memasukkannya kedalam masalah etika dalam penjualan. Pada pokok istilah penjualan dapat diartikan sebagai berikut:

Menurut Phillip Kotler (2009:22) dalam bukunya yang berjudul Analisis & Desain, menjelaskan bahwa: “Konsep penjualan adalah meyakini bahwa konsumen dan perusahaan bisnis, tidak akan secara teratur membeli cukup banyak produk-produk yang ditawarkan oleh organisasi tertentu. Oleh karena itu, organisasi yang bersangkutan harus melakukan usaha penjualan dan promosi yang agresif. “

Dasar-dasar pemikiran yang terkandung dalam konsep penjualan yaitu tugas utama dari perusahaan adalah mendapatkan penjualan cukup dari produknya serta para konsumen tidak mungkin membeli barang dengan jumlah yang cukup banyak tanpa mendapat dorongan.

2.9 *Point of Sales*

Point of Sales (POS) adalah waktu dan tempat di mana transaksi ritel selesai dilakukan. Namun biasanya diartikan sebagai sebuah perangkat atau sistem yang digunakan untuk menyimpan *record* transaksi untuk perusahaan. Sebagai contoh *cash register* pada swalayan grosir, toko ritel, dan lain sebagainya (Bars, 2005).

Berdasarkan penjelasan diatas maka yang dimaksud dengan *Point of Sales* bukan sebuah mesin kasir mekanis sederhana, tetapi sebuah mesin kasir yang terhubung komputer baik dalam media penyimpanan data produk maupun transaksional. Berikut beberapa keuntungan menggunakan *Point of Sales* dibandingkan dengan tradisional *cash register*:

	POS Systems	Cash Registers
Inventory Control	Yes	No
Invoicing and Receiving	Yes	No
Interfacing with Accounting	Yes	No
Reports	Yes	Z-Out
Enforced accuracy	Yes	No
Integrated CC handling	Yes	Limited
General Ledger |	Yes	No
Accounts Payable	Yes	No
Accounts Receivable	Yes	No
Preferred by tax consultants	Yes	No

Gambar 2.11 Perbandingan Antara POS dan *Cash Register*

Komponen utama dalam POS yang umum dikenal yaitu sebuah komputer, *additional device* (*printer* struk, dsb.) dan aplikasi POS yang menyatukan kedua komponen tersebut. Penggunaan POS seperti ini cocok untuk digunakan pada toko seperti swalayan, toko grosir maupun *franchise*.

Namun perkembangan POS tidak berhenti sampai disini. Dengan penggunaan teknologi *mobile* dan sensor yang semakin banyak serta dari penyimpanan data *back-office* ke *cloud* membuat kebutuhan semakin kompleks antara penjual dan pelanggan (Ellison, 2013).

Mobile POS adalah perkembangan yang paling signifikan. Mobilitas yang tinggi merupakan alasan utama menggunakan *mobile* POS disamping bentuknya yang kecil. Berikut beberapa kelebihan menggunakan *mobile* POS:

1. Solusi Penyimpanan *Cloud*

   Keuntungan yang didapat yaitu membutuhkan lebih sedikit sumber daya TI dan mengurangi biaya dan kompleksitas yang terkait dengan POS.

2. Hemat Ruangan

   Tidak memakan tempat karena *mobile* POS umumnya dibawa bersamaan dengan *sales person* yang bertugas.

3. Tingkat kapabilitas *Smartphone* yang Tinggi

   Penggunaan *smartphone* dan *tablet* semakin banyak sehingga mudah didapat dan memiliki visual lebih baik daripada sistem POS biasa.

4. Sensor Pendukung

   Sensor pendukung seperti Wifi dan Bluetooth membuat *smartphone* atau *tablet* memiliki kompabilitas dengan perangkat tambahan seperti *mobile* printer serta sensor ini menyediakan *data stream* baru yang dapat dimanfaatkan dengan cara yang unik, mulai dari *indicator* kedekatan secara *real-time* untuk menuju niat konsumen.