

Pertemuan 1

Pengantar Rekayasa Perangkat Lunak

1.1. Definisi

Istilah rekayasa perangkat lunak (RPL) adalah produk dari dua kata, perangkat lunak, dan rekayasa. Perangkat lunak adalah kumpulan program yang terintegrasi. Perangkat lunak terdiri dari instruksi dan kode yang diatur dengan hati-hati yang ditulis oleh pengembang pada berbagai bahasa komputer tertentu. Program komputer dan dokumentasi terkait seperti persyaratan, model desain, dan manual pengguna.

Rekayasa adalah penerapan pengetahuan ilmiah dan praktis untuk menemukan, merancang, membangun, memelihara, dan meningkatkan kerangka kerja, proses, dan lain-lain. Rekayasa Perangkat Lunak adalah cabang teknik yang terkait dengan evolusi produk perangkat lunak menggunakan prinsip, teknik, dan prosedur ilmiah yang terdefinisi dengan baik secara sistematis. Hasil RPL adalah produk perangkat lunak yang efektif dan dapat diandalkan

RPL diperlukan karena alasan berikut:

1. Untuk mengelola perangkat lunak besar.
Pengembangan perangkat lunak besar memerlukan rekayasa secara ilmiah
2. Untuk Skalabilitas yang lebih besar
Lebih mudah membuat perangkat lunak baru daripada mengembangkan yang sudah ada
3. Manajemen biaya
Harga software semakin lama semakin meningkat
4. Untuk mengelola sifat dinamis perangkat lunak

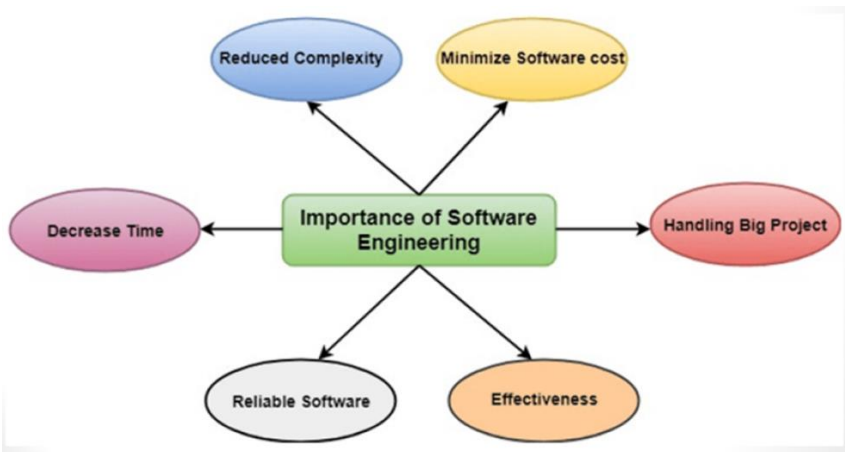
Perlu memperbarui perangkat lunak yang sudah ada

5. Untuk Manajemen kualitas yang lebih baik
Langkah yang baik menentukan kualitas perangkat lunak

Selain alasan diperlukan, RPL adalah sebuah metode yang penting karena beberapa alasan:

1. Mengurangi kompleksitas. Pekerjaan RPL dibagi menjadi tahapan pengembangan dan didistribusikan kepada pihak yang diberi kewenangan dan tanggung jawab.
2. Meminimalisir biaya. Menggunakan Analisa proyek yang membandingkan antara waktu, biaya dan sumber tenaga yang digunakan untuk pengembangan perangkat lunak.
3. Dapat digunakan untuk menangani proyek skala besar.
4. Efektif. Biaya yang digunakan sesuai dengan waktu yang diperlukan.
5. Software yang dihasilkan dapat diandalkan.
6. Mengurangi waktu pengembangan perangkat lunak

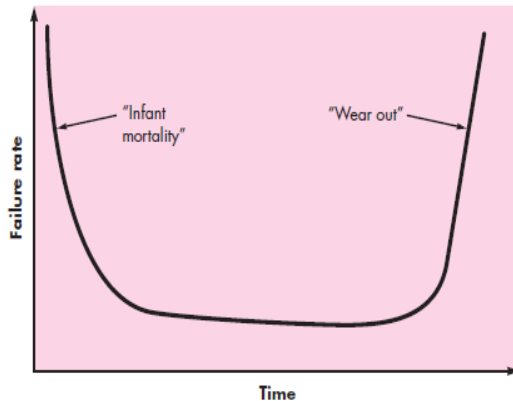
Alasan-alasan tersebut seperti digambarkan pada gambar 1.1.



Gambar 1.1. Pentingnya RPL

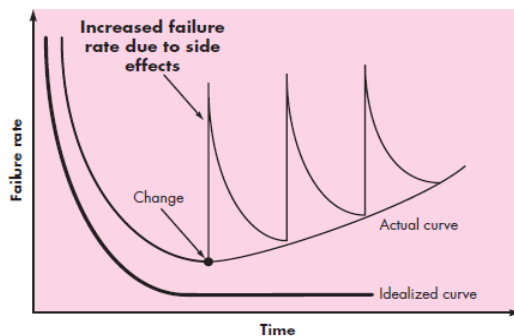
1.2. Perangkat Lunak

Software adalah sebuah produk melalui pengembangan atau rekayasa, bukan pabrikasi. Dibandingkan dengan perangkat keras, kegagalan perangkat lunak memiliki rentang yang lebih lama. Rentang kegagalan diartikan sebagai waktu sebuah perangkat digunakan. Seperti terlihat pada gambar 1.2. dan 1.3.



Gambar 1.2. Rentang waktu kegagalan perangkat keras

Rentang kegagalan perangkat keras pada awalnya dapat digunakan sebagaimana tujuan fungsinya. Pada rentang waktu tertentu perangkat lunak tersebut akan tidak dapat lagi menjalankan fungsinya (*wear out*).



Gambar 1.3. Rentang Waktu kegagalan perangkat lunak

Tidak seperti perangkat keras, rentang waktu kegagalan perangkat lunak tetap masih berfungsi selamanya (*idealized curve*). Rentang waktu perangkat lunak akan mengalami kegagalan pada saat terdapat kebutuhan baru yang muncul. Kegagalan fungsi perangkat lunak diperbaiki dengan cara mengupdate perangkat lunak sesuai kebutuhan. Contoh: perangkat lunak produk dari ms office mengeluarkan versi terbarunya bukan karena versi lama gagal berfungsi, tetapi karena ada kebutuhan baru yang muncul, seperti kebutuhan untuk terhubung dengan internet.

Berikut beberapa contoh aplikasi dari perangkat lunak:

- **Sistem Software**

(e.g., compilers, editors, and file management utilities)

- **Aplikasi Software**

(e.g., point-of-sale transaction processing, real-time manufacturing process control).

- **Engineering/Science Software**

Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing

- **Embeddes Software**

(e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

- **Product-line Software**

(e.g., inventory control products) or address mass consumer markets (e.g., word processing, preadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

- **Webapps (Web-Applications)**

WebApps can be little more than a set of linked hypertext files that present information using text and limited graphics

- **AI SoftwareWebApps**

Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing

Perkembangan ilmu komputer menghasilkan terdapat beberapa kateгоре perangkat lunak baru:

1. *Open World Computing* – komputasi yang tersebar luas dan terdistribusi
2. *Ubiquitous Computation*– jaringan nirkabel
3. *Netsourcing* – Web sebagai mesin komputasi
4. *OpenSource* – kode sumber 'gratis' membuka komunitas komputasi

Sistem lama sering berkembang karena satu atau lebih alasan berikut:

1. Perangkat lunak harus disesuaikan untuk memenuhi kebutuhan lingkungan atau teknologi komputasi baru.
2. Perangkat lunak harus ditingkatkan untuk mengimplementasikan kebutuhan bisnis baru.
3. Perangkat lunak harus diperluas agar dapat dioperasikan dengan sistem atau database lain yang lebih modern.
4. Perangkat lunak harus dirancang ulang untuk membuatnya layak dalam lingkungan jaringan.

Pengembangan aplikasi Perangkat lunak berbasis web (WebApps) perlu mempertimbangkan berdasarkan atribut berikut ditemui:

1. Intensitas jaringan.
WebApp berada di jaringan dan harus melayani kebutuhan komunitas klien yang beragam

2. Konkurensi

Sejumlah besar pengguna dapat mengakses WebApp pada satu waktu. Dalam banyak kasus, pola penggunaan di antara pengguna akhir akan sangat bervariasi

3. Beban tak terduga

Jumlah pengguna WebApp dapat bervariasi berdasarkan urutan besarnya dari hari ke hari

4. Kinerja

Jika pengguna WebApp harus menunggu terlalu lama (untuk akses, untuk pemrosesan sisi server, untuk pemformatan dan tampilan sisi klien), dia mungkin memutuskan untuk pergi ke tempat lain.

5. Ketersediaan

Meskipun ekspektasi ketersediaan 100 persen tidak masuk akal, pengguna WebApps populer sering meminta akses 24jam dalam sehari / 7 hari dalam seminggu /365 hari dalam setahun.

6. Berbasis Data

Fungsi utama dari banyak WebApps adalah menggunakan hypermedia untuk menyajikan teks, grafik, audio, dan konten video kepada pengguna akhir.

7. Sensitif Konten

Kualitas dan sifat estetika konten tetap menjadi penentu penting kualitas WebApp

8. Evolusi Berkelanjutan

WebApps (khususnya, kontennya) untuk diperbarui pada jadwal menit demi menit atau agar konten dihitung secara independen untuk setiap permintaan

9. Kesegeraan

WebApps sering menunjukkan waktu-ke-pasar yang bisa menjadi hitungan beberapa hari atau minggu

10. Keamanan

Langkah-langkah keamanan yang kuat harus diterapkan

11. Estetis

Estetika mungkin memiliki banyak kaitan dengan kesuksesan seperti desain teknis

Pertemuan 2

Belajar dari Kegagalan Pengembangan Perangkat Lunak

2.1. Survey

Dalam satu tahun, 49% dari para responden mengalami setidaknya satu kali kegagalan proyek. Pada periode yang sama, hanya 2% dari seluruh organisasi yang selalu berhasil mencapai manfaat yang ditargetkan. Sementara 86% dari seluruh organisasi yang disurvei kehilangan sampai 25% dari target keuntungan mereka dari seluruh portfolio proyek. (Zarella, 2005)

Beberapa proyek perangkat lunak yang gagal dapat dilihat pada table berikut:

Tabel 2.1. Kegagalan proyek perangkat lunak

No	Perusahaan	Kegagalan
1	FoxMeyer Corp PROYEK: Sistem ERP dari SAP	<ul style="list-style-type: none">instalasi sistem Enterprise Resource Planning (ERP) pada tahun 1996 berkontribusi padakebangkrutan FoxMeyer.Klaim gugatan hukum perusahaan distributor obat ini terhadap SAP AG,FoxMeyer berupaya menggugat US\$ 1 Milyar karena kerusakan yang diakibatkan
2	Snap-On Inc PROYEK: Konversi ke sistem order-entry baru dari The Baan Co.	<ul style="list-style-type: none">tiga tahun masa desain dan implementasi, menyebabkan perusahaan perkakas ini kehilangan penjualan sebanyak US\$ 50 jutaBiaya operasional Snap-On melonjak 40%, terutama untuk menutupi biaya pengiriman, pesanan tertunda dan persediaan mengalami salah hitungberalih ke pesaing Snap-On
3	Angkatan Udara AS proyek ERP (Oracle) setelah biayanya melampaui US\$ 1 Milyar	<ul style="list-style-type: none">biaya menggelembung melakukan terlampau banyak tambahan coding kustomisasi untuk proses integrasiproyek tersebut akan membutuhkan tambahan dana US\$ 1.1 Milyar hanya untuk menyelesaikan seperempat dari lingkup aslinya,

		<ul style="list-style-type: none"> dan proyek ini belum akan selesai setidaknya sampai 2020. (Kanaracus, 2012)
4	Pengembangan sistem manajemen kasus pengadilan di California	<ul style="list-style-type: none"> dibutuhkan tambahan dana US\$ 343 juta untuk mengimplementasikan sistem pada 11 pengadilan di seluruh California pada tahun fiskal 2020-2021. <ul style="list-style-type: none"> Keputusan menghentikan proyek tersebut alasan keuangan meskipun proyek tersebut pada akhirnya akan selesai namun akan segera ketinggalan jaman

2.2. Identifikasi kegagalan dan menghindari kegagalan proyek perangkat lunak

1. **Persyaratan:** tidak jelas, kurangnya kesepakatan, kurangnya prioritas, kontradiktif, ambigu, dan tidak tepat.
2. **Sumber Daya:** Kurangnya sumber daya, konflik sumberdaya, pergantian SDM kunci, dan perencanaan yang buruk.
3. **Jadwal:** Terlalu ketat, tidak realistis, dan terlalu optimistis.
4. **Perencanaan:** Berdasarkan data yang tidak cukup, ada hal-hal yang terlewatkan, rincian yang tidak cukup, dan perkiraan yang keliru.
5. **Risiko:** Tidak diidentifikasi atau tidak diasumsikan, serta tidak dikelola.

Kemungkinan terbesar kegagalan proyek TI:

1. Proyek dengan kompleksitas tinggi, lebih logis berpotensi gagal
2. Proyek berjangka pendek
 - 49% mengindikasikan bahwa proyek jangka pendek (< 1 tahun) gagal
 - 14 % mengindikasikan bahwa proyek jangka pendek (> 1 tahun) gagal

Sedangkan hambatan paling umum yang mengganggu pemulihan proyek yang gagal adalah:

- Mendapatkan kesepakatan dari para stakeholder untuk menerima perubahan yang dibutuhkan untuk
- membawa proyek kembali ke jalur, apakah berupa perubahan dalam ruang lingkup, anggaran, sumber daya, dll.
- Komunikasi yang buruk dan adanya campur tangan pemangku kepentingan, serta kurangnya kejelasan proyek dan kepercayaan atas pelaksanaannya.
- Konflik prioritas dan politik.
- Menemukan sumber daya berkualitas yang dibutuhkan untuk menyelesaikan proyek.
- Kurangnya proses atau metodologi untuk membantu membawa proyek kembali ke jalur.

Untuk menghindari kegagalan proyek perangkat lunak, maka perlu dilakukan:

1. Fokus pada pengelolaan strategi dan pemangku kepentingan daripada secara eksklusif berkonsentrasi pada anggaran dan jadwal.
2. Menguasai isi dari proyek dan teknologi yang digunakan dengan cara mengamankan para personil kunci, baik internal maupun eksternal.
3. Membangun tim yang efektif dengan cara menyelaraskan insentif yang akan mereka terima dengan tujuan keseluruhan dari proyek.
4. Fasih dalam menjalankan kegiatan kunci manajemen proyek, seperti siklus penyampaian yang cepat dan pengecekan kualitas yang teliti.

Tidak kalah pentingnya agar proyek perangkat lunak minim kegagalan maka perlu:

- Ruang lingkup dan persyaratan yang tidak jelas dan tidak disepakati sejak awal
- Kurangnya sumber daya yang berkualitas dan komitmen bersama seluruh pemangku kepentingan
- Jadwal yang terlalu ketat dan tidak realistis
- Perencanaan yang kurang matang dan selaras, serta tidak komprehensif
- Faktor resiko yang tidak teridentifikasi secara menyeluruh dan tidak dikelola secara baik

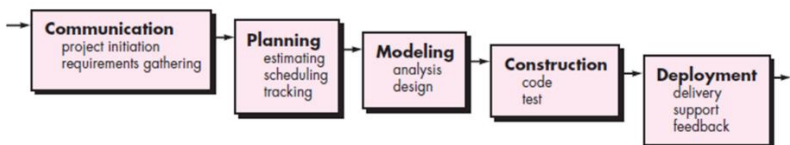
2.3. Proses Model

Proses model adalah metodologi pengembangan perangkat lunak yang diperlukan agar terhindar dari kegagalan proyek perangkat lunak. Terdapat beberapa proses model. Secara sederhana proses model terbagi kedalam dua kelompok, yaitu (1) Proses model konvensional dan (2) proses model agile (*agile development*).

1. Proses model konvensional

a. Waterfall

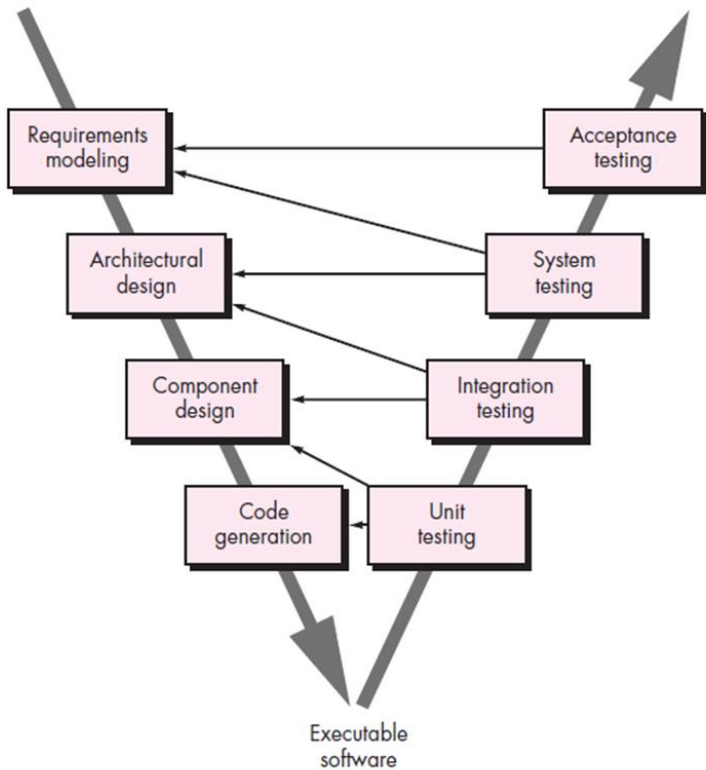
Kadang-kadang disebut siklus hidup klasik. Pendekatan yang dilakukan secara sistematis dan sekuensial untuk pengembangan perangkat lunak. Pengembangan dimulai dengan spesifikasi kebutuhan pelanggan dan berkembang melalui perencanaan, pemodelan, konstruksi, dan penerapan, yang berpuncak pada dukungan berkelanjutan dari perangkat lunak yang telah selesai (Gambar 2.1)



Gambar 2.1. Model Proses Waterfall

b. Model proses V

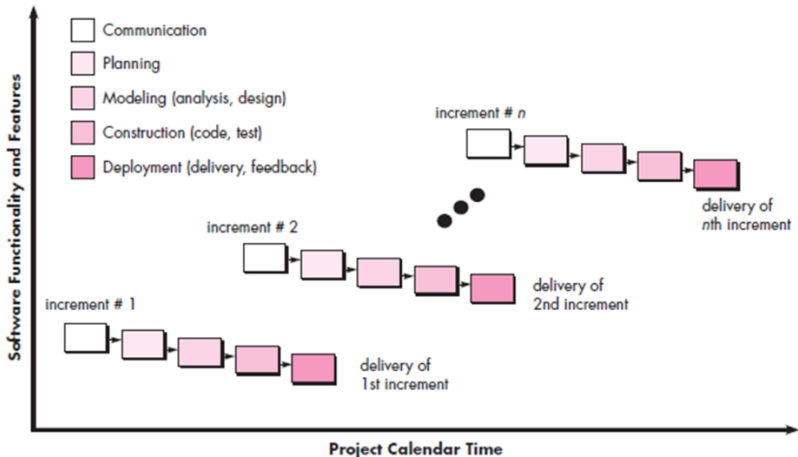
Model proses V adalah variasi dari model proses Waterfall. Model V menggambarkan relasi dari aksi yang menjamin kualitas dengan komunikasi, pemodelan, dan aktifitas konstruksi awal. Pada dasarnya tidak ada perbedaan antara model proses V dengan model proses waterfall. Model proses V menyediakan cara menggambarkan bagaimana memferifikasi dan memvalidasi aksi yang diaplikasikan untuk pekerjaan-pekerjaan engineering (Gambar 2.2).



Gambar 2.2. Model proses V

c. Model Proses Incremental

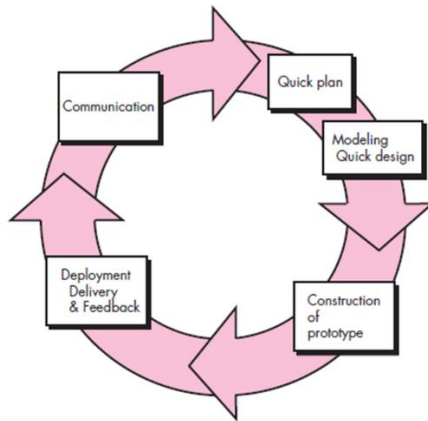
Model proses incremental mengkombinasikan elemen linier dan parallel aliran proses setiap linier sequence menghasilkan "increments" yang dapat dikirimkan dari perangkat lunak dengan cara yang mirip dengan incremented yang dihasilkan oleh aliran proses evolusioner (Gambar 2.3)



Gambar 2.4. Model proses Incremental

d. Model Proses Paradigma Prototyping

Model proses Paradigma Prototyping bersifat iterative, dicirikan dengan cara yang memungkinkan mendapatkan versi perangkat lunak yang semakin lengkap. Model proses Paradigma Prototyping dapat berfungsi sebagai "sistem pertama". Meskipun beberapa prototipe dibuat sebagai "sekali pakai", yang lain bersifat evolusioner dalam arti bahwa prototipe perlahan-lahan berkembang menjadi sistem yang sebenarnya.

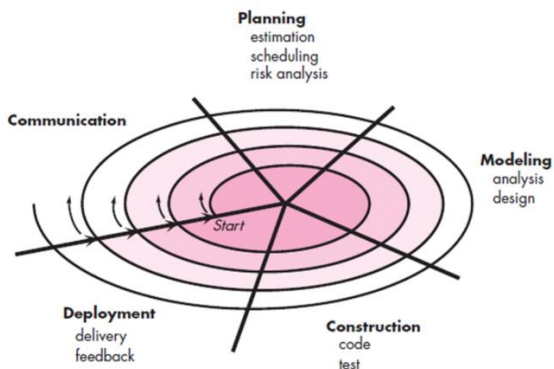


Gambar 2.5. Model proses Paradigma Prototyping

e. Model proses Spiral

Terdapat dua fitur pembeda utama pada model proses Spiral:

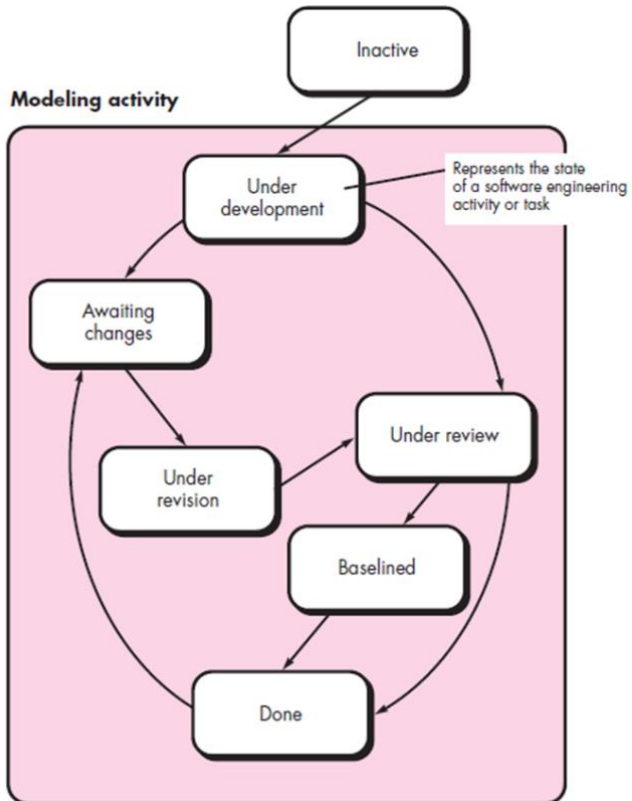
1. Salah satunya adalah pendekatan siklik untuk menumbuhkan derajat definisi dan implementasi sistem secara bertahap sambil mengurangi tingkat risikonya.
2. Yang lainnya adalah serangkaian tonggak titik jangkar untuk memastikan komitmen pemangku kepentingan agar dapat dilaksanakan dan solusi sistem yang saling memuaskan.



Gambar 2.6. Model Proses Spiral

f. Model proses Concurrent

Model proses Concurrent disebut juga sebagai concurrent engineering, yang memungkinkan tim perangkat lunak untuk merepresentasikan elemen berulang dan bersamaan dari model proses mana pun. Misalnya, aktivitas pemodelan yang ditentukan untuk model spiral diselesaikan dengan menjalankan satu atau lebih tindakan rekayasa perangkat lunak berikut: pembuatan prototipe, analisis, dan desain



Gambar 2.7. Model proses Concurrent

2. Model Proses Agile (*agile development*)

a. Extreme Programming (XP)

- b. Adaptive Software Development (ASD)
- c. SCRUM

2.4. Diskusi:

1. RPL yang baik akan berdampak baik ke siapa saja?
2. Apa dampak-dampak baik tersebut?

2.5. Tugas

1. Mencari model proses yang sekarang banyak digunakan untuk RPL
2. Mencari data terbaru kegagalan proyek software

Pertemuan 3

Model Proses Agile

3.1. Pengantar

Perlu ditekankan sebuah pernyataan bagi pengembangan perangkat lunak dengan menggunakan proses model agile yaitu, (1) Bahwa pengembangan perangkat lunak adalah kerja tim, (2) Untuk itu diperlukan komitmen setiap anggota tim menjalankan proyek perangkat lunak. Komitmen setiap anggota tim diperlukan agar memperoleh manfaat dari proses model agile: (1) Individu dan interaksi melalui proses dan alat, (2) Pekerjaan perangkat lunak melalui dokumentasi yang komprehensif, (3) Kolaborasi pelanggan melalui negosiasi kontrak, (4) Menanggapi perubahan yang mengikuti rencana.

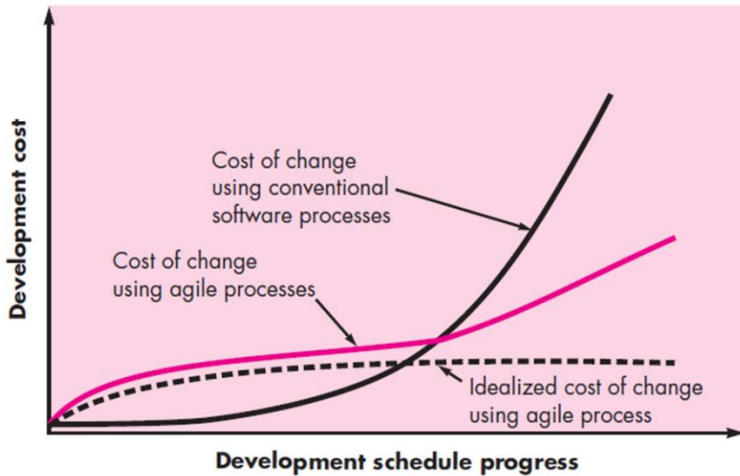
3.2. Agility

Secara singkat agility sebagai model proses pengembangan perangkat lunak digambarkan sebagai:

- Respons yang efektif (cepat dan adaptif) terhadap perubahan
- Komunikasi yang efektif di antara semua pemangku kepentingan
- Menarik pelanggan ke dalam tim
- Mengatur tim sehingga mengontrol pekerjaan yang dilakukan
- Menghasilkan perangkat lunak diselesaikan dengan cepat dan bertahap

Pengalaman pengembangan perangkat lunak menunjukkan biaya perubahan meningkat secara nonlinier seiring kemajuan proyek.

Para pendukung agility berpendapat bahwa proses agile yang dirancang dengan baik “meratakan” kurva biaya perubahan, tim perangkat lunak untuk mengakomodasi perubahan yang terlambat dalam proyek perangkat lunak tanpa dampak biaya dan waktu (Gambar 3.1).



Gambar 3.1. Perbandingan biaya pengembangan model proses konvensional dan agile

Model proses agile didorong oleh kebutuhan *customer* untuk mendapatkan perangkat lunak dengan biaya rendah dan pengerjaan proyek perangkat lunak yang tidak lama. Menyadari bahwa rencana berumur pendek, maka mengembangkan perangkat lunak secara berulang dengan penekanan berat pada aktivitas konstruksi akan memberikan beberapa peningkatan efisiensi proyek perangkat lunak. Pengembang perangkat lunak dituntut beradaptasi terhadap perubahan yang terjadi.

Dengan demikian model proses agile mencatat beberapa prinsip agility:

1. Prioritas memuaskan pelanggan melalui pengiriman awal dan berkelanjutan perangkat lunak.
2. Menerima persyaratan yang berubah, bahkan yang terlambat dalam pengembangan. Proses tangkas memanfaatkan perubahan untuk keunggulan kompetitif pelanggan.
3. Sering mengirimkan perangkat lunak yang berfungsi, dari beberapa minggu hingga beberapa bulan, dengan preferensi untuk skala waktu yang lebih pendek.
4. Pelaku bisnis dan pengembang harus bekerja sama setiap hari selama proyek berlangsung.
5. Bangun proyek di sekitar individu yang termotivasi. Beri lingkungan dan dukungan yang dibutuhkan, dan percayakan proyek dapat diselesaikan.
6. Metode yang paling efisien dan efektif untuk menyampaikan informasi kepada dan di dalam tim pengembangan adalah percakapan tatap muka.
7. Perangkat lunak yang berfungsi adalah ukuran utama kemajuan.
8. Mendorong pengembangan berkelanjutan. Para sponsor, pengembang, dan pengguna harus dapat mempertahankan kecepatan konstan tanpa batas.
9. Perhatian terus menerus pada keunggulan teknis dan desain yang baik
10. Kesederhanaan — memaksimalkan jumlah pekerjaan yang belum diselesaikan — sangat penting.
11. Arsitektur, persyaratan, dan desain terbaik muncul dari tim yang mengatur dirinya sendiri.
12. Secara berkala, tim merefleksikan bagaimana menjadi lebih efektif, kemudian menyesuaikan dan menyesuaikan perilakunya.

Faktor manusia menjadi penting, mengingat dalam model proses agile pengembangan perangkat lunak dilakukan oleh tim. Komitmen untuk melakukan setiap bagian pekerjaan oleh anggota tim menjadi pemicu efisiensi dalam bekerja. Pada akhirnya mengimbaskan kepada waktu yang dan biaya pengembangan perangkat lunak yang efektif. Proses yang harus dilakukan anggota di dalam tim adalah menyesuaikan kebutuhan tim dan bukan sebaliknya. Sehingga setiap anggota di dalam tim harus gesit yang dicirikan dengan:

- Kompeten
- Fokus pada satu tujuan
- Kolaborasi
- Kemampuan pengambilan keputusan
- Saling percaya dan hormat
- Kemampuan menyelesaikan Fuzzy (persoalan ambigu)
- Mengatur diri sendiri

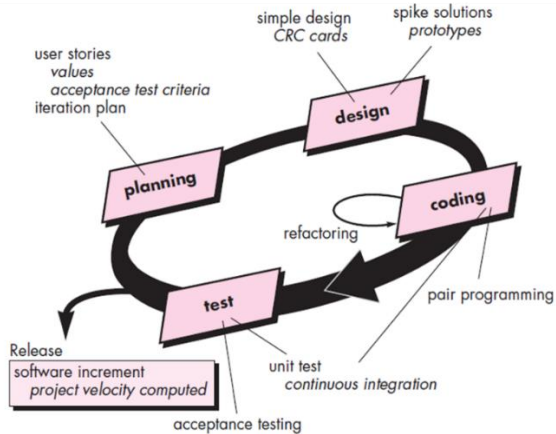
3.3. Model Proses Agile

a. Model Proses Extreme Programming (XP)

Proses Agile yang paling banyak digunakan oleh praktisi adalah model proses XP. Model proses XP awalnya diusulkan oleh Kent Beck. Beck mendefinisikan sekumpulan lima nilai yang menjadi dasar untuk semua pekerjaan yang dilakukan sebagai bagian dari XP yaitu: komunikasi, kesederhanaan, umpan balik, keberanian, dan rasa hormat.

Masing-masing nilai ini digunakan sebagai driver untuk aktivitas, tindakan, dan tugas XP tertentu.

XP menggunakan pendekatan berorientasi objek sebagai paradigma pengembangan pilihannya dan mencakup seperangkat aturan dan praktik yang terjadi dalam konteks empat aktivitas kerangka kerja: perencanaan, desain, pengkodean, dan pengujian (Gambar 3.2)



Gambar 3.2. Model proses XP

Secara singkat keempat tahapan proses XP dijelaskan sebagai berikut.

Planning:

Dimulai dari mendengarkan ‘*user story*’, yang diperoleh melalui komunikasi langsung dengan *user*. Tim Agile kemudian menilai setiap ‘dan menetapkan ‘biaya’. *User story* dikelompokkan untuk pengembangan berikutnya. Diperlukan ‘Komitmen’ setiap anggota tim untuk mentaati jadwal. Setelah rilis ‘proyek pertama’ (software increment) telah dikirimkan, tim XP menghitung kecepatan proyek.

Design:

Setiap anggota tim XP harus mengikuti prinsip KIS (*keep it simple*). Agar pekerjaan pengembang perangkat lunak menghasilkan dokumen yang terstruktur gunakan CRC (class-responsibilitycollaborator) Card. Untuk desain yang sulit, gunakan desain prototype ‘*spike solution*’. Model proses XP mendukung *Refactoring* yaitu, penyempurnaan berulang dari desain program internal.

Coding:

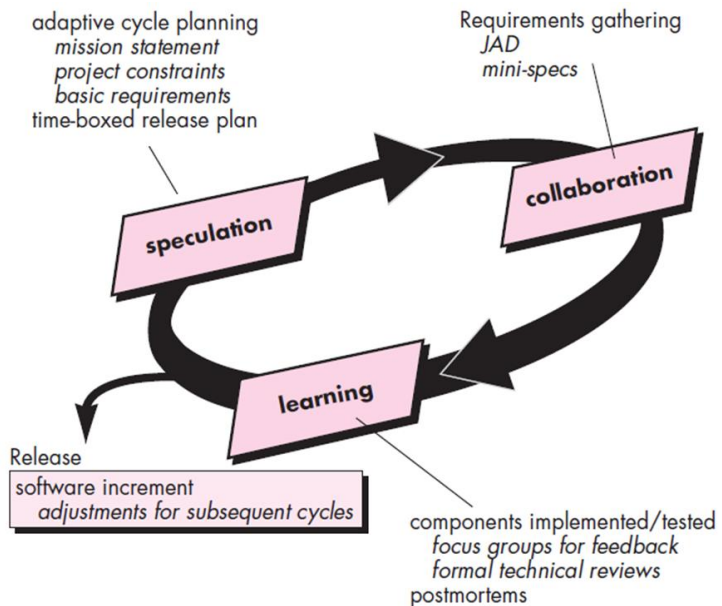
Merekomendasikan konstruksi pengujian tahapan sebelumnya (tahap *design*) sebelum pengkodean dimulai. Untuk hasil coding yang lebih baik gunakan pemrograman pasangan (agar terjadi proses kontrol).

Testing:

Semua unit test dieksekusi setiap hari. Uji penerimaan ditentukan oleh pelanggan dan dijalankan untuk menilai fungsionalitas yang dihasilkan.

b. Model proses Adaptive Software Development (ASD)

ASD dikenalkan oleh Jim Highsmith. ASD fokus pada kolaborasi antar anggota dan tim.



Gambar 3.3. Model Proses ASD

Secara singkat ketiga tahapan proses ASD dijelaskan sebagai berikut.

Spekulasi:

Proyek dimulai dan perencanaan siklus adaptif dilakukan (pernyataan misi pelanggan, kendala proyek (misalnya, tanggal pengiriman atau deskripsi pengguna), dan persyaratan dasar) — untuk menentukan rangkaian siklus yang diperlukan untuk proyek. Berdasarkan informasi yang diperoleh pada penyelesaian siklus pertama, rencana tersebut ditinjau ulang dan disesuaikan sehingga pekerjaan yang direncanakan lebih sesuai dengan kenyataan di mana tim ASD bekerja

Kolaborasi:

Orang-orang yang bekerja bersama harus saling percaya untuk (1) mengkritik tanpa permusuhan, (2) membantu tanpa kebencian, (3) bekerja sekeras atau lebih keras dari yang mereka lakukan, (4) memiliki keterampilan untuk berkontribusi pada pekerjaan yang ada, dan (5) mengomunikasikan masalah atau kekhawatiran dengan cara yang mengarah pada tindakan yang efektif.

Learning:

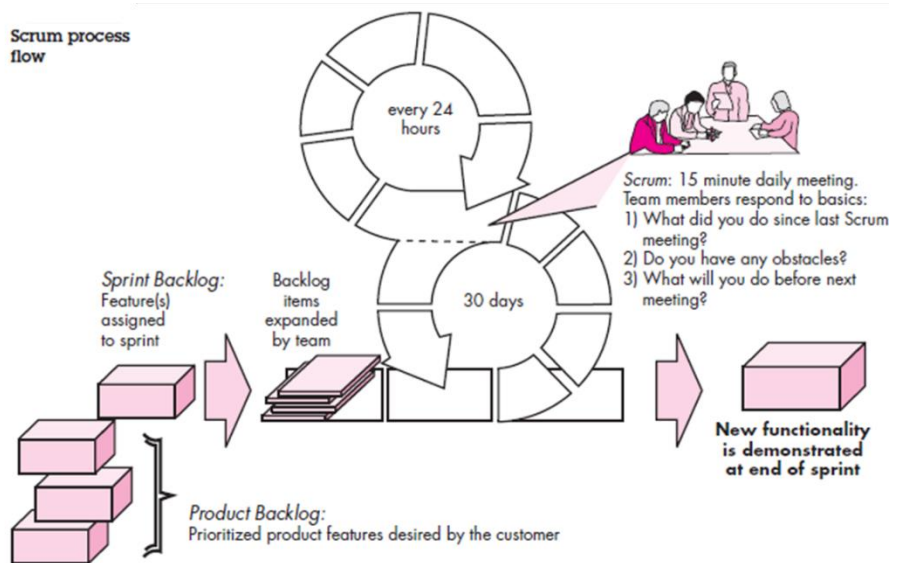
Sebagai anggota tim ASD mulai mengembangkan komponen yang merupakan bagian dari siklus adaptif, penekanannya adalah pada "belajar" sebanyak kemajuan menuju siklus selesai.

c. Model proses SCRUM

Model proses SCRUM versi asli diciptakan oleh Schawber dan Beedle. Fitur yang membedakan dengan model proses lainnya pada agile adalah:

- Pekerjaan pengembangan dibagi ke dalam 'paket'

- pengujian dan dokumentasi sedang berlangsung saat produk dibuat
- pekerjaan terjadi di 'sprint' dan berasal dari 'backlog' dari persyaratan yang ada
- rapat sangat singkat dan terkadang dilakukan tanpa kursi
- demo dikirim ke pelanggan dengan pembatasan waktu



Gambar 3.4. Model porses SCRUM

Secara singkat ketiga tahapan proses SCRUM dijelaskan sebagai berikut.

Backlog:

Mendaftar prioritas persyaratan proyek atau fitur yang memberikan nilai bisnis bagi pelanggan. Item pada produk backlog adalah user story, perubahan fungsi yang ada, dan perbaikan bug.

Sprint

Terdiri dari unit kerja yang diperlukan untuk mencapai persyaratan yang ditentukan dalam *backlog* yang harus sesuai dengan kotak waktu yang telah ditentukan

Meeting

Rapat singkat (biasanya 15 menit) yang diadakan setiap hari. Setiap hari terdapat tiga pertanyaan kunci diajukan dan dijawab oleh semua anggota tim adalah

1. Apa yang telah dilakukan sejak pertemuan tim terakhir?
2. Hambatan apa yang ditemui?
3. Apa yang direncanakan untuk dicapai pada pertemuan tim berikutnya?

3.4. Diskusi

1. Mengapa muncul model proses agile sebagai alternatif pengganti model proses konvensional?

3.5. Tugas

1. Temukan perbedaan antara model proses XP, ASD, dan SCRUM!

Pertemuan 4

Memahami dan Analisa Kebutuhan

4.1. Memahami kebutuhan

Pengembangan perangkat lunak dilakukan oleh pengembang untuk memenuhi kebutuhan pengguna. Produk perangkat lunak yang baik adalah hasil dari memahami kebutuhan pengguna. Terdapat tujuh pemahaman yang harus dipertimbangkan oleh pengembang perangkat lunak:

1. Inception

Mengajukan serangkaian pertanyaan tentang pemahaman dasar masalah orang-orang yang menginginkan solusi, sifat solusi yang diinginkan, dan efektivitas komunikasi awal dan kolaborasi antara pelanggan dan pengembang.

2. Elisitasi

Mendapatkan kebutuhan dari semua pemangku kepentingan.

3. Elaborasi

Membuat model analisis yang mengidentifikasi data, fungsi dan kebutuhan.

4. Negosiasi

Menyepakati sistem penyampaian yang realistis untuk pengembang dan pelanggan

5. Spesifikasi

dapat berupa salah satu (atau lebih) dari berikut ini:

- Dokumen tertulis
- Satu set model
- Matematika formal
- Kumpulan skenario pengguna (kasus penggunaan)
- Sebuah prototipe

6. Validasi

Sebuah mekanisme tinjauan dengan cara

- menemukan kesalahan dalam konten atau interpretasi
- menemukan area di mana klarifikasi mungkin diperlukan
- menemukan informasi yang hilang
- inkonsistensi (masalah utama bila produk atau sistem besar direkayasa)
- Kebutuhan yang bertentangan atau tidak realistis (tidak dapat dicapai).

7. Pengelolaan Kebutuhan

Serangkaian kegiatan yang membantu tim proyek mengidentifikasi, mengontrol, dan melacak persyaratan dan berubah menjadi persyaratan kapan saja saat proyek berlanjut

1. Inception

Inception merupakan Langkah awal perlu dipahami oleh tim pengembang perangkat lunak. Pada dasarnya inception adalah pekerjaan tim pengembang perangkat lunak untuk mengidentifikasi pemangku kepentingan. Identifikasi pemangku kepentingan dapat dilakukan dengan cara mengajak bicara ke 'siapapun' yang dipandang memiliki pengaruh terhadap produk pengembangan perangkat lunak. Mengajak bicara kepada 'siapapun' ini bertujuan untuk mengenali kepentingan dari berbagai sudut. Jika sudah teridentifikasi siapa saja pemangku kepentingan, tahapan berikutnya adalah mengupayakan kolaborasi antar kepentingan yang teridentifikasi. Beberapa panduan pertanyaan yang mungkin diperlukan pada saat mengidentifikasi pemangku kepentingan diantaranya:

- Siapa di balik permintaan untuk pekerjaan ini?
- Siapa yang akan menggunakan solusi tersebut?

- Apa manfaat ekonomi dari orang yang sukses larutan
- Apakah ada sumber solusi lain yang Anda miliki perlu?

2. Elisitasi

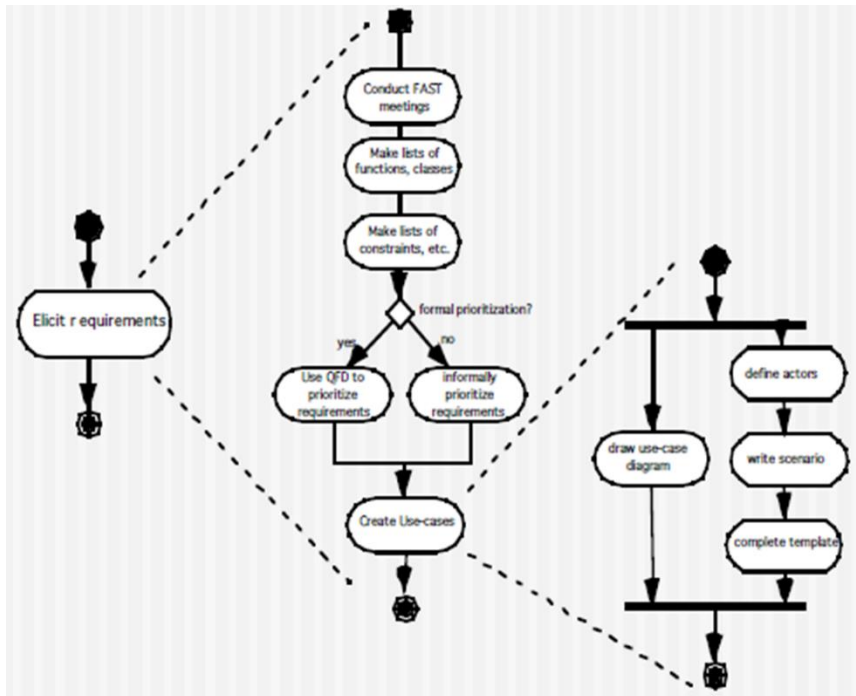
Bentuk paling sederhana dari elisitasi adalah pertemuan dilakukan dan dihadiri oleh kedua belah pihak: pengembang perangkat lunak dan pemangku kepentingan yang telah teridentifikasi pada pemahaman inception. Untuk itu aturan untuk persiapan dan partisipasi ditetapkan dan sebuah agenda disarankan. Fasilitator pertemuan (bisa dari pihak pelanggan, pengembang, atau orang luar) mengontrol rapat. Tujuannya pertemuan adalah:

- Mengidentifikasi masalah
- Mengusulkan elemen solusi
- Menegosiasikan pendekatan yang berbeda, dan
- Menentukan satu set awal kebutuhan solusi

Gambar 4.1. menunjukkan bagaimana tahapan untuk menentukan kebutuhan pemangku kepentingan. Dimulai dari pertemuan untuk mendaftar sejumlah fungsi, *class*, dan kendala yang dihadapi. Akhir pertemuan diakhiri dengan hasil menciptakan use case. Hasil Use case adalah gambar setiap actor yang melakukan fungsi.

Produk yang dihasilkan dari elisitasi kebutuhan adalah:

- Pernyataan kebutuhan dan kelayakan.
- Pernyataan ruang lingkup sistem atau produk.
- Daftar pelanggan, pengguna, dan pemangku kepentingan lainnya yang berpartisipasi dalam perolehan kebutuhan
- Deskripsi lingkungan teknis sistem.
- Daftar kebutuhan (sebaiknya diatur berdasarkan fungsi) dan batasan domain yang berlaku untuk masing-masing.



Gambar 4.1. Elisitasi Kebutuhan

- Sekumpulan skenario penggunaan yang memberikan wawasan tentang penggunaan sistem atau produk dalam kondisi operasi yang berbeda.
- Prototipe apa pun yang dikembangkan untuk menentukan persyaratan dengan lebih baik.

3. Elaborasi

Usaha untuk membangun model Analisa. Produk Analisa menggambarkan bagaimana setiap actor melakukan aktifitas di dalam sistem dan bagaimana setiap actor berhubungan aktifitasnya di dalam sistem. Model analisis memiliki beberapa elemen, diantaranya:

- Elemen berbasis scenario, terdiri dari:

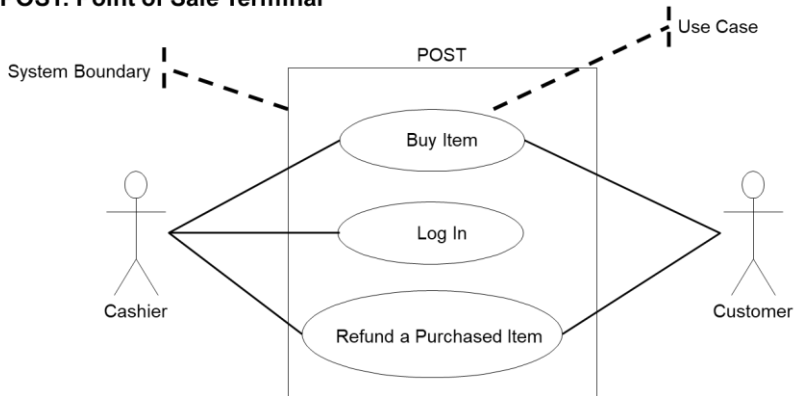
- Fungsional — memproses narasi untuk fungsi perangkat lunak
- Use-case — deskripsi interaksi antara "Aktor" dan sistem
- Elemen berbasis kelas
 - Menggunakan model Diagram Class
- Elemen perilaku
 - Menggunakan model State Diagram
- Elemen berorientasi aliran
 - Menggunakan model Data Flow Diagram

Pemodelan berbasis scenario memanfaatkan fungsi Use case untuk menggambarkan interaksi antara actor dengan sistem. Use case adalah sekumpulan skenario pengguna yang mendeskripsikan urutan penggunaan system. Setiap skenario dijelaskan dari sudut pandang seorang "aktor" — orang atau perangkat yang berinteraksi dengan perangkat lunak dengan cara tertentu. Setiap skenario menjawab pertanyaan-pertanyaan berikut:

- Siapakah aktor utama, aktor sekunder ?
- Apa tujuan aktor ?
- Prasyarat apa yang harus ada sebelum cerita dimulai ?
- Apa tugas atau fungsi utama yang dilakukan oleh aktor ?
- Ekstensi apa yang mungkin dianggap sebagai cerita yang dijelaskan ?
- Variasi apa dalam interaksi aktor yang mungkin ?
- Sistem informasi apa yang akan diperoleh, diproduksi, atau diubah oleh aktor ?
- Apakah aktor harus menginformasikan sistem tentang perubahan eksternal lingkungan Hidup?
- Informasi apa yang diinginkan aktor dari sistem ?
- Apakah aktor ingin diberi tahu tentang perubahan yang tidak terduga ?

Contoh use case gambar 4.2

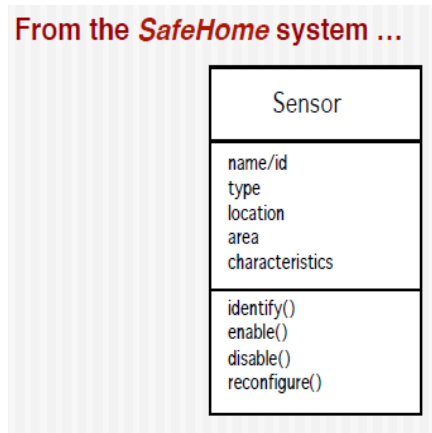
POST: Point of Sale Terminal



Gambar 4.2. Use Case Point of Sale Terminal

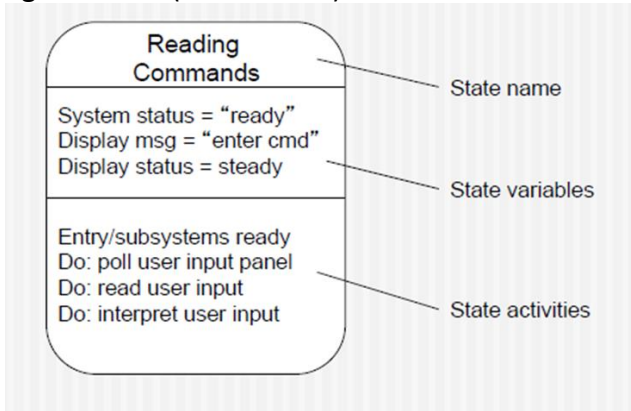
Diagram class

Contoh diagram class gambar 4.3



Gambar 4.3. Diagram class SafeHome System

Contoh Diagram State (Gambar 4.4)



Gambar 4.4. Diagram State

Pertemuan 5

User Story pada XP

5.1. Pengantar XP

XP adalah salah satu model proses kategori agile development. Langkah-langkah pengembangan perangkat lunak menggunakan model proses XP adalah *Planning*, *Design*, *Coding*, dan *Testing*. Dua tahapan memiliki model untuk memudahkan pemahaman bagi pihak pengembang perangkat lunak yaitu tahap *planning* dan *design*. Tahap *planning* memiliki model pengembangan 'user story'. Tahap *design* memiliki model pengembangan 'Colaboration Responsibility Class' (CRC). Tahap *Coding* memil

5.2. Use Story

Use Story ditulis dari sudut pandang pelanggan. Digunakan untuk mengembangkan pemahaman bersama antara pengembang dengan

pelanggan tetapi tidak begitu diperlukan untuk memandu desain dan pengembangan. Untuk membuat kemajuan, setiap user story harus dibagi menjadi tugas. Setiap tugas kemudian membutuhkan perkiraan yang terkait dengannya. Seluruh tim harus berpartisipasi dalam memecahkan user story ke dalam tugas; perencanaan harus digunakan untuk menetapkan perkiraan dari sudut pandang pelanggan.

User Story adalah unit kerja untuk mengembangkan fungsionalitas yang:

- Sangat spesifik (memiliki contoh konkrit)
- Memberikan nilai kepada pelanggan
- Dapat diuji secara independen dari yang lain (nantifitur)
- Dapat diselesaikan dalam satu iterasi

User story memiliki tiga komponen utama, yaitu:

1. Story Card adalah bagan yang digunakan untuk menuliskan materi user story

Nama		
<i>User Story</i>		<i>Priority</i> _____

Sebagai		<i>Size</i> _____

Saya ingin		

Sehingga		

Gambar 5.1 Story Card

Contoh: User Story pada Story Card

User Story : **Pendaftaran Pasien Klinik**

Sebagai : Direktur Bagian Klinik di RS

Saya ingin: Pendaftaran Pasien Klinik bisa mendaftar melalui aplikasi

Sehingga: Pasien tidak perlu langsung datang ke klinik RS untuk mendaftar

Catatan:

Prioritas dan Size adalah catatan yang digunakan untuk mengurutkan tingkat penting dari user story dan seberapa invest yang diperlukan untuk mengerjakan user story tersebut.

2. Conversation, Catatan yang diperoleh dari pengembangan story card

Contoh: melanjutkan diskripsi user story pada srotly card

Conversation terjadi antara pihak pengembang software (PS) dengan direktur bagian Klinik RS (dir)

PS	dir
Apakah pendaftar harus dilakukan oleh pasien sendiri?	Ya, sebelum mendaftar harus registrasi terlebih dulu
Data apa saja yang diperlukan untuk pendaftaran?	<ul style="list-style-type: none">- Pada saat registrasi, data yang diperlukan: nama, tgl. Lahir, tempat lahir, Id, L/P- Saat mendaftar data yang diperlukan : tujuan klinik, tanggal periksa, dst.

3. Story Tests, pengujian terhadap catatan hasil Conversation
Contoh:

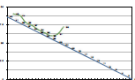
- Bagaimana jika member mendaftarkan orang lain sebagai pasien?
- Apakah bisa terdeteksi jika pendaftar mengisi 2 slot jadwal waktu pemeriksaan sekaligus?

6.2. Papan Besar

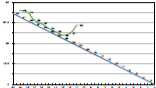

Papan Besar digunakan untuk mencatat progress sebuah user story. Papan Besar adalah fitur utama dari ruang kerja tim. Papan Besar diperbarui setidaknya sekali sehari selama pertemuan. Tapi bisa berguna untuk memperbaruinya lebih sering dari itu.

Menyusun user story pada Papan besar dapat digambarkan sebagai berikut:


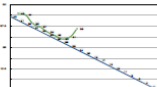


1. Dimulai dari meletakkan setiap user story di bagian bawah tengah began. (Gambar 5.2)
2. Setiap User story kemudian dipindah ke bagian atas samping kiri dan mempersiapkan indikator pencapaian untuk setiap komponen. (Gambar 5.3)
3. Tempatkan ke setiap indikator di dalam user story (gambar 5.4)
4. Pindahkan indicator untuk progress dan selesainya setiap user case (Gambar 5.5a dan 5.5b)
5. Pindahkan ke kolom Next jika User story terjadi tabrakan (Gambar 5.6)
6. Jika User Story sudah selesai pindahkan ke bagian 'complete' (Gambar 5.7)

User Stories	In Progress	Complete	Burn Down
			
			Next
	Start with this...		
	<div data-bbox="408 1037 476 1077"> <p>Title: Story Current Done</p> <p>Description: The website will allow customers to search for items in online catalog.</p> </div> <div data-bbox="492 1037 560 1077"> <p>Title: Story package</p> <p>Description: An Online Catalog will allow users to view, search, package and receive online.</p> </div> <div data-bbox="576 1037 644 1077"> <p>Title: Story online</p> <p>Description: An Online Catalog will allow users to view, search, package and receive online.</p> </div>	Completed	

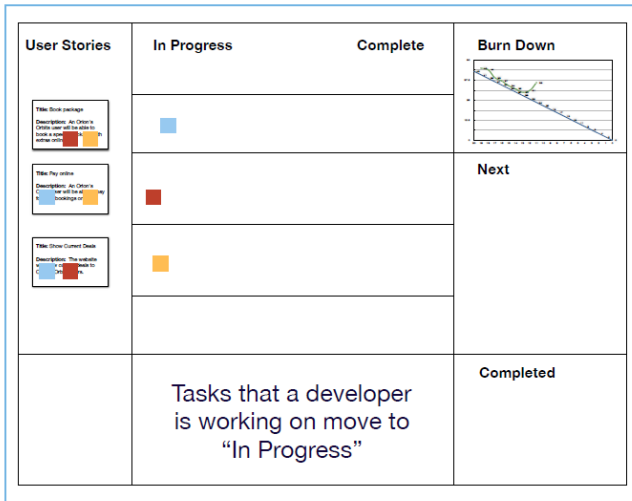
Gambar 5.2

User Stories	In Progress	Complete	Burn Down
<p>Title: Book package</p> <p>Description: As a user, I can buy a book package with one click.</p>			
<p>Title: Pay online</p> <p>Description: As a user, I can pay for my book package online.</p>	And add user stories		
<p>Title: Show Current Deals</p> <p>Description: The website will show current deals to users.</p>	one per swim lane		
			Completed

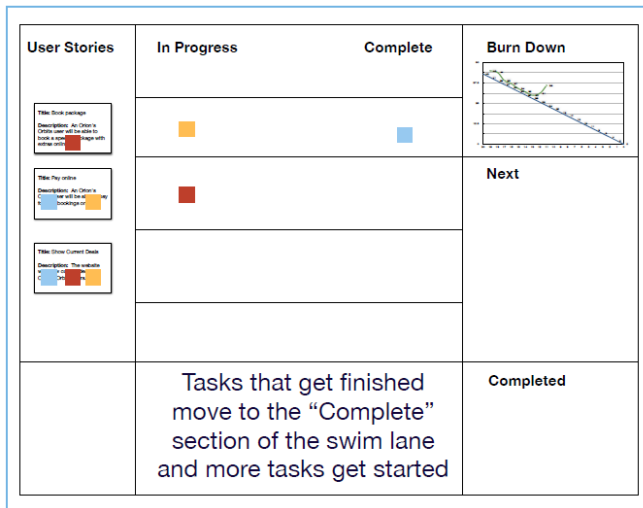
Gambar 5.3

User Stories	In Progress	Complete	Burn Down
<p>Title: Book package</p> <p>Description: As a user, I can buy a book package with one click.</p> 			
<p>Title: Pay online</p> <p>Description: As a user, I can pay for my book package online.</p> 			
<p>Title: Show Current Deals</p> <p>Description: The website will show current deals to users.</p> 			
	Now add tasks each with a description and estimate		Completed

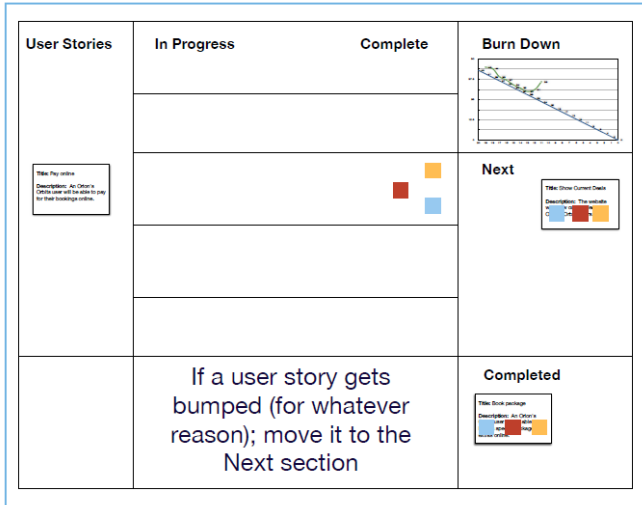
Gambar 5.4



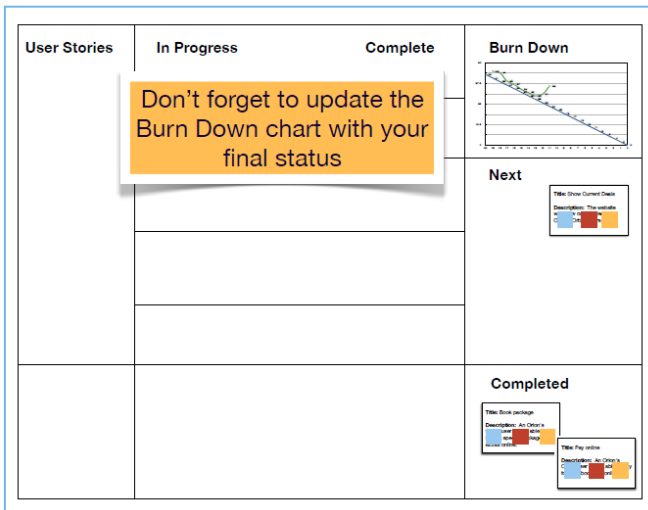
Gambar 5.5a



Gambar 5.5b



Gambar 5.6



Gambar 5.7

Pertemuan 6

CRC (Class Responsibilities Collaborator)

6.1. CRC Card

CRC Card adalah kartu indeks standar yang telah dibagi menjadi tiga bagian yaitu, Class, Responsibilities, dan Collaborators (Gambar 6.1)

Class Name	
Responsibilities	Collaborators

Gambar 6.1. CRC Card

Class mewakili kumpulan objek yang serupa. Objek adalah orang, tempat, benda, peristiwa, konsep, layar, atau laporan yang relevan dengan sistem yang ada.

Responsibility adalah segala sesuatu yang diketahui atau dilakukan oleh class. Misalnya, pelanggan memiliki nama, nomor pelanggan, dan nomor telepon. Ini adalah hal-hal yang diketahui pelanggan. Pelanggan juga order produk, membatalkan order, dan melakukan pembayaran. Ini adalah hal-hal yang dilakukan pelanggan. Hal-hal yang diketahui dan dilakukan oleh suatu Class merupakan Responsibility. Responsibility ditampilkan di kolom kiri kartu CRC.

Terkadang Class akan memiliki Responsibility untuk dipenuhi, tetapi tidak memiliki cukup informasi untuk melakukannya. Ketika ini

terjadi, Class harus berkolaborasi (Collaboration) dengan Class lain untuk menyelesaikan pekerjaan.

Misalnya, objek Order memiliki Responsibility untuk menghitung total order.

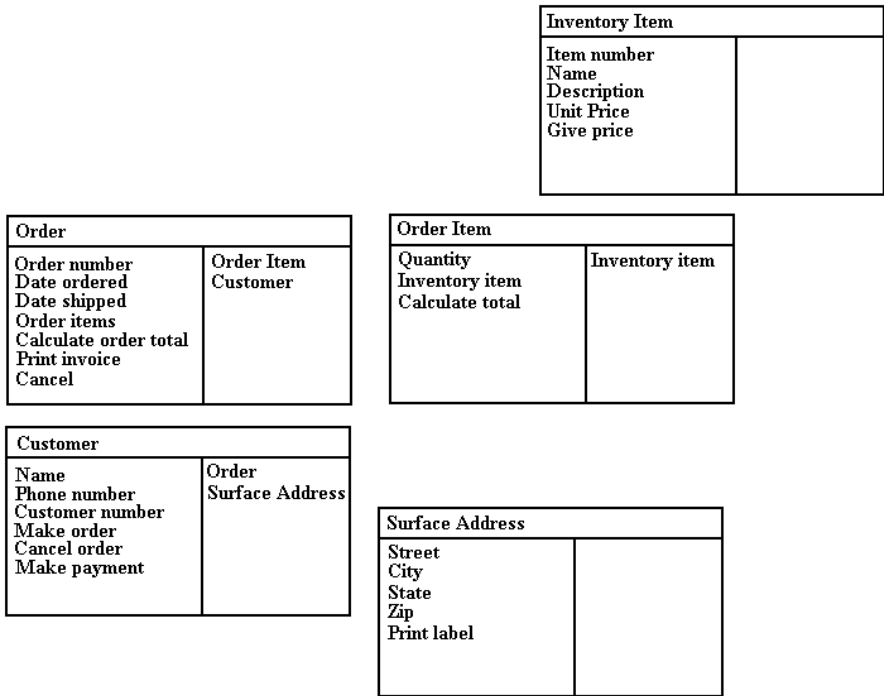
Meskipun mengetahui tentang objek Item Order yang merupakan bagian dari Order, Order tidak mengetahui berapa banyak item yang dipesan (Item Order mengetahui hal ini) juga tidak mengetahui harga item (Item Inventaris mengetahui hal ini).

Contoh:

Untuk menghitung total order, objek Order berkolaborasi dengan setiap objek Item Order untuk menghitung totalnya, lalu menjumlahkan semua total untuk menghitung total keseluruhan. Untuk setiap Item Order untuk menghitung total individualnya, ia harus berkolaborasi dengan Item Inventaris untuk menentukan biaya item yang dipesan, mengalikannya dengan jumlah yang dipesan (yang memang diketahui)

Model CRC adalah kumpulan CRC Card yang mewakili keseluruhan atau sebagian dari aplikasi atau domain masalah. Penggunaan yang paling umum untuk model CRC adalah untuk mengumpulkan dan menentukan kebutuhan pengguna untuk aplikasi berorientasi objek. Contoh model CRC untuk sistem kontrol pengiriman / inventaris, yang menunjukkan CRC Card.

Perhatikan penempatan kartu: Kartu yang berkolaborasi saling berdekatan, kartu yang tidak berkolaborasi tidak saling berdekatan



Gambar 6.2. CRC Model

6.2. Membangun CRC

Enam Langkah membangun CRC:

1. Kumpulkan tim model CRC.
2. Mengatur ruang modeling.
3. Lakukan brainstorming.
4. Menjelaskan teknik pemodelan CRC.
5. Lakukan langkah-langkah pemodelan CRC secara iteratif.
6. Lakukan pengujian skenario use-case.

1. Mengumpulkan Tim CRC

Terdapat 4 peran yang berbeda di dalam tim CRC:

1. Business Domain Expert (BDEx). Ini adalah pengguna sebenarnya dari sistem dan terkadang senior pengembang dengan pengalaman bertahun-tahun dalam domain masalah
2. Fasilitator. Ini adalah orang yang menjalankan sesi
3. Scribe(s). Juru tulis mencatat bisnis terperinci logika yang tidak ditangkap pada kartu CRC.
4. Pengamat. Peran ini duduk di belakang ruangan dan tidak berpartisipasi dalam sesi.

2. *Mengatur ruang modeling.*

Beberapa hal yang perlu dilakukan:

1. Cadangan ruang pertemuan yang memiliki sesuatu untuk ditulis. Papan tulis untuk menulis brainstorming dan prototyping. Gunakan kertas dan flip.
2. Bawa perlengkapan pemodelan CRC. Anda akan membutuhkan beberapa paket kartu indeks dan beberapa papan tulis penanda.
3. Memiliki meja model. Harus ada meja besar untuk orang bekerja.
4. Siapkan kursi dan meja untuk juru tulis. Letakkan di belakang atau di samping ruangan tempat mereka keluar jalan tapi masih bisa melihat apa yang terjadi.
5. Memiliki kursi untuk BDEx.
6. Memiliki kursi untuk pengamat. Jika ada pengamat, letakkan di belakang ruangan.

3. *Brainstorming*

Brainstorming adalah teknik pembuatan ide, akan digunakan tim untuk mengidentifikasi dan memahami persyaratan untuk aplikasi yang mereka bangun. Dilakukan untuk membangun ide-ide orang lain. Pertanyaan potensial yang dapat digunakan untuk menghasilkan ide selama brainstorming meliputi:

- Untuk siapa sistem ini?
- Apa yang akan mereka lakukan dengan sistem?
- Mengapa kita melakukan ini?
- Mengapa kita melakukan ini seperti yang kita lakukan?
- Kebutuhan bisnis apa yang didukung oleh sistem ini?

Pertanyaan-pertanyaan lainnya dapat dibangun sesuai dengan situasi yang berkembang.

4. *Menjelaskan teknik pemodelan CRC.*

Setelah brainstorming selesai, fasilitator harus menjelaskan proses pemodelan CRC. Ini biasanya memakan waktu antara sepuluh dan lima belas menit dan akan sering mencakup pembuatan beberapa contoh kartu CRC. Karena orang belajar paling baik dengan melakukannya, adalah ide yang sangat baik bagi fasilitator untuk memimpin BDE melalui pembuatan contoh kartu CRC.

5. *Pemodelan CRC secara iterative*

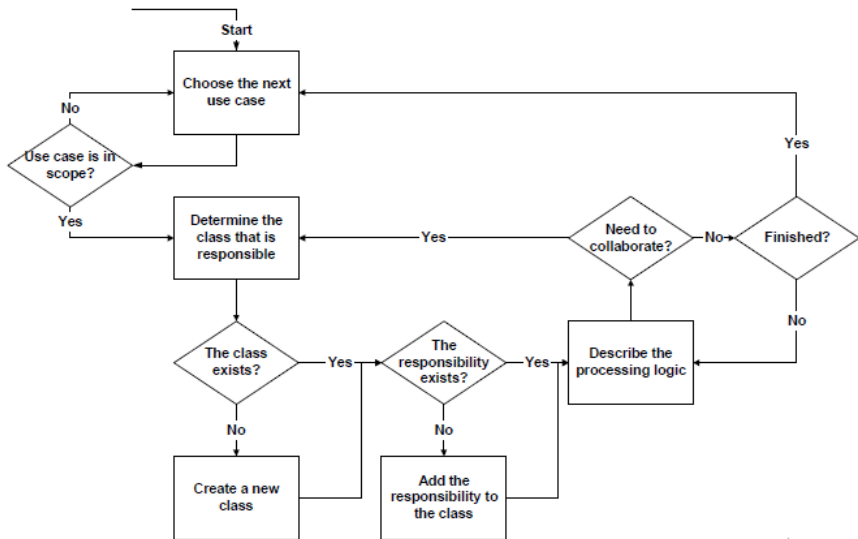
Group BDEx mengelilingi meja besar dan mengisi kartu CRC. Langkah-langkah pemodelan CRC adalah:

- Temukan class
- Temukan responsibility
- Tentukan collaborator
- Tentukan use-case

· Susun kartu di atas meja

6. Pengujian skenario use-case

Pengujian skenario use-case adalah pola proses tugas, digambarkan pada Gambar 6.3, di mana pengguna secara aktif terlibat dengan memastikan bahwa persyaratan pengguna adalah tepat.



Gambar 6.3. pengujian skenario use case

Pertemuan 7

Memodelkan Kebutuhan

7.1. Analisa Kebutuhan

Analisa kebutuhan adalah kegiatan (1) menspesifikasi karakteristik operasional perangkat lunak, (2) menunjukkan antarmuka perangkat lunak dengan elemen sistem lain (3) menetapkan kendala yang harus dipenuhi perangkat lunak.

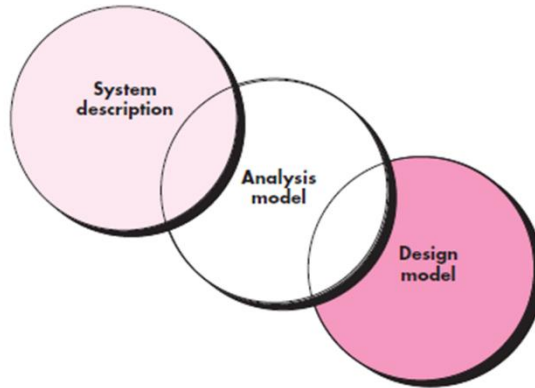
Analisis kebutuhan memungkinkan (A software engineer, analis, atau pemodel: (1) untuk menguraikan kebutuhan dasar yang ditetapkan selama Insepsi dan elisitasi, (2) tugas negosiasi merupakan bagian dari rekayasa kebutuhan.

Pemodelan kebutuhan menghasilkan satu atau lebih jenis model berikut:

- Model kebutuhan berbasis skenario dari berbagai sudut pandang "aktor"
- Model data yang menggambarkan domain informasi untuk masalah tersebut
- Model berorientasi class yang mewakili class berorientasi objek (atribut dan operasi) dan cara class berkolaborasi untuk mencapai kebutuhan sistem
- Model berorientasi flow yang merepresentasikan elemen fungsional dari sistem dan bagaimana mereka mengubah data saat bergerak melalui sistem
- Model perilaku yang menggambarkan bagaimana perangkat lunak berperilaku sebagai konsekuensi "event" eksternal.

7.2. Model Kebutuhan

Salah satu model kebutuhan pada pengembangan perangkat lunak adalah Aturan Analisa Thumb (Gambar 7.1)



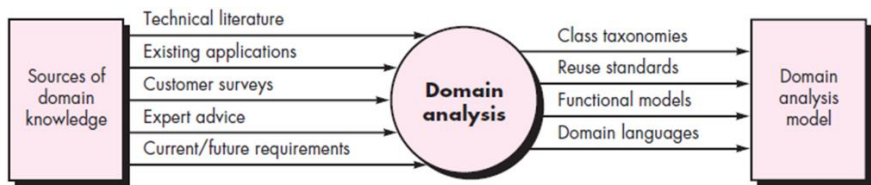
Gambar 7.1. Analisa Thumb

Aturan Analisa Thumb menyatakan:

- Model harus fokus pada kebutuhan
- Setiap elemen model kebutuhan harus menambah pemahaman tentang domain informasi, fungsi, dan perilaku sistem.
- Tunda pertimbangan infrastruktur dan model nonfungsional lainnya hingga desain selesai
- Minimalkan kopling di seluruh sistem. Kurangi keterkaitan antara *class* dan fungsi
- Pastikan model kebutuhan memberikan nilai bagi semua pemangku kepentingan
- Buat modelnya sesederhana mungkin

Untuk analisa domain perlu dipertimbangkan beberapa hal berikut (Gambar 7.2):

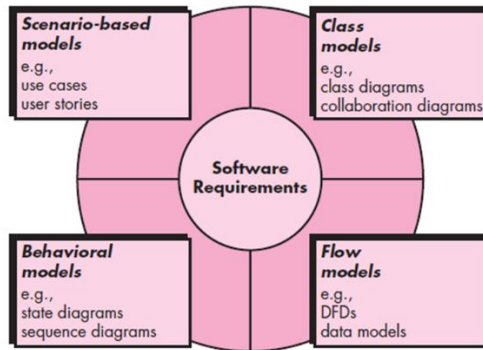
- Tentukan domain yang akan dianalisa
- Kumpulkan contoh aplikasi yang representatif di domain
- Analisis setiap aplikasi dalam sampel
- Kembangkan model analisis untuk objek



Gambar 7.2. Analisa Domain

Terdapat dua elemen Analisa kebutuhan (Gambar 7.3) :

1. Pemodelan kebutuhan, yang disebut analisis terstruktur, menganggap data dan proses yang mengubah data sebagai entitas terpisah. Objek data dimodelkan dengan cara mendefinisikan atribut dan hubungannya
2. Pendekatan kedua untuk pemodelan analisis, yang disebut analisis berorientasi objek, berfokus pada definisi class dan cara mereka berkolaborasi satu sama lain untuk memenuhi kebutuhan user.



Gambar 7.3. Elemen Analisa Kebutuhan

7.3. Pemodelan Skenario

Pemodelan berbasis skenario menggunakan Use-Case untuk mendefinisikan actor dan apa yang dikerjakan di dalam system. Pertanyaan-pertanyaan diperlukan untuk membantu use case membentuk skenario, diantaranya:

1. Apa yang akan kita gambarkan
2. Seberapa banyak kita akan gambarkan?
3. Seberapa rinci kita deskripsikan?
4. Bagaimana kita mengelola diskripsi tersebut?

Untuk menjawab pertanyaan tersebut maka dilakukan pemodelan:

1. Insepsi dan Elisitasi – menyediakan informasi yang dibutuhkan melalui Use-Case

Koleksi kebutuhan digunakan untuk:

- a. Identifikasi Stakeholder
- b. Mendefinisikan cakupan masalah
- c. Spesifikasi tujuan operasional
- d. Menentukan prioritas
- e. Mendaftar kebutuhan yang diketahui

f. Menjelaskan obyek yang akan dimanipulasi sistem

Untuk memulai mengembangkan Use case, daftar semua fungsi atau aktifitas yang dilakukan oleh *actor*

2. Seiring kemajuan komunikasi antar pemangku kepentingan, tim pengumpul kebutuhan mengembangkan Use Case untuk setiap fungsi. Secara umum, Use Case ditulis pertama kali dengan gaya naratif informal. Dalam formalitas yang lebih diperlukan, use case yang sama ditulis ulang menggunakan formal terstruktur yang serupa dengan yang diusulkan.

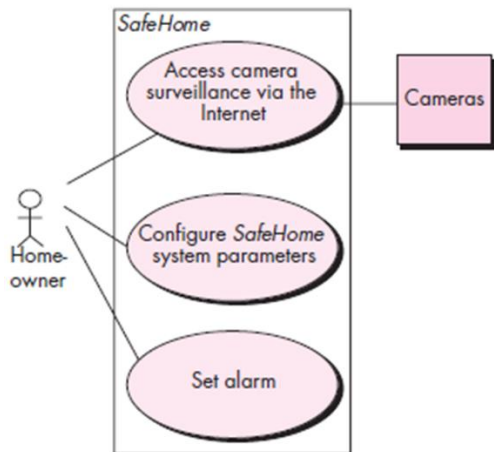
3. Rincian Use-Case

Secara sederhana Use=Case adalah skenario yang menggambarkan "urutan penggunaan" untuk sistem aktor mewakili peran yang dimainkan orang atau perangkat sebagai fungsi system. Pengguna dapat memainkan sejumlah peran berbeda untuk skenario tertentu. Beberapa pertanyaan berikut membantu menyusun Use-Case:

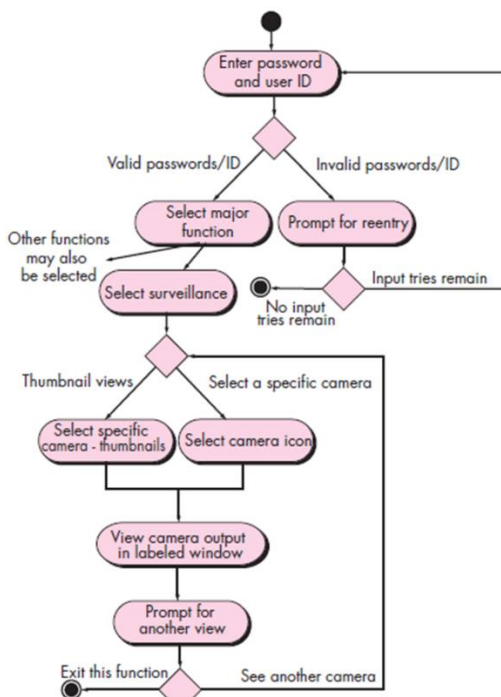
- Apa tugas utama atau fungsi yang dilakukan oleh Aktor?
- Sistem informasi apa yang akan diperoleh, diproduksi atau diubah oleh aktor?
- Apakah aktor harus menginformasikan sistem tentang perubahan lingkungan eksternal?
- Informasi apa yang diharapkan Aktor dari sistem?
- Apakah aktor perlu diberi informasi perubahan yang tidak diharapkan?

4. Mengelola Use-Case

Berikut contoh bagaimana memodelkan kebutuhan dengan Use - Case, Diagram Aktifitas, Diagram Swimlane.

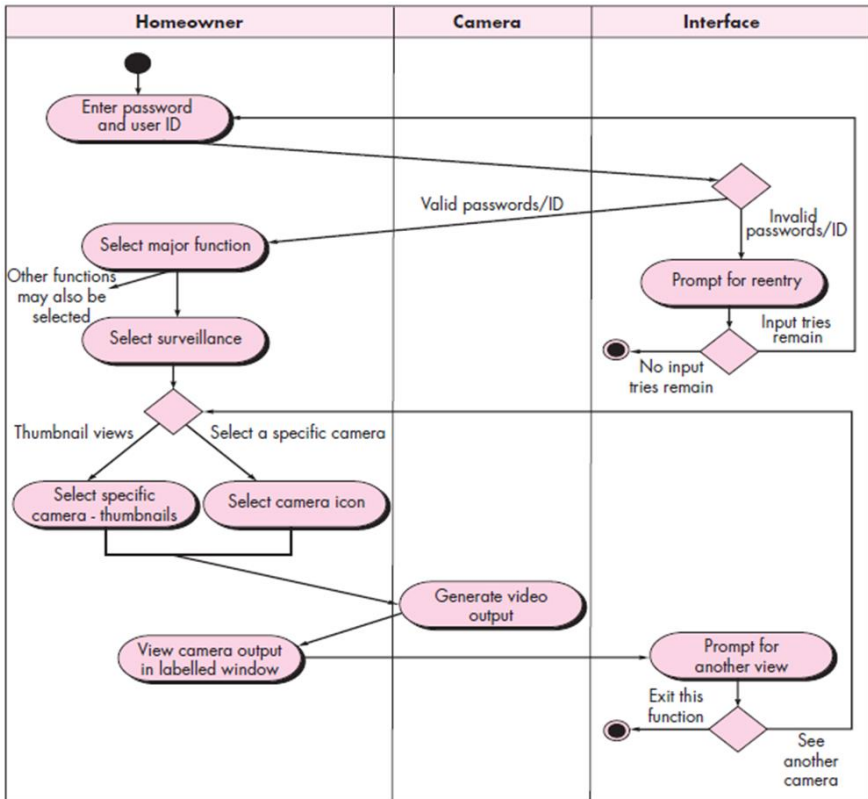


Gambar 7.4. Contoh Use Case



Gambar 7.5. Diagram Aktifitas

Diagram Aktifitas melengkapi Use-Case dengan memberikan representasi grafis dari aliran interaksi dalam skenario tertentu.



Gambar 7.6. Swimlane

Diagram swimlane memungkinkan pemodel untuk mengalirkan aktivitas yang dijelaskan oleh use case sekaligus menunjukkan aktor mana (jika ada beberapa aktor yang terlibat dalam use case tertentu) atau analisis class memiliki tanggung jawab untuk tindakan yang dijelaskan oleh segitiga aktivitas

Pertemuan 8

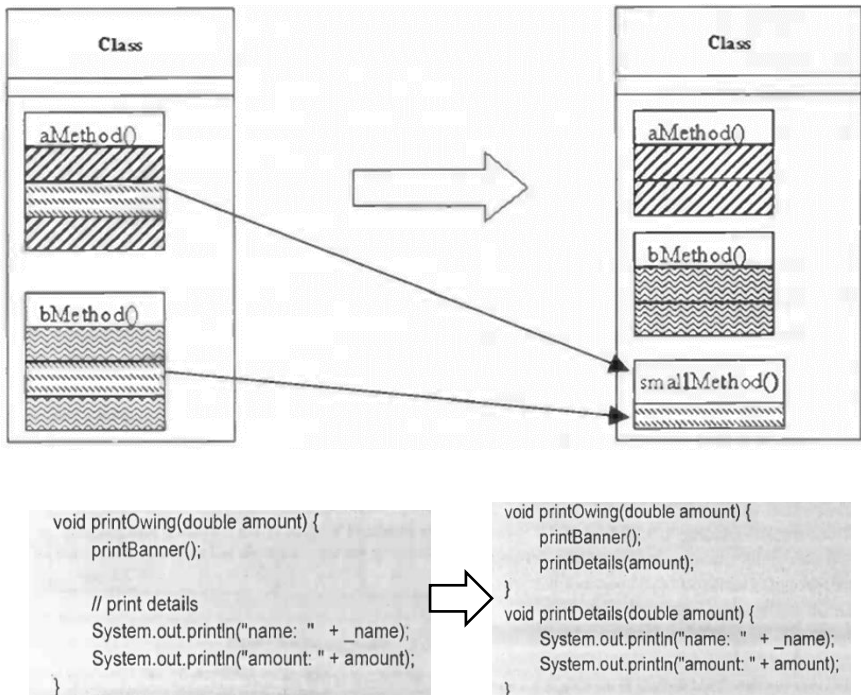
Refactoring

8.1. Definisi

Refactoring adalah proses restrukturisasi sistem perangkat lunak tidak mengubah perilaku kode yang dapat diamati meningkatkan struktur internalnya, sehingga kode menjadi lebih mudah dipahami dan lebih murah untuk dimodifikasi.

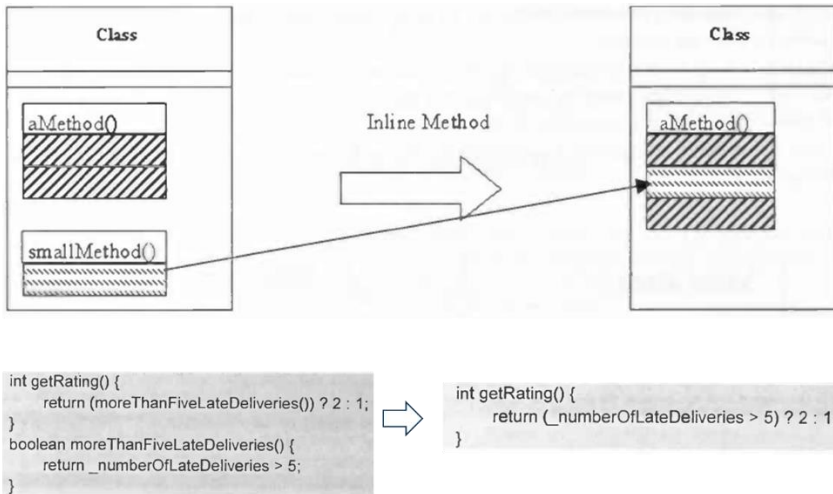
Contoh:

1. Metode Ekstraksi class



Gambar 8.1 Metode Refactoring Ekstraksi Class

2. Metode In Line

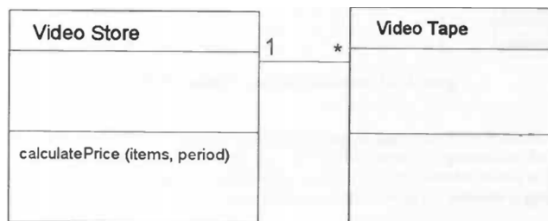


Gambar 8.2. Metode Refactoring In Line

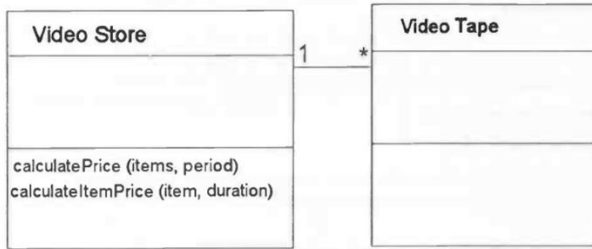
8.2. Contoh Refactoring

Skenario:

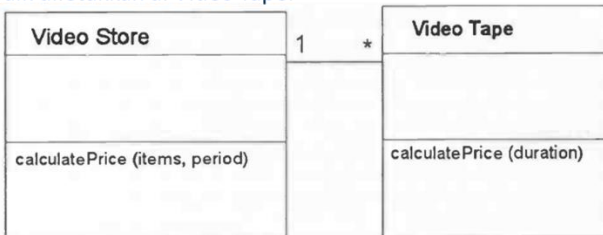
Terdapat sebuah rental video. Penyewaan video berdasarkan tarif perjam. Penghitungan jumlah tarif agar lebih mudah dilakukan setiap hari. Rental Video juga akan menambahkan CD music untuk disewakan



CalculatePrice dapat disederhanakan dengan pemanggilan ulang menggunakan method baru CalculateItemPrice



CalculateItemPrice tidak menggunakan data dari Video Store, tapi menggunakan data dari Video Tape. Sehingga fungsi CalculateItemPrice lebih baik diletakkan di Video Tape.



CalculateItemPrice menjadi CalculatePrice dan sudah tidak menggunakan parameter item

Gambar 8.3. Contoh Refactoring

Pertemuan 9

Pemodelan Database

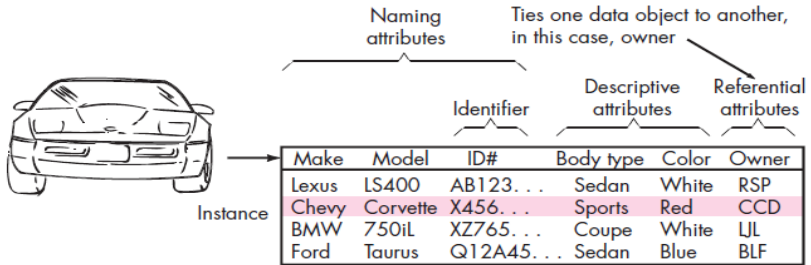
9.1. Pemodelan Data

Memodelkan data adalah menguji objek data secara independen dari pemrosesan. Dengan demikian berarti perlu memusatkan perhatian pada domain data dan membuat model di tingkat abstraksi pelanggan menunjukkan bagaimana objek data berhubungan satu sama lain.

Untuk lebih memahami maka perlu diketahui beberapa hal berikut:

1. Obyek Data, adalah representasi informasi gabungan yang harus dipahami oleh perangkat lunak. Informasi gabungan dapat berupa sesuatu yang memiliki properti atau atribut berbeda.
2. Informasi gabungan juga dapat berupa entitas eksternal (misalnya, apapun yang menghasilkan atau mengkonsumsi informasi), sesuatu (misalnya, laporan atau tampilan), kejadian (misalnya, panggilan telepon) atau peristiwa (misalnya, alarm), peran (misalnya, tenaga penjual), unit organisasi (misalnya, departemen akuntansi), tempat (misalnya, gudang), atau struktur (misalnya, file)
3. Deskripsi objek data menggabungkan objek data dan semua atributnya.
4. Objek data hanya mengenkapsulasi data — tidak ada referensi dalam objek data ke operasi yang bekerja pada data.
5. Oleh karena itu, objek data dapat direpresentasikan sebagai tabel berikut.

9.2. Tabel Representasi Obyek Data



Gambar 9.1. Tabel Spesifikasi Mobil

Obyek Data berisi sekumpulan atribut yang bertindak sebagai aspek, kualitas, karakteristik, atau deskriptor obyek.

Obyek :

- Mobil

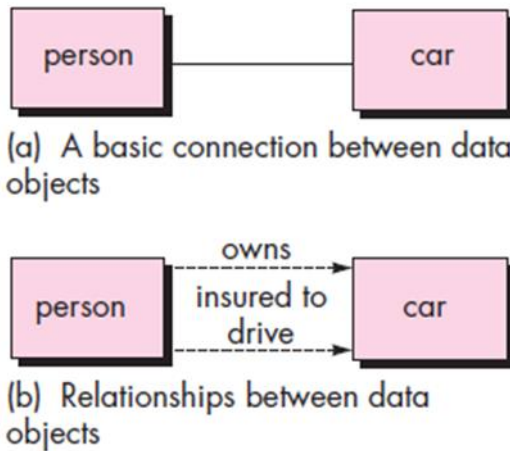
Atribut:

- Make,
- Model
- Body Type, Price
- Option Code

9.3. Relationship

Obyek data terhubung satu sama lain dengan cara yang berbeda. Terdapat relasi antara orang dan mobil karena kedua objek tersebut saling berhubungan. Orang memiliki mobil. Orang diasuransikan untuk mengendarai mobil. Hubungan 'memiliki' dan 'diasuransikan untuk mengendarai' menentukan hubungan yang relevan antara orang dan mobil.

Obyek dapat berelasi dengan beberapa cara



Gambar 9.2. Model Relasi

9.4. Entity Relationship Diagram (ERD)

Level 1 - memodelkan semua objek data (entitas) dan "koneksi" mereka satu sama lain.

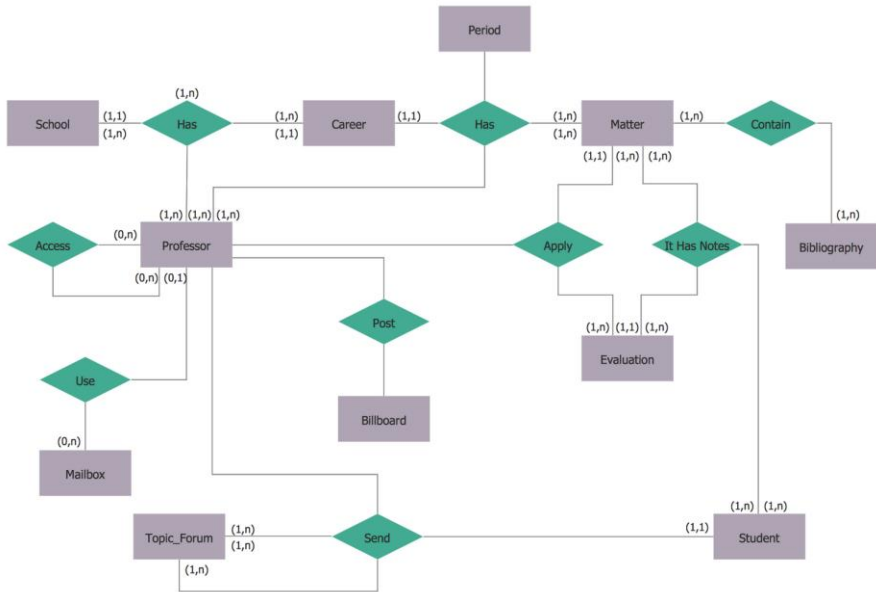
Level 2 - memodelkan semua entitas dan hubungannya.

Level 3 - memodelkan semua entitas, hubungan, dan atribut yang memberikan kedalaman.

Gambar 9.3. menggambarkan bagaimana setiap entitas (persegi) saling berelasi dengan entitas lain melalui relasi (belahketupat). Notasi di sebelah entitas dan relasi adalah cardinalitas:

- (0,n) menunjukkan hubungan antar entitas. Entitas pertama boleh tidak berhubungan atau berhubungan dengan banyak entitas kedua
- (0,1) menunjukkan hubungan antar entitas. Entitas pertama boleh tidak berhubungan atau berhubungan dengan hanya satu entitas kedua

- (1,1) menunjukkan hubungan antar entitas. Satu Entitas pertama berhubungan dengan hanya satu entitas kedua.
- (1,n) menunjukkan hubungan antar entitas. Satu Entitas pertama boleh berhubungan dengan banyak entitas kedua



Gambar 9.3. Contoh ERD

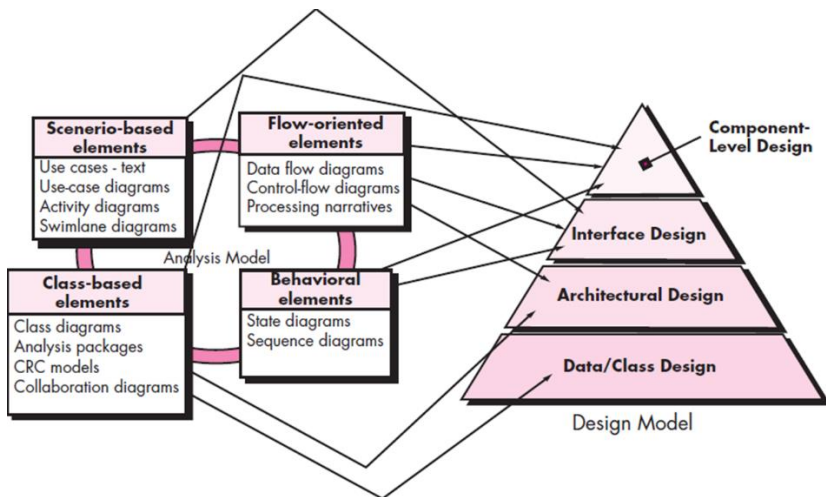
Pertemuan 10

Konsep Desain

Tujuan pengembangan sebuah perangkat lunak adalah menghasilkan produk yang berkualitas baik. Produk perangkat lunak dikatakan berkualitas baik jika memiliki 3 syarat berikut:

1. Firmness: Suatu program seharusnya tidak memiliki bug yang menghambat fungsinya.
2. Commodity: Suatu program harus sesuai dengan tujuan yang dimaksudkan.
3. Delight: Pengalaman menggunakan program ini haruslah menyenangkan

Model kebutuhan perlu diubah menjadi model desain. Tujuan transformasi ini adalah memberi panduan bagi tahapan coding (Gambar 10.1).



Gambar 10.1. Mengubah model kebutuhan menjadi model desain

Alasan mengapa desain harus berkualitas:

1. Desain harus menerapkan semua kebutuhan yang diinginkan oleh customer.
2. Desain harus menjadi panduan yang dapat dibaca dan dimengerti bagi mereka yang menghasilkan kode dan bagi mereka yang menguji dan kemudian mendukung perangkat lunak.
3. Desain harus memberikan gambaran lengkap tentang perangkat lunak, menangani data, fungsional, dan domain perilaku dari perspektif implementasi

Untuk menghasilkan produk yang berkualitas, berikut panduannya:

1. Sebuah desain harus menunjukkan arsitektur yang (1) telah dibuat menggunakan pola arsitektur yang dapat dikenali, (2) terdiri dari komponen yang menunjukkan karakteristik desain yang baik, dan (3) dapat diimplementasikan secara evolusioner, sehingga memfasilitasi implementasi dan pengujian.
2. Sebuah desain harus modular; artinya, perangkat lunak harus secara logis dipartisi menjadi elemen atau subsistem.
3. Sebuah desain harus mengandung representasi data, arsitektur, antarmuka, dan komponen yang berbeda
4. Sebuah desain harus mengarah pada struktur data yang sesuai untuk kelas yang akan diterapkan dan diambil dari pola data yang dapat dikenali.
5. Sebuah desain harus mengarah pada komponen yang menunjukkan karakteristik fungsional independen.
6. Sebuah desain harus mengarah pada antarmuka yang mengurangi kompleksitas koneksi antar komponen dan dengan lingkungan luar.
7. Sebuah desain harus diturunkan menggunakan metode berulang yang didorong oleh informasi yang diperoleh selama analisis kebutuhan perangkat lunak.

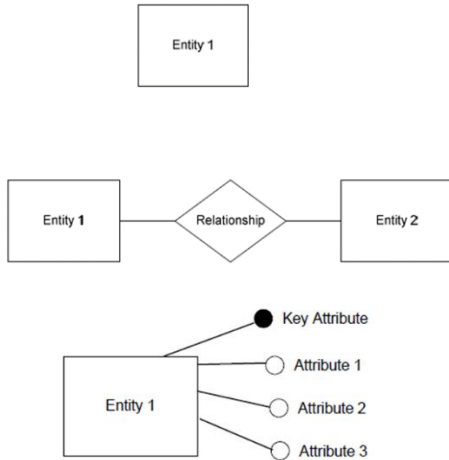
8. Sebuah desain harus direpresentasikan menggunakan notasi yang secara efektif mengkomunikasikan maknanya.

Pertemuan 11

Entity Relationship Diagram (ERD)

(lanjut)

Terdapat 3 elemen ERD: Entitas, Relationship, dan attribute.



Gambar 10.1. Contoh ERD.

Atribut data adalah properti yang umum dimiliki sebuah entitas. Untuk membangun ERD diperlukan Langkah-langkah berikut:

1. Identifikasi Entiti
2. Tentukan Relasi
3. Gambar E-RD Awal
4. Isi Kardinalitas
5. Tentukan Primary Key
6. Gambar Key-Based ERD
7. Identifikasi Atribut
8. Petakan Atribut
9. Gambar atribut ERD selengkapnya
10. Check Hasil

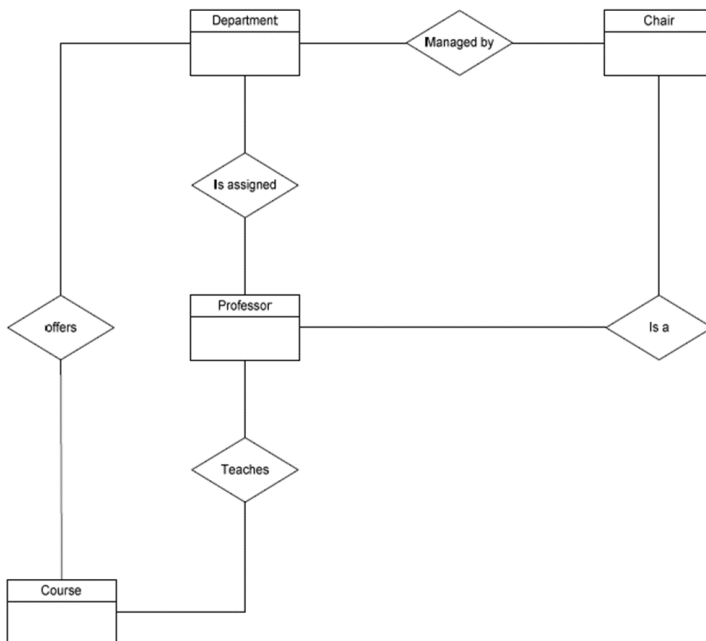
1. Identifikasi Entiti

- Temukan semua kata benda dan frase kata benda dalam deskripsi
- Pertimbangkan kandidat entitas ini
- Calon entitas adalah departemen, ketua, profesor, mata kuliah, dan bagian mata kuliah.

2. Tentukan Relasi

course	offered by		faculty by	
professor	assigned to			research
chair	manages		is a	
department		manages by	is assigned	offers
	department	chair	professor	course

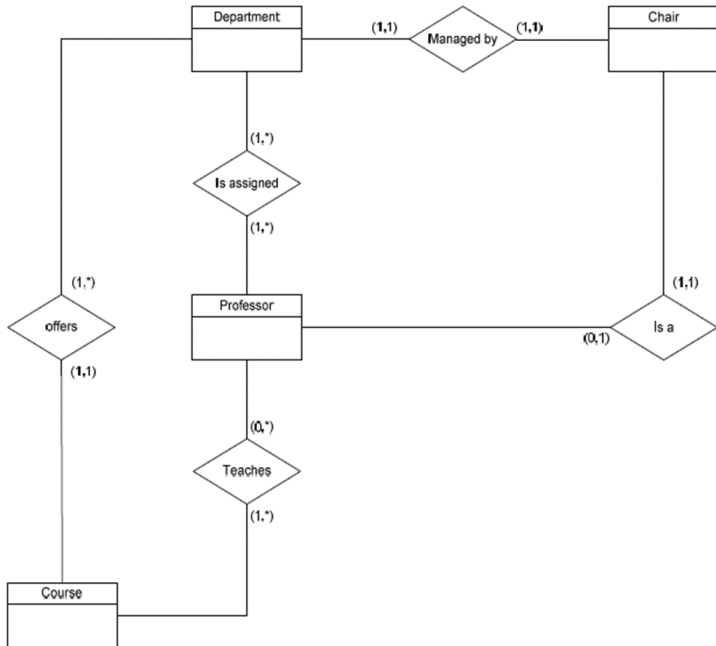
3. Gambar E-RD Awal



Gambar 11.1. ERD awal

4. Isi Kardinalitas

Perlu menghilangkan hubungan banyak-ke-banyak, dan menutup hubungan satu-ke-satu yang masuk akal. Misalnya, Ketua, tanpa perilaku apa pun, sebenarnya hanyalah atribut sebuah departemen. Jadi kita dapat menghapusnya sebagai entitas dan kemudian menambahkannya sebagai atribut.



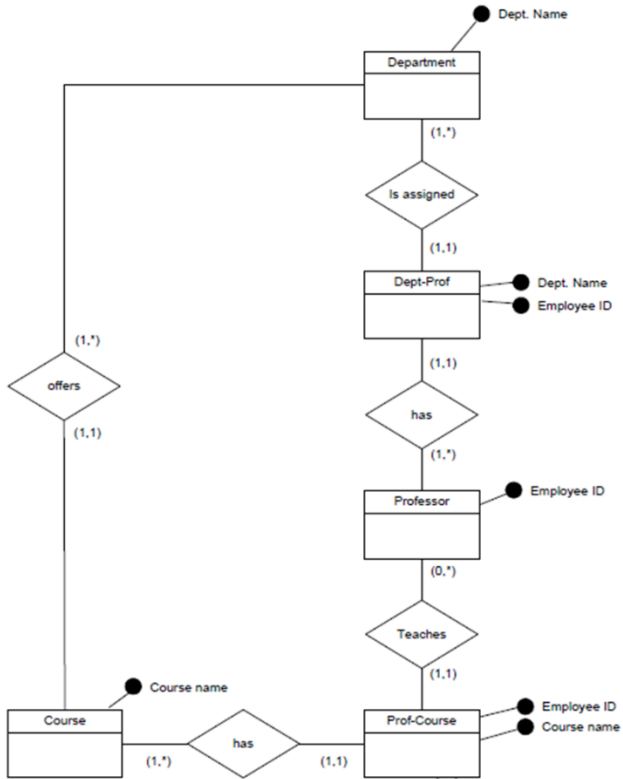
Gambar 11.2. ERD dengan kardinalitas

5. Tentukan Primary Key

<u>department</u>	name
dept-prof	dept. name employee id
<u>professor</u>	employee id
prof-course	employee id course name
course	course name

Gambar 11.3. Primary Key

6. Gambar Key-Based ERD



Gambar 11.4. Key-Based ERD

7. Identifikasi Atribut

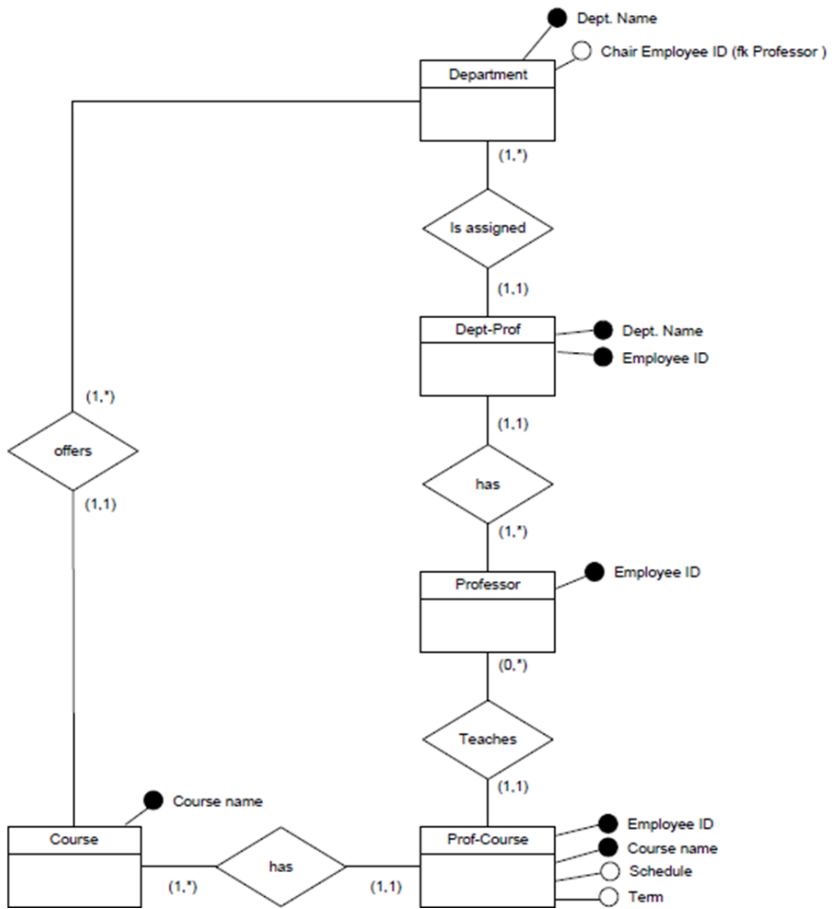
Identifikasi semua karakteristik entitas yang relevan dengan domain yang dianalisis. Mengecualikan kunci yang sudah diidentifikasi: Jadwal, Masa Berlaku, Nama Profesor, Ketua Departemen (yang merupakan ID karyawan, kunci asing untuk Profesor).

8. Petakan Atribut

Tentukan entitas mana yang dimiliki setiap karakteristik. Jangan menduplikasi atribut di seluruh entitas. Jika perlu, pegang mereka dalam entitas baru yang terkait.

Jadwal -> Prof-Course, Term -> Prof-Course, Ketua -> Departemen

9. Gambar atribut ERD selengkapnya



Gambar 11.5. ERD lengkap

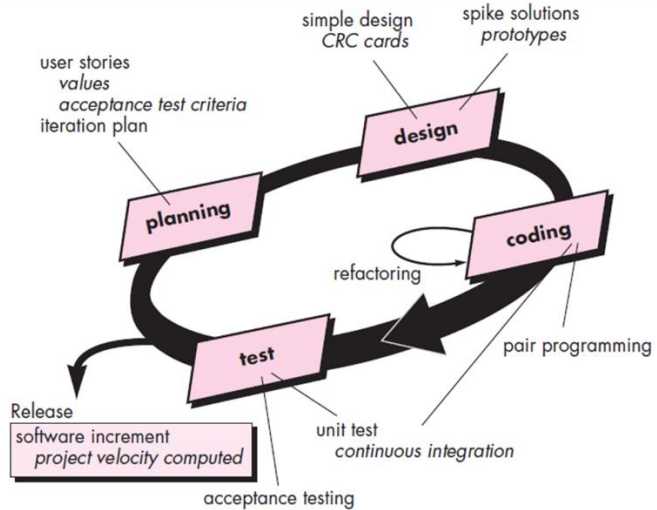
TUGAS Kelompok:

1. Buat E-RD sesuai dengan topik kelompok masing-masing
2. Masukkan hasil desain data dari E-RD ke dalam aplikasi MS. Access.

Pertemuan 12

Planning, Design, Coding

Pengembangan perangkat lunak memiliki tahapan dengan metode Extreme Programming (XP) seperti terlihat pada gambar 12.1.

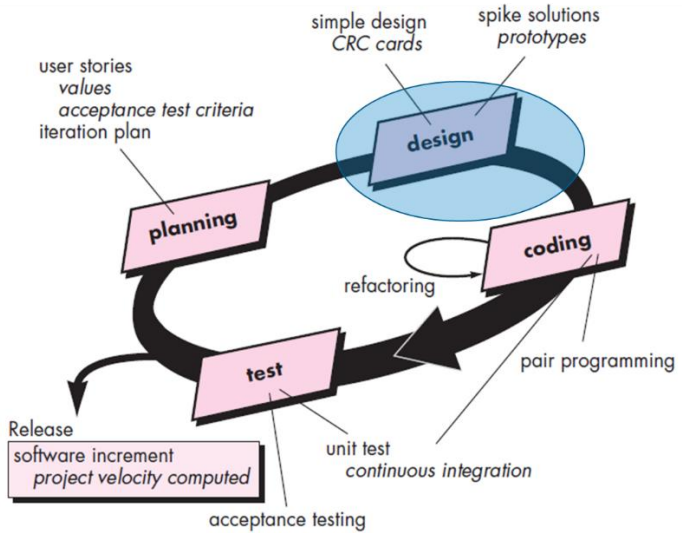


Gambar 12.1. Tahapan

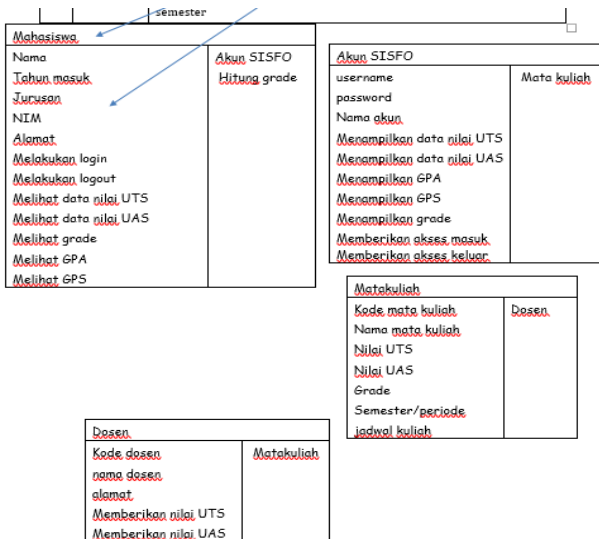
Sebagai seorang Mahasiswa

Saya ingin mengetahui hasil UTS dan UAS saya setiap semester

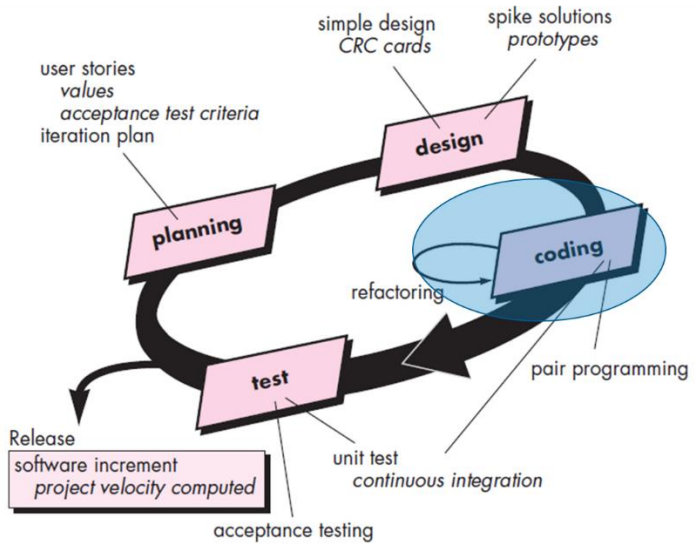
Sehingga saya bisa membandingkan perkembangan diri saya di setiap semester



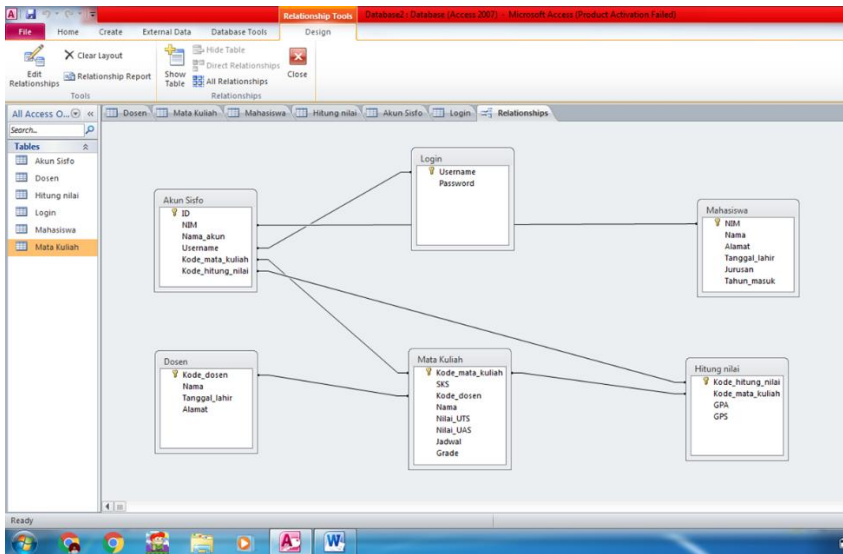
Gambar 12.2. Tahapan Desain



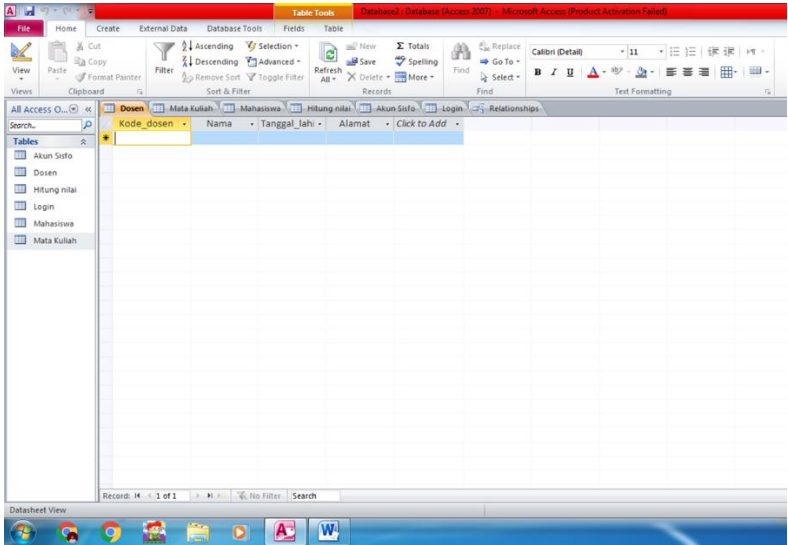
Gambar 12.3. Hasil thapan desain



Gambar 12.4. Tahapan Coding



Gambar 12.5. Hasil coding pada database



Pertemuan 13

Desain Webapp

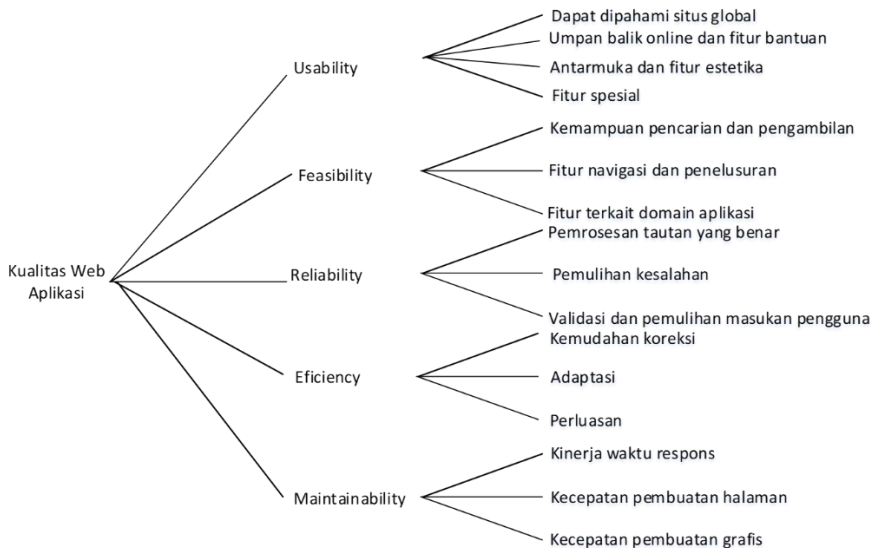
Terdapat dua pendekatan mendasar mendesain webapp:

1. cita-cita artistik untuk mengekspresikan diri Anda dan
2. cita-cita teknik untuk memecahkan masalah bagi pelanggan

Desain webapp dilakukan pada saat:

- Ketika konten dan fungsinya kompleks;
- Ketika ukuran WebApp mencakup ratusan atau ribuan objek konten, fungsi, dan class analisis;
- Ketika kesuksesan WebApp akan berdampak langsung pada kesuksesan bisnis,

Gambar 13.1 menunjukkan pohon kebutuhan desain webapp.



Gambar 13.1. Pohon kualitas kebutuhan

Pertemuan 14

Software Testing Strategies

Pengujian perangkat lunak adalah proses menjalankan program dengan maksud khusus untuk menemukan kesalahan sebelum dikirimkan ke pengguna akhir. Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merupakan review puncak terhadap spesifikasi, desain dan pembuatan program. Pengujian perangkat lunak menghabiskan upaya 30-40% dari total pekerjaan proyek. Untuk proyek yang membahayakan nyawa manusia, biaya pengujian bisa 3-5 X proyek biasa

Tujuan pengujian adalah:

1. Menjalankan program untuk menemukan error.
2. Test case yang bagus adalah yang memiliki kemungkinan terbesar untuk menemukan error yang tersembunyi.
3. Pengujian yang sukses adalah yang berhasil menemukan error yang tersembunyi.

Prinsip Pengujian adalah:

- Harus bisa dilacak hingga sampai ke kebutuhan customer.
- Harus direncanakan sejak model dibuat.
- Dari lingkup kecil menuju yang besar.
- Tidak bisa semua kemungkinan diuji.
- Dilakukan oleh pihak ketiga yang independen.

Testability adalah kemudahan untuk diuji mempunyai karakteristik:

- Operability: mudah digunakan.
- Observability: mudah diamati.
- Controlability: mudah dikendalikan.
- Decomposability: mudah diuraikan.

- Simplicity: lingkup kecil, semakin mudah diuji.
- Stability: jarang berubah.
- Understandability: mudah dipahami.

Desain Kasus Pengujian:

Black box testing

- Memastikan fungsional P/L berjalan.
- Kesesuaian input dengan output.
- Tidak memperhatikan proses logic internal.

White box testing

- Pengamatan detail prosedur.
- Mengamati sampai level percabangan kondisi dan perulangan.

Reference:

1. Ambler S., '**CRC Modeling Bridging the Communication Gap Between Developers and Users**', An AmbySoft Inc. White Paper, 1998
2. Cahyono A. L., Nugroho E., "BELAJAR DARI KEGAGALAN PROYEK-PROYEK TEKNOLOGI INFORMASI", SemNasIF, 2014.
3. Cohn M., 'User Story Applied: For Agile Software Development', Addison Wesley, 2004.
4. Pressman R., 'Software Engineering, 7th edition, 2008