

BAB 1

REKAYASA SOFTWARE

1.1 Definisi Software

Istilah *software* dalam Bahasa Indonesia dikenal dengan istilah perangkat lunak. Namun demikian kecenderungan untuk tetap menyebut *software* dalam berbagai literatur terkait teknologi informasi lebih banyak dan terkesan lebih alami. Oleh karena itu, istilah yang digunakan dalam buku ini adalah *software*. *Software* sering ditulis juga dengan singkatan SW atau S/W.

Secara historis istilah *software* pertama kali muncul pada tahun 1957 dan dinyatakan oleh Alan Turing. Sejak saat itu istilah *software* sering dipakai dalam dunia teknologi komputer. Secara bahasa menurut www.dictionary.com *software* dijelaskan sebagai berikut:

The programs used to direct the operation of a computer, as well as documentation giving instructions on how to use them.

Sedangkan menurut www.businessdictionary.com, istilah *software* didefinisikan sebagai berikut:

Organized information in the form of operating systems, utilities, programs, dan applications that enable computers to work.

Roger S. Pressman memberikan definisi tentang *software* sebagai berikut:

Software is (1) instructions (computer programs) that when executed provide desired function and performance, (2) data structures that enable the programs to adequately manipulate information, and (3) documents that describe the operation and use of the programs

Dengan demikian, *software* memiliki arti yang sangat luas, sehingga bisa dikatakan segala sesuatu yang menyangkut pemanfaatan komputer melalui program komputer, apakah itu mengembangkan program komputer maupun menggunakan program komputer disesuaikan dengan kebutuhannya masing-masing.

Pada kenyataannya *software* selalu terikat dengan *hardware* yang terdiri dari berbagai macam komponen. Oleh karena itu, komputer hanya bisa berfungsi apabila *software* dan *hardware* keduanya berjalan sebagaimana mestinya. *Software* juga tidak selalu harus menghasilkan tampilan visual, tetapi bisa berupa gerakan motorik seperti *software* untuk memfungsikan printer.

1.2 Definisi *Software Engineering*

Ian Sommerille mendefinisikan istilah *Software Engineering* (Rekayasa *Software* atau Rekayasa Perangkat Lunak) sebagai berikut:

Software engineering is an engineering discipline which is concerned with all aspect of software production from the early sages of system specification through to maintaining the system after it has gone into use.

Dengan demikian, segala keilmuan yang berkaitan dengan seluruh aspek produk *software* mulai dari tahapan awal sampai dengan tahapan pemeliharaan *software* paska pembuatannya. Rekayasa *software* tidak hanya pada permasalahan perencanaan dan perancangan saja, tetapi merupakan proses yang terintegrasi dan menyeluruh dari berbagai aspek, mulai dari sebelum suatu *software* dibuat sampai dengan selesai, bahkan sampai pada tahap penggunaan *software*.

Adapun, institusi IEEE mendefinisikan Rekayasa *Software* sebagai berikut:

Software engineering: (1) The application of systematic, disciplined, quantifiable, approach to development, operation and maintenance software. (2) The study of approaches as in (1)

Artinya bahwa Rekayasa *Software* memiliki cakupan yang luas, disamping membahas tentang sistematis, langkah-langkah pengembangan, perhitungan, juga membahas mengenai pendekatan yang seharusnya dilakukan, pemakaian *software*, serta membahas pula tentang proses pemeliharaan *software*.

Sedangkan, menurut Fritz Bauer mendefinisikan Rekayasa *Software* sebagai berikut:

Software engineering is establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Dengan demikian bisa dikatakan bahwa Rekayasa *Software* mencakup segala sesuatu hal dalam proses pengembangan *software* mulai dari analisis proses, analisis kebutuhan sistem, perancangan sistem, sampai implementasi sehingga membentuk suatu siklus hidup *software*. Rekayasa *Software* juga ditujukan untuk menghasilkan produk *software* yang berkualitas dan berfungsi secara efisien.

1.3 Tipe *Software*

Software yang dikenal saat ini dapat dikelompokkan secara umum menjadi beberapa jenis.

1. Bahasa Pemrograman

Software yang sangat mendasar dan berfungsi untuk membuat berbagai jenis program aplikasi seperti Java, Visual Basic, Fortran, dll

2. Sistem Operasi

Penggunaan komputer yang paling dasar dan menjadi platform untuk berjalannya berbagai aplikasi komputer seperti Linux, Windows, Android, dll

3. Aplikasi Perkantoran

Membantu pekerjaan perkantoran secara umum seperti Microsoft Office, Open Office, dll

4. Sistem Informasi

Pengolahan data menjadi informasi yang ditujukan untuk menunjang berbagai kepentingan bisnis seperti ERP, Moodle, dll

5. Basisdata

Penyimpanan dan pemrosesan data dalam ukuran yang sangat besar, seiring dengan perkembangan teknologi *hardware* yang mendukungnya. *Software* jenis ini biasanya disebut dengan *Database Management System* (DBMS) dan sangat terkait dalam pembuatan aplikasi sistem informasi, seperti Oracle, PostgreSQL, MySQL, dll.

6. *Game*

Program aplikasi untuk tujuan *entertainment*, dari mulai yang sederhana seperti Tetris, sampai dengan yang rumit berupa permainan strategi seperti Final Fantasy, dll.

1.4 Tahapan Pengembangan *Software*

Gambar 1.1 menunjukkan tahapan-tahapan secara umum yang dilakukan dalam *Software Engineering* yaitu sebagai berikut:

1. *Domain Engineering*

Pemahaman permasalahan yang muncul dan akan dijadikan objek pembuatan *software*. Menentukan kesepakatan bersama antara pengembang dan pengguna tentang ruang lingkup kebutuhan *software* yang harus dikembangkan.

2. *Requirement Engineering*

Pemahaman kebutuhan pengguna terhadap *software* yang akan dibangun, dan menetapkan solusinya. Seluruh kebutuhan pengguna idealnya dapat dipenuhi dan dituangkan dalam bentuk spesifikasi kebutuhan *software*.

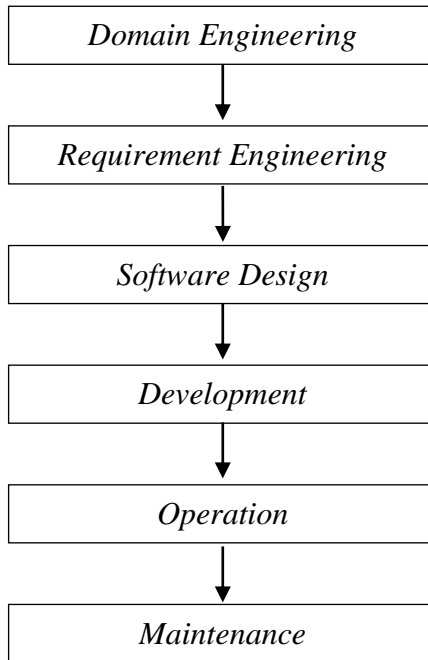
3. *Software Design*

Perancangan *blueprint* dari *software* yang akan dibuat baik berupa *database*, tampilan, modul-modul program yang harus dibuat, dengan menggunakan berbagai diagram supaya memudahkan pihak-pihak terkait dalam memahami dengan benar *software* yang harus diimplementasikan.

4. *Development*

Pengembangan *software* yaitu berupa program aplikasi komputer yang pembuatannya dilakukan sesuai dengan rancangan yang telah ditetapkan pada tahapan sebelumnya. Pembuatan aplikasi menggunakan salah satu bahasa pemrograman yang disepakati seperti Java, C, dll. Dalam hal ini termasuk juga pembuatan *database* untuk penyimpanan data yang diperlukan oleh aplikasi, dengan menggunakan *software database management system* atau DBMS.

Software yang diperlukan untuk pengembangan secara umum terbagi menjadi dua jenis yaitu *proprietary* dan *open source*. *Proprietary* memerlukan biaya untuk memakainya, sedangkan *open source* bisa digunakan secara cuma-cuma. Namun demikian, masing-masing memiliki kelebihan dan kekurangan, sehingga bisa digunakan kedua-duanya secara proposional.



Gambar 1.1 Tahapan Umum Rekayasa *Software*

5. *Operation*

Implementasi *software* yang telah dikembangkan, dalam rangka ujicoba *software* yang dilakukan bersama-sama antara pengembang dan pengguna. Kegiatan ini bisa juga termasuk pelatihan kepada pengguna untuk menggunakan *software*, dan perbaikan *software* paska produksi apabila ditemukan *error* atau hal lain yang belum sesuai dengan apa yang diinginkan.

6. *Maintenance*

Proses pemeliharaan *software* untuk menjamin bahwa *software* berjalan secara normal, dan semua kebutuhan *software* dipastikan dapat dipenuhi dan dapat disepakati oleh pihak pengembang maupun pihak pengguna.

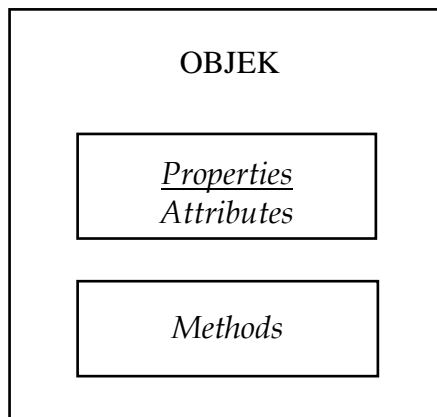
Dalam masa pemeliharaan biasanya dilakukan perbaikan-perbaikan apabila terjadi kesalahan yang belum terdeteksi pada waktu pengembangan. Termasuk dalam proses pemeliharaan yaitu perbaikan dengan tujuan untuk meningkatkan kinerja *software*. Bisa juga perbaikan terkait dengan keamanan *software* yang disebut dengan *security hole*.

BAB 2

ORIENTASI OBJEK

2.1 Pengenalan Objek

Ciri khas yang terpenting dalam pengembangan berbasis objek adalah bagaimana pengembangan sistem dirumuskan secara terpusat pada pembuatan objek-objek yang mendukungnya. Objek dapat dimisalkan sebagai suatu representasi benda nyata dimana memiliki beberapa atribut (*attribute*) dan beberapa metode(*method*), seperti pada gambar 2.1. Atribut menunjukkan data objek, sedangkan metode menunjukkan fungsi yang bisa dilakukan oleh objek.



Gambar 2.1. Konsep Objek

Dalam pengembangan aplikasi sistem informasi menggunakan pendekatan objek, program aplikasi akan dianalisis dan dirancang menjadi kumpulan objek yang satu

sama lain saling terkait, sehingga membentuk sistem aplikasi yang dibutuhkan.

Properties menunjukkan atribut dari suatu objek. Misalnya, untuk Objek Orang maka yang menunjukkan statusnya adalah diantaranya [nama], [umur], [kelamin], [pekerjaan]. *Property* disebut juga dengan *Attribute* atau *Field*.

Method menunjukkan pemrosesan yang bisa dilakukan terhadap suatu objek, atau untuk mendapatkan sesuatu respon dari suatu objek. Misalnya, pada Objek Orang [menanyakan nama], [menanyakan pekerjaan]. Pada umumnya digunakan untuk mengetahui status suatu, dan melakukan setting terhadap status suatu objek. *Method* disebut juga dengan istilah *Function*.

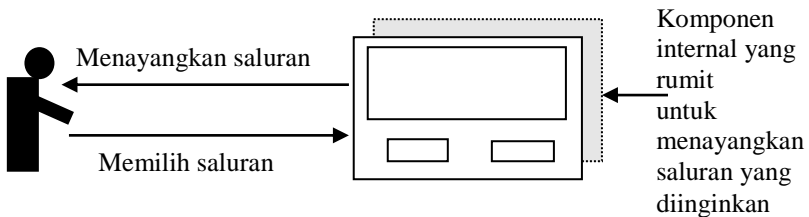
2.2 Karakteristik Objek

Ada tiga ciri utama yang berkaitan dengan pengembangan aplikasi berbasis objek sebagai berikut:

1. Pengkapsulan (*Capsulization*)

Pengkapsulan mengacu kepada mekanisme pengendalian akses terhadap *property* dan/atau *method* suatu objek oleh objek lain, artinya menyembunyikan *property* dan/atau *method* yang dimiliki suatu objek dari pengaksesan oleh objek lain, dengan hanya menyediakan beberapa saja yang dipandang perlu untuk bisa dipakai secara terbuka. Untuk mengatur penyembunyian seperti ini dalam *object oriented* diatur pelaksanaannya dengan menggunakan beberapa aturan visibility yaitu *private*, *protected* atau *public*.

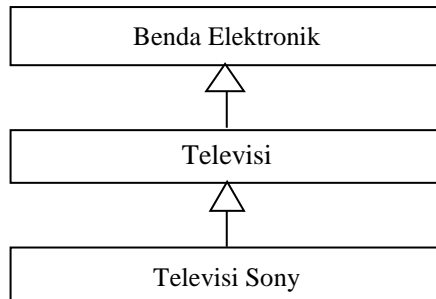
Sebagai contoh pada gambar 2.2, jam atau televisi adalah sebagai suatu objek. Biasanya tidak perlu mengetahui secara langsung bagaimana struktur internal dari benda-benda tersebut, dan tidak pernah akan ada fungsi tertentu untuk menunjukkan bagaimana struktur di dalamnya. Objek hanya memperlihatkan fungsi apa saja yang bisa digunakan oleh objek lain, dimana dalam contoh ini fungsi yang tersedia dari kedua objek tersebut diantaranya, adalah [memilih saluran yang diinginkan] dan [menampilkan saluran terpilih].



Gambar 2.2 Objek Televisi

2. Pewarisan (*Inheritance*)

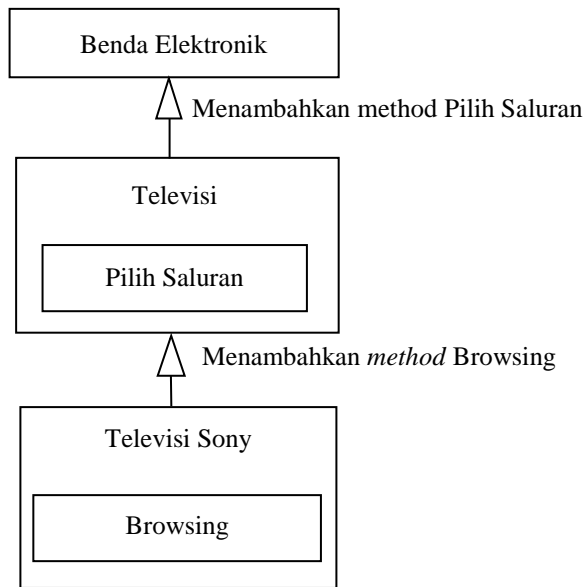
Ciri khas yang paling utama, yaitu teknik untuk menggunaulkan kode program yang sudah ada dengan penambahan status atau fungsi baru. Pewarisan untuk memperjelas struktur program dengan membantu peningkatan objek program yang dapat digunaulang.



Gambar 2.3 Pewarisan Objek

Seperti contoh pada Gambar 2.3, sebagai objek orangtuanya (*parent object*) adalah Benda Elektronik. Kemudian dibuat objek baru sebagai *child object* yaitu Televisi, dan bisa dibuat juga objek lain dari objek Televisi yaitu Televisi Sony. Dengan demikian, objek Televisi Sony dapat mewarisi objek Televisi. Demikian juga objek Televisi dapat mewarisi objek Benda Elektronik.

Pada objek-objek warisan bisa ditambahkan dengan attribute dan method yang baru, atau mendefinisi ulang attribute dan method yang ada pada objek orangtuanya, seperti pada Gambar 2.4. Sedangkan, Pada contoh di Gambar 2.5, objek Televisi di buat dari objek Benda Elektronik dengan menambahkan *method* Pilih Saluran untuk melihat acara televisi. Sedangkan pada objek Televisi Sony ditambahkan *method* baru yaitu *Browsing* untuk melihat tampilan web.



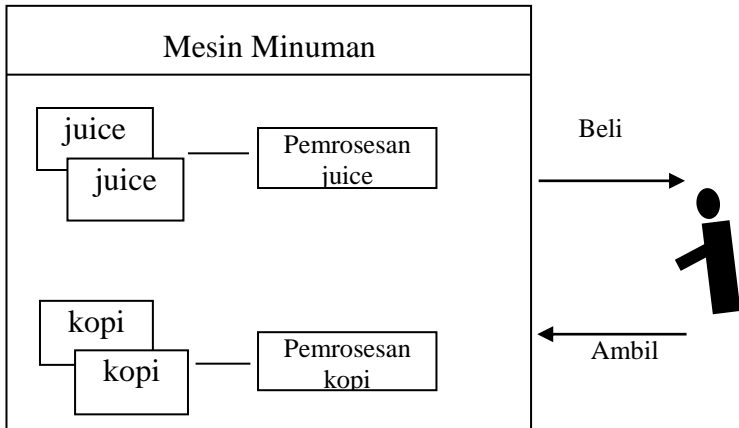
Gambar 2.4. Pembuatan Objek

3. Polimorfisme (*Polymorphism*)

Dalam pemrograman cara seperti ini adalah sangat ampuh dan ekspresif sehingga sering dapat menyingkat dan memudahkan pembacaan kode program. Ciri khas ini sedikit lebih sulit dibandingkan dengan dua ciri khas di atas yang telah dibahas. Untuk memudahkan pemahaman perhatikan pembicaraan berikut ini.

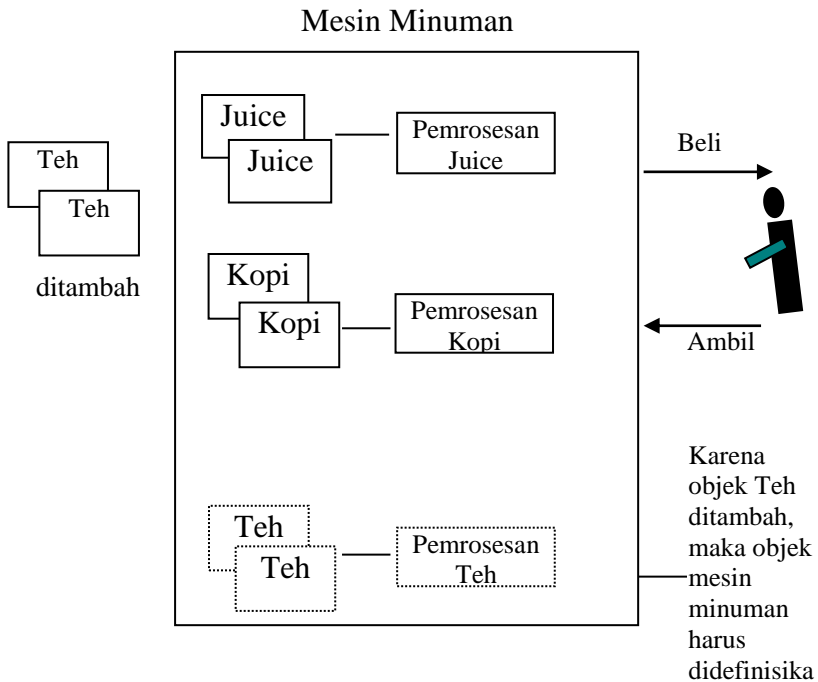
Ada alat penjual minuman otomatis (mesin minuman), dan minuman yang dijual pada mesin tersebut seperti juice, kopi. Masing-masing dianggap sebagai objek. Pada Objek mesin minuman di dalamnya terdapat objek lain yaitu objek minuman, dan menjualnya sesuai dengan keinginan pembeli. Objek mesin minuman ini akan memproses

penjualan minuman kalau objek minumannya memenuhi persyaratan tertentu (bentuk, ukuran, berat).



Gambar 2.5 Objek Mesin Minuman

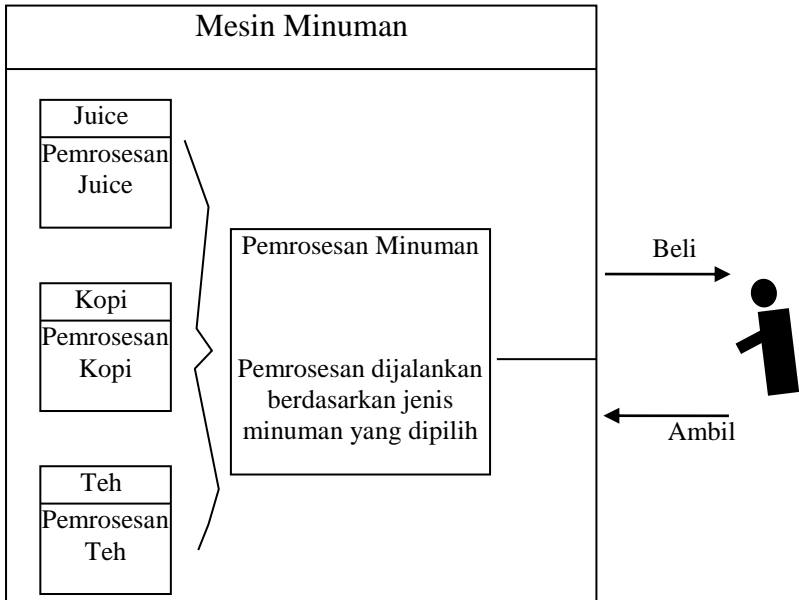
Ketika penyusunan program untuk sistem seperti ini, biasanya pada saat membuat objek mesin minuman objek minuman harus terlebih dahulu dibuat. Terutama sekali pada saat objek Minumannya lebih dari satu yang diproses secara berbeda. Setiap jumlah jenis minuman yang akan dijual bertambah, maka objek Mesin Minuman juga harus dibuat kembali.



Gambar 2.6 Jenis minuman bertambah, pendefinisian objek diulang

Untuk mengatasi permasalahan seperti inilah polimorfisme itu diperlukan. Pada contoh ini baik Juice maupun Kopi adalah sama-sama objek Minuman. Jadi ketika membuat objek Mesin Minuman tidak perlu masing-masing dibuat objek Juice dan objek Kopinya, tetapi cukup dengan membuat objek Minuman sebagai objek orangtua dari objek Juice dan Kopi. Dengan demikian, kalau terjadi penambahan jenis minuman pada waktu program dijalankan, tidak perlu lagi dilakukan perubahan program pada objek Mesin Minuman, artinya program menjadi lebih fleksibel, karena jenis minuman apapun selama merupakan

objek pewarisan dari objek minuman maka akan diberlakukan sama. Hal inilah yang memudahkan pembuatan program berbasis objek.



Gambar 2.7 Objek dengan Polimolisme

Pada Gambar 2.7, ada pemrosesan untuk minuman, dari sana [pemrosesan untuk Juice, Kopi] nya dibuat. Kalau mesin minuman tersebut menjalankan pemrosesan minuman, maka pemrosesan pada objek yang dimaksud akan dijalankan. Karena pemrosesan terhadap masing-masing objek minuman terpisah maka bersifat fleksibel.

Dengan demikian, pemrosesan untuk objek Kopi dan objek juice dapat didefinisikan di masing-masing, dari objek Mesin Minuman masing-masing pemrosesan bisa dijalankan dengan cara yang sama.

Pada contoh ini jelas bahwa selama objek Juice, Kopi, Teh itu dibuat melalui pewarisan dari objek minuman, maka berapapun penambahan jumlah jenis minumannya, hal ini tidak akan memerlukan perubahan program pada objek mesin minuman karena semuanya bisa diperlakukan secara seragam.

Polimorfisme seperti ini khususnya di dalam bahasa pemrograman Java dikenal dengan istilah *overload* dimana suatu fungsi walaupun namanya sama namun bisa dianggap berbeda (bisa menjalankan perintah yang berbeda) dengan memiliki jumlah atau jenis variable yang berbeda. Dengan demikian, *method* yang sama dengan cara pemanggilan yang berbeda, bisa melakukan pemrosesan yang berbeda pula.

BAB 3

PROSES REKAYASA *SOFTWARE*

3.1 *Capability Maturity Model*

Software Engineering Institute (SEI) telah mengembangkan model pengembangan yang komprehensif untuk memprediksi kemampuan sebuah organisasi pengembang *software* dalam mengikuti alur proses rekayasa *software* menjadi beberapa level. Tujuannya adalah agar bisa menentukan sejauh mana kondisi saat ini sebuah organisasi pengembang *software* sejauh mana mempunyai kemampuan dengan mengetahui tingkat maturitas pemrosesannya dalam pengembangan *software*.

SEI menggunakan hasil pengukurannya dalam lima tingkat atau level yang disebut dengan *Capability Maturity Model* (CMM). CMM menunjukkan kegiatan penting apa saja yang diperlukan dalam setiap level maturitasnya. SEI menggunakan level seperti ini supaya secara global bisa mengetahui sejauh mana proses rekayasa *software* telah nyata dipraktekkan dalam proses pengembangan *software* yang sebenarnya. Kelima level maturitas adalah sebagai berikut:

Level 1: Initial

Level yang paling rendah yang bersifat *ad hoc* dan bahkan terkadang bersifat *chaotic*. Beberapa proses telah terdefiniskan dan sangat tergantung pada usaha perorangan.

Level 2: Repeatable

Dasar-dasar proses dalam pengelolaan proyek telah dilakukan untuk penelusuran biaya, perencanaan, dan fungsionalitas. Tata tertib proses yang utama dan diperlukan dalam pelaksanaan proyek telah dijalankan sebagaimana mestinya, sehingga bisa mengulangi keberhasilan proyek untuk aplikasi yang sejenis..

Level 3: Defined

Pengelolaan dan kegiatan rekayasa yang dilakukan sebagai proses pembuatan *software* telah terdokumentasi, mengikuti standar tertentu, dan terintegrasi secara keseluruhan organisasi. Seluruh proyek yang dikerjakan telah terdokumentasi sesuai dengan alur proses pengembangan *software*. Level ini mencakup seluruh karakteristik pada level 2.

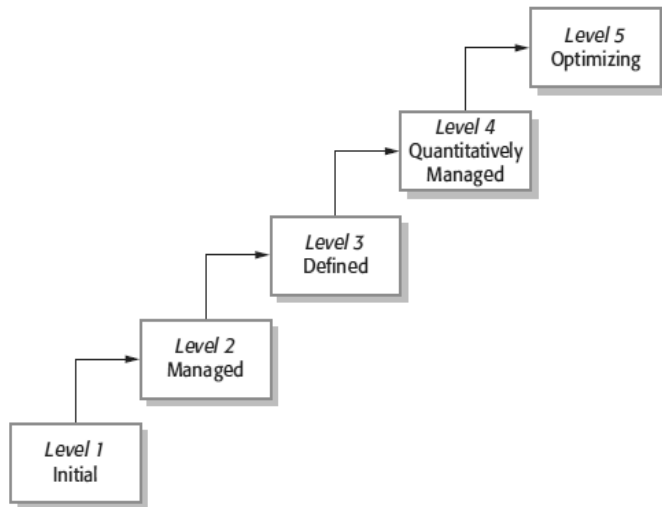
Level 4: Managed

Pengukuran secara rinci tentang proses pembuatan *software* dan jaminan mutu produk telah dilengkapi. Proses pembuatan *software* dan produk telah benar-benar dipahami dan dikendalikan dengan terukur. Level ini mencakup seluruh karakter yang ditetapkan untuk level 3.

Level 5: Optimizing

Perbaikan proses yang berkesinambungan diaktifkan oleh umpan balik kuantitatif dari proses dan dari pengujian ide-ide inovatif dan teknologi. Level ini mencakup semua karakteristik yang ditetapkan untuk level 4.

Kelima level yang didefinisikan oleh SEI diturunkan sebagai konsekuensi dari evaluasi respon terhadap kuesioner penilaian SEI yang didasarkan pada CMM. Hasil kuesioner yang disaring untuk penomoran tingkat yang menunjukkan tingkat kematangan proses suatu organisasi pengembangan *software*.



Gambar 3.1 Level Maturitas

Ke-delapan belas *Key Process Area* (KPA) ditentukan untuk keseluruhan level maturitas, dimana harus dilakukan untuk pencapaian maturitasnya, yaitu sebagai berikut:

Kematangan proses level 2

- *Software configuration management*
- *Software quality assurance*
- *Software subcontract management*
- *Software project tracking and oversight*

- *Software project planning*
- *Requirements management*

Kematangan proses level 3

- *Peer reviews*
- *Intergroup coordination*
- *Software product engineering*
- *Integrated software management*
- *Training program*
- *Organization process definition*
- *Organization process focus*

Kematangan proses level 4

- *Software quality management*
- *Quantitative process management*

Kematangan proses level 5

- *Process change management*
- *Technology change management*
- *Defect prevention*

Setiap KPA yang ditetapkan merupakan satu set *key practices* (praktek-praktek rekayasa *software*) yang dapat mendukung pencapaian tujuan.

3.2 Metodologi

Tahapan-tahapan atau proses apa saja yang dilakukan dalam pengembangan suatu program aplikasi atau *software* sistem informasi yang secara umum dipakai, yaitu meliputi:

1. *User Requirement*
2. *Analysis*
3. *Design*
4. *Coding*
5. *Testing*
6. *Implementation & Maintenance*

1. *User Requirement*

Tim pengembang sistem bekerjasama dengan counter part dari pihak user melakukan pengenalan sistem melalui wawancara, diskusi, *questionaree* dengan dilengkapi berbagai dokumen berupa formulir atau program yang sudah ada untuk dikaji, sehingga keinginan dari pihak user tentang bentuk sistem yang dibutuhkan seperti apa menjadi jelas batasannya.

2. *Analysis*

Hal-hal penting yang harus diperhatikan pada waktu analisis *user requirement*.

- a. Fokus kepada [what/apa yang harus dilakukan oleh sistem] dan bukan pada [How/bagaimana cara melakukannya]
- b. Mendengarkan secara seksama apa saja yang dikatakan oleh user, dan mencatatnya setiap point penting. Bila perlu gunakanlah alat rekam suara untuk pengecekan hasil pembicaraan dengan user agar diperoleh informasi yang detail yang diperlukan dalam penggambaran diagram usecase.

- c. Menentukan skala prioritas disesuaikan dengan keterbatasan anggaran dan resource yang dimiliki oleh *user*.
- d. Penulisan dokumentasi hasil analisis yang mudah dimengerti (mudah dicerna saat nanti dilihat kembali) dengan menggunakan UML terutama diagram *usecase*, diagram *class*, diagram *activity*, dan diagram *statechart*.

Pada tahapan ini harus sudah terdefiniskan gambaran sistem baik secara software maupun secara hardware yang akan dibangun yang disetujui oleh pihak user disesuaikan dengan budget yang tersedia.

3. *Design*

Untuk merancang hasil analisis menjadi sebuah rancangan *software* sistem informasi. Hasil rancangan merupakan notasi-notasi tertentu yang menunjukkan bagaimana sebuah software harus dibuat. Hasil dari perancangan adalah pedoman bagi programmer untuk diimplementasikan menjadi software sebenarnya yang akan dijalankan oleh penggunanya. Rancangan yang dibuat meliputi rancangan *hardware*, *database*, dan sistem *software*.

4. *Coding*

Membuat pemrograman berdasarkan perancangan yang

telah disepakati antara tim pengembang dengan pihak user. Hasil pada tahap rancangan yang berorientasi objek sebaiknya memang menggunakan secara maksimal pemrograman objek. Namun tidak salah juga menjabarkannya dalam bentuk pemrograman struktural, terutama pemrograman database yang belum sepenuhnya ada DBMS yang secara total mensupport orientasi objek.

5. *Testing*

Melakukan testing *software* berupa debugging dalam berbagai level dari mulai tahap modulasi, sistem, integrasi, dan ujicoba langsung oleh user. Untuk selanjutnya dilakukan perbaikan-perbaikan sampai *software* bisa berjalan sesuai dengan rancangan yang diinginkan.

6. *Implementation & Maintenance*

Instalasi *software* yang telah dibuat dan dipraktekkan di lapangan. Pada kenyataanya setelah sistem diimplementasikan selalu ada error yang di luar dugaan yang bersifat human *error*, sehingga masih perbaikan lebih lanjut. Hal ini pasti terjadi, dan tidak boleh bersifat esensial karena kalau hal-hal mendasar terjadi akan berakibat fatal dan berpengaruh terhadap keseluruhan sistem.

3.3 *Waterfall*

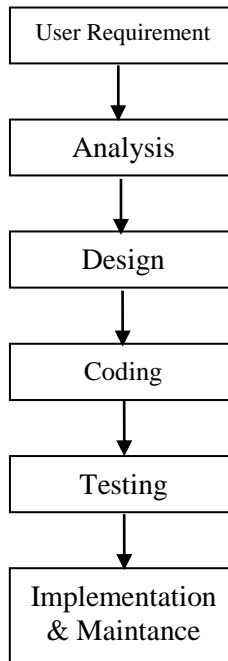
Metodologi ini adalah yang paling lama dan pertama kali diperkenalkan dalam rekayasa *software*. Seperti pada

Gambar 3.2, metodologi ini pada dasarnya bersifat linier artinya pengembangan *software* akan dilakukan secara berurut dari satu tahapan ke tahapan berikutnya, tanpa diperbolehkan kembali ke tahapan sebelumnya.

Ciri khas metodologi ini, bahwa perpindahan dari satu tahapan ke tahapan berikutnya dilakukan secara sistematis. Tahapan selanjutnya hanya bisa dilakukan apabila tahapan sebelumnya benar-benar telah sepenuhnya dilaksanakan.

Metodologi ini sangat cocok terutama untuk mengembangkan suatu *software* yang berskala besar. *Software* dikembangkan untuk mendukung proses bisnis suatu sistem yang tidak akan mengalami perubahan lagi.

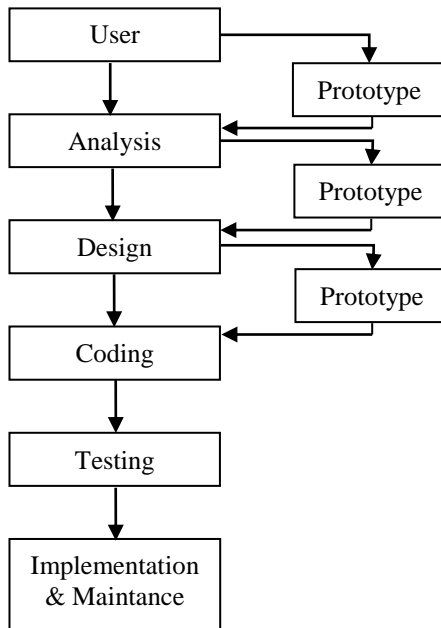
Metodologi ini menjadi pondasi bagi metodologi lainnya yang diperkenalkan kemudian, dengan melakukan modifikasi baik berupa penambahan proses atau pengulangan tahapan.



Gambar 3.2 *Waterfall*

3.4 *Prototype*

Metodologi ini secara tahapan secara umum mirip dengan *waterfall*. Seperti pada Gambar 3.3 yang membedakan adalah dari sejak tahapan awal diterapkan prototipe atau model *software* yang diperkirakan oleh *developer* berdasarkan survey awal dan yang akan dipakai kelak. Mulai tahapan awal model *software* diperlihatkan, tujuannya agar bisa menangkap kebutuhan *software* yang secara detail. Dengan demikian, diharapkan cara seperti ini dapat menghindari kesalahpahaman tentang *software* yang diinginkan oleh penggunanya.



Gambar 3.3 *Prototype*

Masalah yang sering terjadi pengembangan *software* dengan metodologi *waterfall* adalah terjadinya gap antara developer dan pengguna. Hal ini terjadi karena pengguna kurang dilibatkan dalam pengembangan, dan pengguna tidak memiliki bayangan *software* yang dihasilkan. Metodologi ini jika diterapkan, maka gap antara pengembang dan pengguna bisa diminimalisir, karena pengembangan didasarkan pada kerjasama dan komunikasi aktif antara pengembang dan pengguna.

Kelebihan dari metodologi ini adalah dapat mempercepat proses implemenasi *software* karena sejak awal dibuat contohnya, dan dikembangkan secara bertahap sesuai

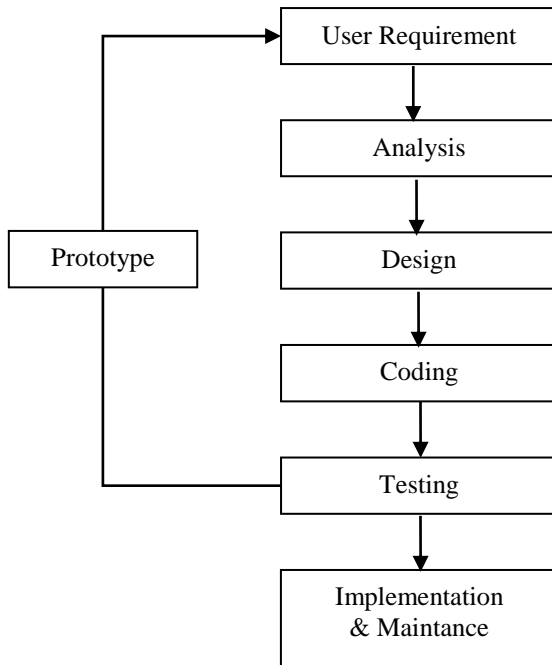
dengan keinginan pengguna. Selain itu, pengguna sudah mempunyai bayangan *software* yang diinginkannya.

3.5 Spiral

Metodologi ini tahapannya ditunjukkan seperti pada Gambar 3.4, dimana pengembangan *software* dilakukan secara berulang melalui sebuah prototipe. Dalam metodologi ini, penggunaan prototipe untuk mempermudah dalam mencari dan menetapkan *user requirement* agar lebih efisien dan lebih bertahap tanpa harus menunggu kesempurnaan setiap tahap pengembangan.

Salah satu kelebihan metodologi ini adalah pada tahapan analisis dilakukan *risk analysis* dalam setiap iterasi. *Risk analysis* dilakukan untuk memprediksi berbagai hal yang tidak diinginkan yang dapat menghambat pengembangan *software*.

Metodologi seperti ini bisa diterapkan terutama untuk sistem skala kecil dan menengah bisa diterapkan tanpa melibatkan terlalu banyaknya tim pengembang yang terlibat.



Gambar 3.4 Spiral

3.6 Poin Penting Pengembangan *Software*

Dalam proyek pengembangan *software* seorang manajer proyek harus memperhatikan secara seksama hal-hal berikut ini

1. Perkiraan waktu pengerjaan dan SDM yang diperlukan sebelum proyek dimulai.
2. Mengawasi kemajuan proyek setelah proyek

dimulai.

Kedua hal ini sangat penting untuk dipikirkan secara matang, karena merupakan faktor penting bagi keberhasilan suatu proyek, dimana ada beberapa keterbatasan, sedangkan tujuan proyek harus tetap dicapai sebagaimana yang diinginkan.

BAB 4

RATIONAL UNIFIED PROCESS

4.1 Pengembangan RUP

Rational Unified Process (RUP) merupakan metodologi yang berkaitan dengan pengembangan *software* berbasis pendekatan objek. Selain itu juga, seiring dengan kebutuhan pengembangan *software* yang lebih efisien dan lebih banyak melibatkan partisipasi pengguna, maka RUP menjadi salah satu alternatif metodologi yang banyak digunakan.

Pengembangan *software* berbasis RUP terdiri dari empat fase, dimana setiap fase memungkinkan dilaksanakan dalam beberapa iterasi.

1. *Inception*

Fase paling awal dimana menentukan nilai proyek dan sumber daya yang diperlukan untuk kelancaran proyek.

2. *Elaboration*

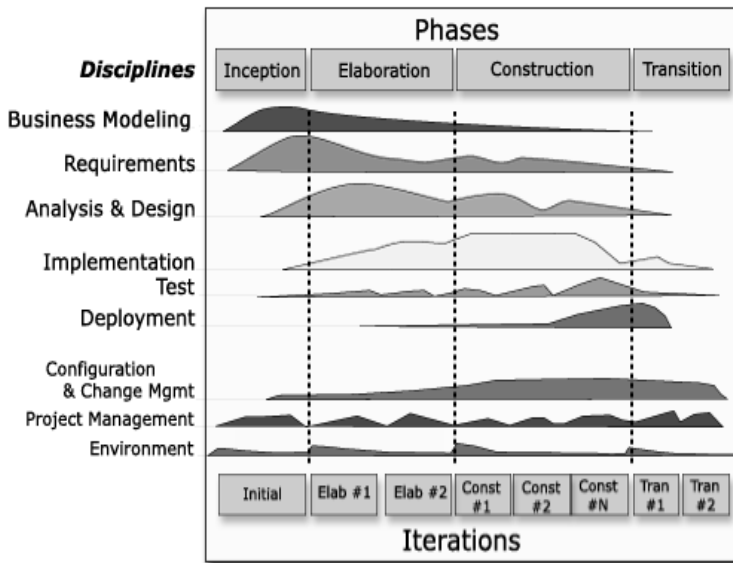
Menentukan arsitektur proyek dan sumber daya yang diperlukan. Pengembang mempertimbangkan berbagai kemungkinan dalam pembuatan *software* dan biaya yang diperlukan.

3. *Construction*

Fase pengembangan *software* sampai dihasilkan produk, disesuaikan dengan hasil rancangannya dan dilakukan ujicoba *software*.

4. *Transition*

Fase akhir melepas *software* ke pengguna. Dilakukan juga penyesuaian akhir atau perubahan secukupnya didasarkan pada *feedback* dari pengguna.



Gambar 4.1 RUP

Seperti tertera pada Gambar 4.1, metodologi RUP bisa dilihat dari dua dimensi

1. Dimensi horisontal yang menunjukkan waktu pengembangan dan aspek siklus hidup proses pengembangan
2. Dimensi vertikal yang menunjukkan disiplin ilmu atau jenis kegiatan yang harus dilakukan dalam pengembangan

Setiap fase di dalam RUP merupakan suatu siklus kecil yang terdiri dari proses utama (*business modeling, requirements, analysis & design, implementation, test, dan deployment*), dan proses pendukung (*configuration & change management, project management, dan environment*).

Setiap fase pengembangan software dalam RUP memiliki pembobotan kegiatan berbeda-beda, yaitu:

1. Fase *Inception* pada kegiatan *business modeling* dan *requirements*
2. Fase *Elaboration* pada kegiatan *requirements, analysis & design*
3. Fase *Construction* pada kegiatan *implementation, test, dan deployment*
4. Fase *Transition* pada kegiatan *deployment, dan kegiatan pendukung*

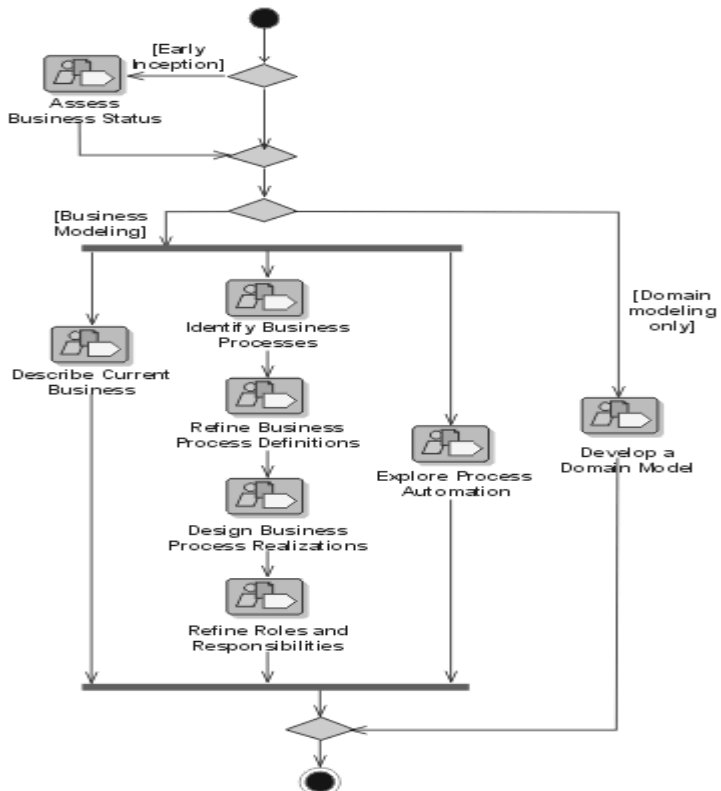
4.2 Business Modeling

Pomedalan bisnis tujuannya untuk mengetahui proses bisnis bagaimana yang sedang berjalan, dan proses bisnis apa saja yang bisa diefektifkan dan membutuhkan pemrosesan dengan didukung oleh aplikasi.

Pemahaman terhadap proses bisnis merupakan kunci keberhasilan *software* yang dihasilkan, karena pada dasarnya *software* yang berhasil adalah *software* yang benar-benar bisa dipakai dan bermanfaat bagi penggunaanya karena mendukung pekerjaan yang dilakukannya.

Tahapan umum untuk melakukan pemodelan bisnis seperti yang ditunjukkan pada Gambar 4.2. Untuk menggambarkan proses bisnis yang berjalan diperlukan survey, wawancara, dan

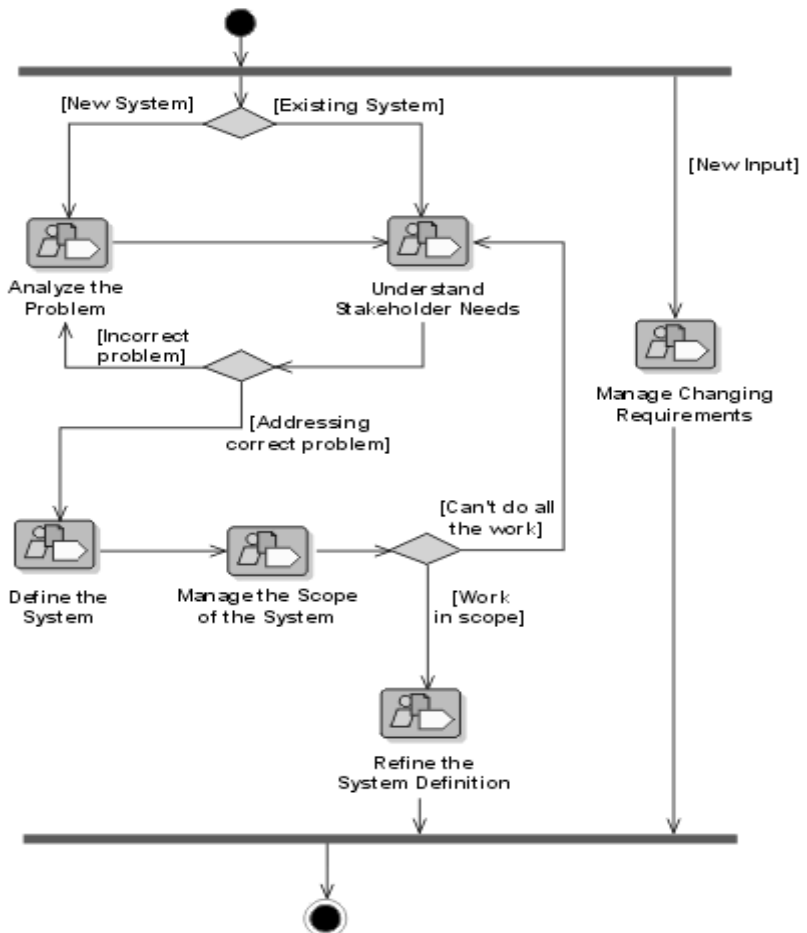
pengecekan dokumentasi yang terkait lainnya.



Gambar 4.2 *Business Modeling*

4.3 Requirements

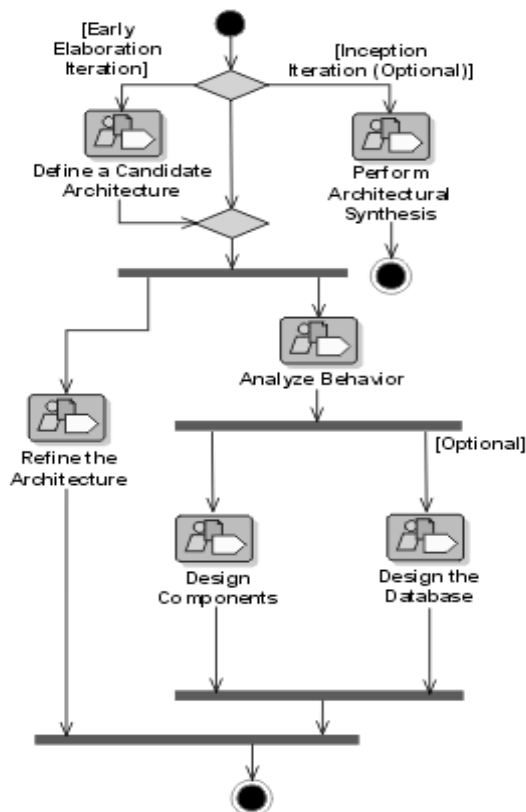
Gambar 4.3 menunjukkan kegiatan memahami berbagai kebutuhan tentang aplikasi yang akan dibangun baik yang bersifat fungsional maupun non-fungsional.



Gambar 4.3 *Requirements*

4.4 Analysis & Design

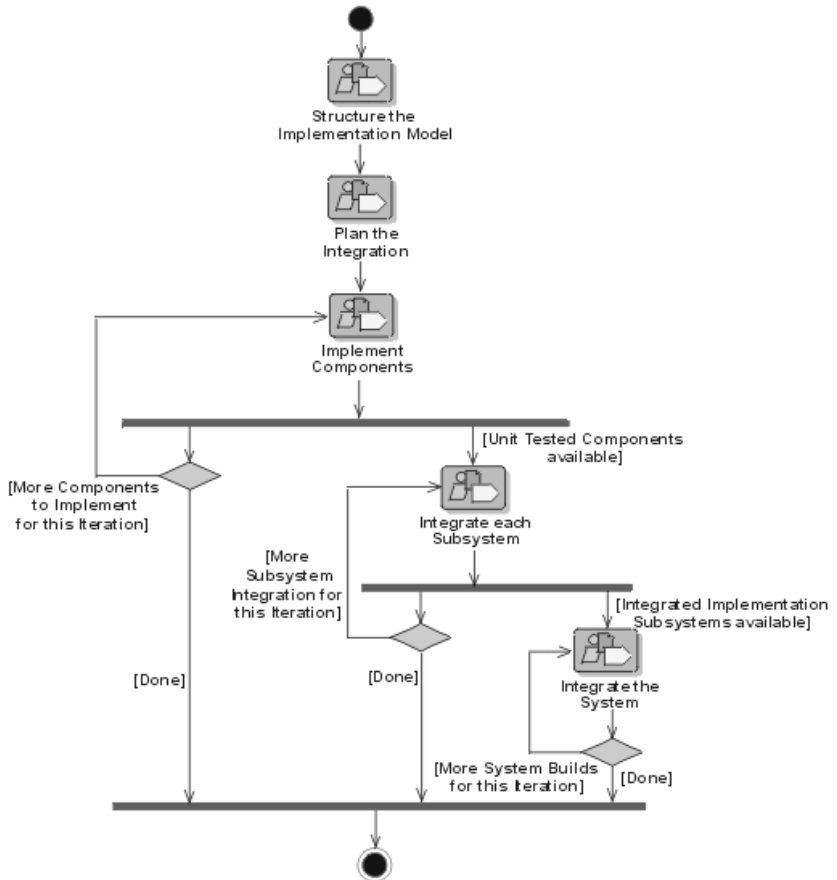
Gambar 4.4 menjelaskan rincian kegiatan analisis berbagai kebutuhan sistem, kemudian dibuatkan solusinya berupa berbagai rancangan yang diperlukan untuk diimplementasikan.



Gambar 4.4 Analysis & Design

4.5 Implementation

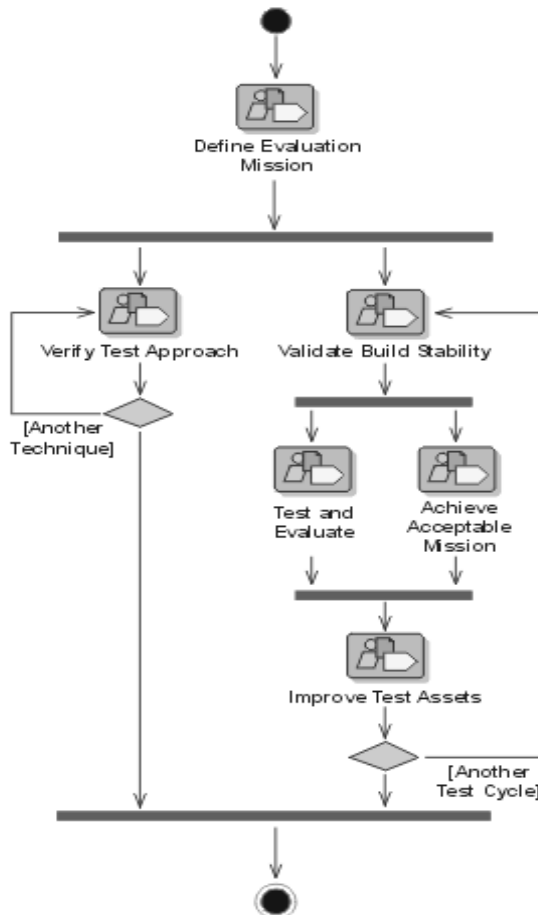
Gambar 4.5 menjelaskan kegiatan yang dilaksanakan untuk membuat software dengan menggunakan bahasa pemrograman.



Gambar 4.5 *Implementation*

4.6 Test

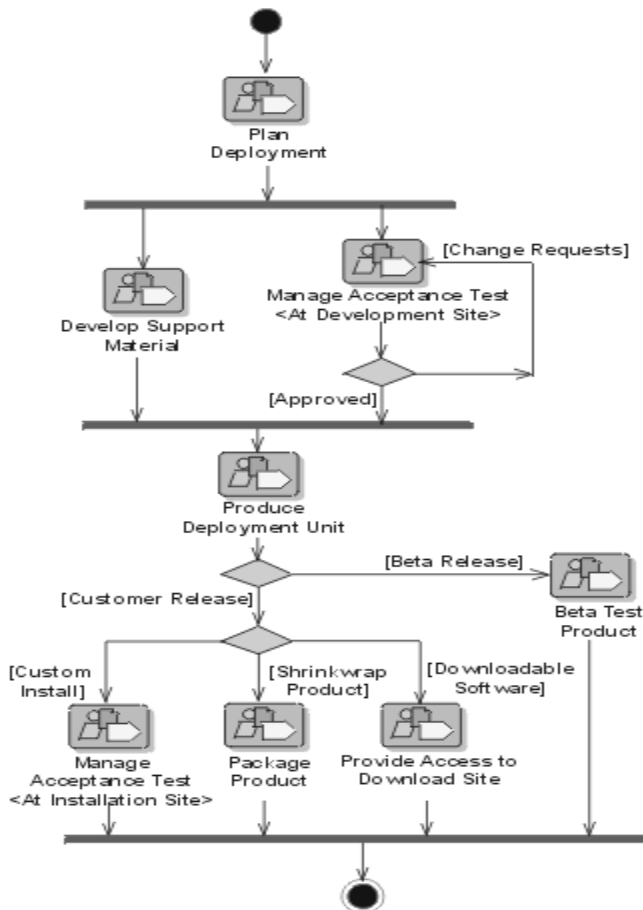
Tahapan ujicoba software untuk meyakinkan bahwa bisa digunakan sebagaimana yang diinginkan. Kegiatan yang dilakukan seperti pada Gambar 4.6.



Gambar 4.6 Test

4.7 Deployment

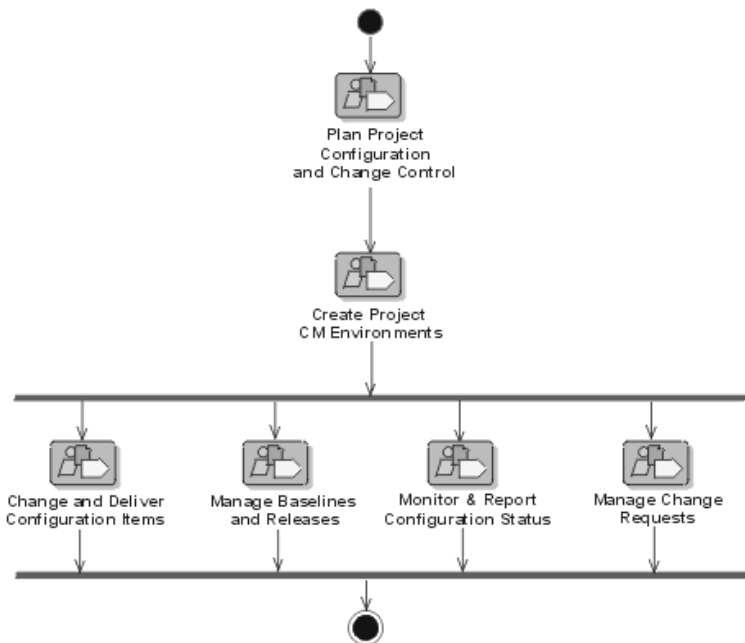
Tahapan penerapan software sistem informasi untuk digunakan sebenarnya oleh pengguna. Gambar 4.7 menunjukkan kegiatan yang dilakukan pada tahapan deployment.



Gambar 4.7 Deployment

4.8 Configuration & Change Management

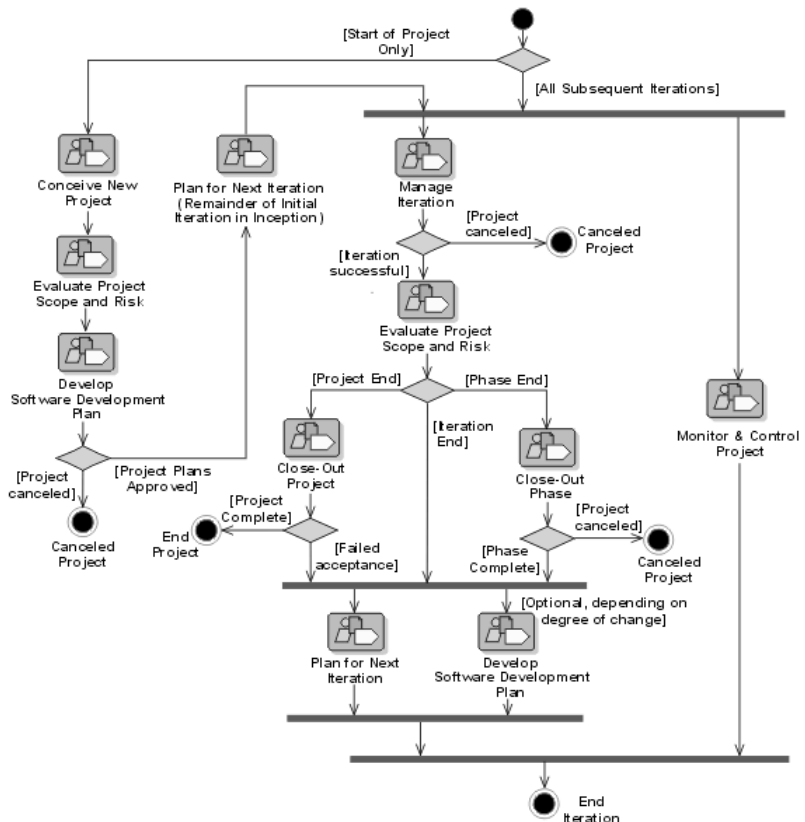
Seperti pada Gambar 4.8, tahapan ini merupakan penerapan *software* sistem informasi sehingga perlu dilakukan pengaturan konfigurasi, dan mengelola perubahan-perubahan yang harus dilakukan untuk menjaga supaya sistem informasi bisa diterapkan.



Gambar 4.8 Configuration & Change Management

4.9 Project Management

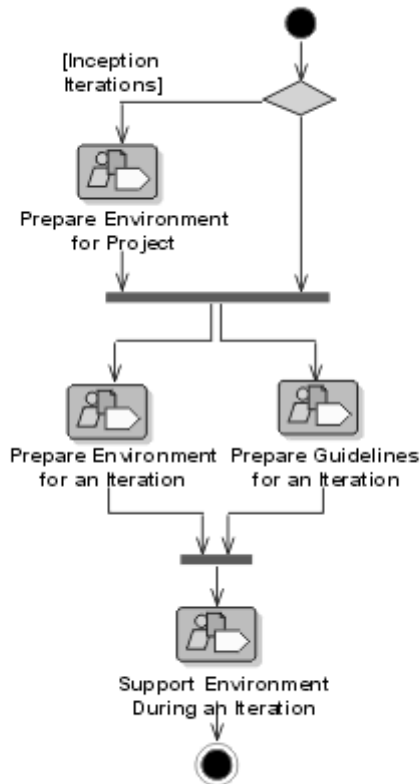
Pengembangan *software* merupakan pekerjaan besar, oleh karena itu memerlukan pengelolaan berbagai hal, baik teknis maupun non-teknis. Pengelolaan proyek dilakukan seperti pada Gambar 4.9.



Gambar 4.9 Project Management

4.10 Environment

Gambar 4.10 menjelaskan tahapan akhir berupa kegiatan untuk mempersiapkan lingkungan yang dapat menunjang pemakaian software sistem informasi secara baik dan benar.



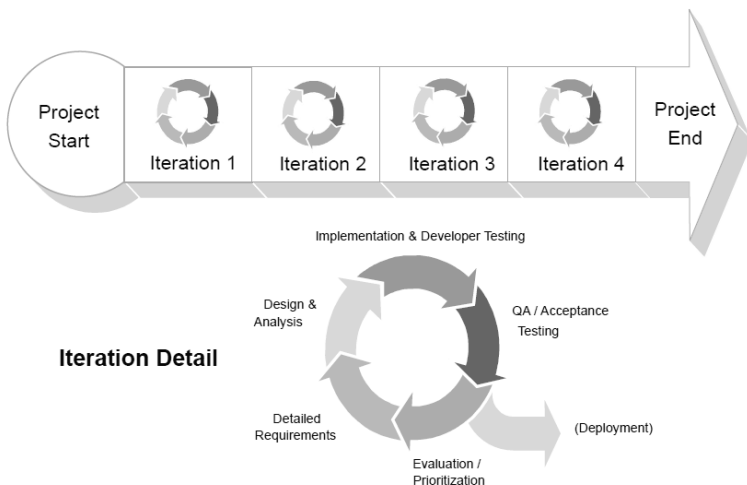
Gambar 4.10 Enviroment

BAB 5

SCRUM

5.1 Pengertian *Scrum*

Sebagai salah satu *framework* pengembangan *software Scrum* bermanfaat untuk menyelesaikan permasalahan yang rumit dan sering berubah-ubah dalam rangka menghasilkan sebuah produk *software* yang bermutu. Sebagaimana pada Gambar 5.1 *Scrum* pada dasarnya mengadopsi *waterfall* yang dilakukan secara iteratif untuk menghasilkan sebuah *software* bermutu.



Gambar 5.1 Alur Proses *Scrum*

Di era awal tahun 90-an *Scrum* pertama kali diperkenalkan dalam pengembangan *software*. *Scrum* bukanlah sebuah proses ataupun teknik untuk mengembangkan produk, tetapi lebih tepat merupakan *framework* dimana Tim *Scrum* dapat memasukkan bermacam-macam proses dan teknik untuk membuat sebuah produk.

Scrum dapat mengekspos mobilitas secara efektif dalam manajemen pengembangan *software* yang dijalankan oleh Tim *Scrum*, sehingga dapat meningkatkan mutu pembuatan *software*.

Beberapa kelebihan pengembangan *software* dengan penggunaan *Scrum* adalah:

1. Kemudahan dalam penyesuaian terhadap perubahan capaian yang diinginkan.
2. Tim besar dapat dipecah menjadi beberapa tim kecil yang disebut dengan Tim *Scrum* untuk memperlancar komunikasi, mengurangi biaya, dan saling memberdayakan satu sama lain.
3. Dokumentasi selama proses pelaksanaan tugas (*task*) dilakukan secara terus-menerus dalam setiap iterasi pengembangan.
4. *Scrum* dapat mengidentifikasi status *task* pada setiap iterasi.

Karena *Scrum* ini sangat fleksibel terhadap perubahan, Tim Penjualan harus selalu siap menerima perubahan selama dalam proses pengembangan *software* kapan saja,

sehingga di sisi lain hal ini merupakan kelemahan dari *Scrum*.

Scrum sebagai sebuah *framework* bisa dikatakan kurang bersifat preskriptif. Hal ini berbeda dengan metodologi yang lebih bersifat detail dalam mengatur personil di dalamnya. *Framework* kebalikannya, hanya merupakan sebuah kerangka yang dapat memberikan banyak ruang untuk improvisasi.

Scrum menerapkan pendekatan berkala (*iterative*) dan bertahap (*incremental*) supaya bisa meningkatkan prediktabilitas dan mengendalikan risiko.

Tiga pilar utama dalam setiap implementasi kontrol proses empiris, yaitu sebagai berikut:

1. Transparansi

Aspek-aspek penting dari proses yang berjalan ditinjau oleh pihak-pihak yang bertanggungjawab terhadap hasilnya. Transparansi mengharuskan aspek-aspek tersebut didefinisikan dengan menggunakan standar yang sama, sehingga semua peninjau memiliki pemahaman yang sama mengenai apa yang sedang ditinjaunya.

Sebagai contoh,

- a. Istilah-istilah pada proses yang sedang dijalankan harus dapat dimengerti oleh semua pihak
- b. Setiap pihak yang bekerja dan pihak yang menerima hasil pekerjaan harus memiliki pemahaman yang sama mengenai arti kata “Selesai”.

2. Inspeksi

Pengguna *Scrum* harus secara rutin memeriksa artefak *Scrum* (*Scrum Board*) beserta perkembangannya supaya perubahan yang dilakukan dapat selalu terdeteksi. Peninjauan sebaiknya tidak terlalu sering dilakukan karena dapat menyebabkan terhambatnya pekerjaan. Peninjauan paling bermanfaat jika dilakukan secara rutin, oleh peninjau yang kompeten, pada saat pekerjaan berjalan.

3. Adaptasi

Apabila peninjau mendapatkan satu atau lebih aspek dari proses mengalami deviasi di luar batasan yang dapat diterima, hingga hasil akhirnya menjadi tidak dapat diterima, maka proses atau materi yang diolah harus diatur ulang. Pengaturan ulang harus dibuat sesegera mungkin untuk meminimalisir deviasi yang lebih jauh.

5.2 Tim *Scrum*

Dalam pengembangan software menggunakan *Scrum*, orang-orang yang terlibat (Tim *Scrum*) dikelompokkan menjadi tiga jenis yaitu

1. *Product Owner*
2. *Tim Developer*
3. *Scrum Master*

Tim *Scrum* mengatur diri sendiri dan berfungsi antar-lintas. Tim yang mengatur dirinya sendiri menentukan cara terbaik untuk menyelesaikan pekerjaannya. Tim yang berfungsi antar-lintas memiliki semua kompetensi yang

dibutuhkan untuk menyelesaikan pekerjaan, tanpa mengandalkan pihak lain yang berada di luar anggota tim.

Model tim di dalam *Scrum* dirancang sedemikian rupa sehingga bisa menjaga fleksibilitas, kreatifitas dan produktifitas, demi mencapai hasil terbaik. Tim *Scrum* menyampaikan hasil pekerjaan yang sudah dianggap selesai secara berkala dan bertahap, ini dilakukan agar Tim *Scrum* bisa mendapatkan masukan sedini mungkin dari setiap hasil pekerjaan yang disampaikan.

1. *Product Owner*

Product Owner bertanggungjawab untuk meninjau produk yang disampaikan, dan memastikan untuk mendapatkan nilai produk yang maksimal dari Tim Pengembang. *Product Owner* merupakan satu-satunya orang yang bertanggungjawab untuk mengelola *Product Backlog*. Pengelolaan *Product Backlog* mencakup hal-hal sebagai berikut:

- a. Mendeskripsikan dengan jelas *item Product Backlog*
- b. Mengurutkan item di dalam *Product Backlog* untuk mencapai tujuan dan misi dengan cara terbaik
- c. Melakukan *review* terhadap hasil pekerjaan Tim *Developer* untuk mendapatkan hasil yang optimal
- d. Memastikan bahwa *Product Backlog* bersifat transparan, jelas, dan dapat dilihat oleh semua pihak, dan mendefinisikan apa yang akan dikerjakan oleh Tim *Scrum* selanjutnya

- e. Memastikan Tim *Developer* dapat memahami *task* yang didaftarkan dalam *Product Backlog*, termasuk batasannya.

Product Owner adalah satu orang dan bukan berupa sebuah komite. *Product Owner* dapat menerima aspirasi dari komite ke dalam *Product Backlog*, namun perubahan prioritas pada *Product Backlog* tetap menjadi wewenang *Product Owner*.

Dalam rangka menunjang keberhasilan *Product Owner* dalam menjalankan tugasnya, seluruh organisasi harus menghormati setiap keputusan yang dibuatnya. Keputusan dari *Product Owner* ini dapat dilihat dari isi dan urutan *Product Backlog*. Tidak ada seorang pun yang dapat memerintah Tim *Developer* untuk mengerjakan kebutuhan lain selain *Product Owner*. Dan Tim *Developer* pun tidak diperbolehkan untuk melakukan apapun yang diperintahkan oleh pihak lain selain *Product Owner*.

Product Owner sebenarnya dapat mengerjakan pekerjaan-pekerjaan di atas, atau menyerahkan pengerjaannya kepada Tim *Developer*. Namun demikian, pihak yang bertanggungjawab tetap hanya di *Product Owner*

2. Tim *Developer*

Tim *Developer* dalam *Scrum* terdiri dari para profesional yang bekerja untuk menyelesaikan *task* atau menghasilkan potongan produk (selanjutnya disebut Inkremen), yang akan dirilis pada setiap akhir *Sprint*. Hanya

anggota Tim *Developer* yang dapat mengembangkan Inkremen ini.

Tim *Developer* dibentuk dan didukung oleh organisasi untuk mengatur dan mengelola pekerjaannya secara mandiri. Sinergi yang ada di dalam Tim *Developer* bisa meningkatkan kinerja lebih efisiensi dan efektifitas.

- a. Harus bisa mengelola dirinya sendiri tanpa campur tangan dari luar. Artinya, tidak ada satu anggotapun (bahkan *Scrum Master*) yang diperbolehkan untuk memerintah Tim *Developer*.
- b. Tim *Developer* berfungsi antar-lintas, dan tim ini seharusnya merupakan tim profesional yang memiliki semua keahlian yang dibutuhkan untuk menghasilkan produk yang diinginkan.
- c. *Scrum* tidak mengenal adanya jabatan tertentu untuk anggota Tim *Developer*, apapun pekerjaan yang dikerjakan oleh masing-masing anggota tim tersebut, tetap statusnya sebagai *Developer*.
- d. Tim *Developer* tidak mengenal adanya sub-tim yang dikhususkan untuk bidang tertentu seperti pengujian atau analisa bisnis.
- e. Anggota Tim *Developer* boleh memiliki spesialisasi keahlian dan fokus di satu area tertentu, namun akuntabilitas dari hasil pekerjaan secara keseluruhan adalah milik Tim *Developer*. Tim *Developer* merupakan kelompok kecil yang keanggotaannya hanya berjumlah 7 ± 2 orang.

3. *Scrum Master*

Scrum Master bertanggung jawab untuk memastikan *Scrum* telah dipahami dan dilaksanakan, serta bisa menciptakan lingkungan yang kondusif untuk pengembangan produk. *Scrum Master* harus bisa memastikan bahwa Tim *Developer* benar-benar mengikuti teori, praktik, dan aturan main *Scrum*.

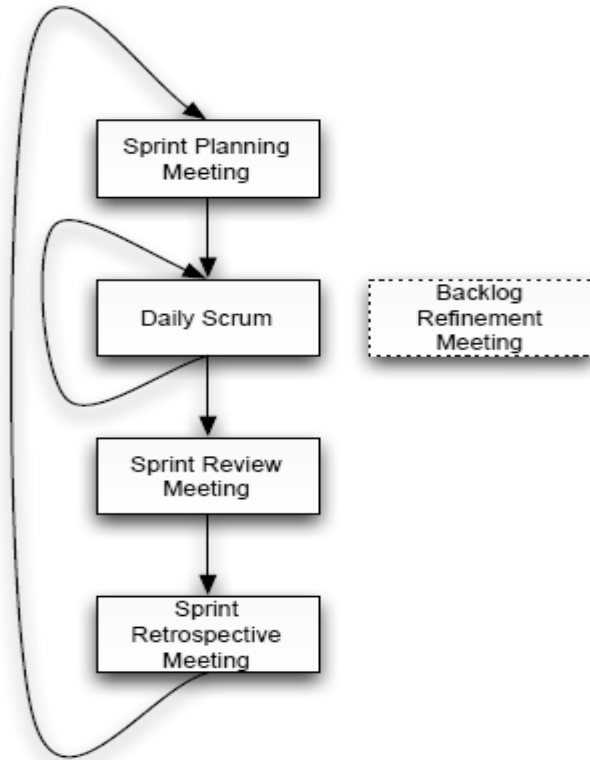
Scrum Master adalah seseorang yang berperan untuk melayani berbagai kebutuhan dari Tim *Developer*. *Scrum Master* membantu pihak di luar Tim *Scrum*, untuk memahami apakah interaksi mereka dengan Tim *Scrum* berjalan dengan baik atau tidak.

Scrum Master membantu setiap pihak untuk merubah interaksi-interaksi yang tidak bermanfaat sehingga bisa memaksimalkan nilai yang dihasilkan oleh Tim *Scrum*

5.3 *Scrum Meeting*

Scrum Master menyediakan berbagai fasilitas untuk keseluruhan pelaksanaan *Scrum Meeting*. Namun demikian, tidak memiliki kewenangan untuk mengambil keputusan dalam *meeting*.

Pelaksanaan *Scrum Meeting* dijalankan melalui beberapa proses secara iteratif seperti pada Gambar 5.2.



Gambar 5.2 *Scrum Flow*

1. *Sprint Planning Meeting*

Pekerjaan yang akan dilaksanakan di dalam *Sprint* direncanakan pada saat *Sprint Planning*. Perencanaan ini dilaksanakan oleh seluruh anggota Tim *Scrum* secara kolaboratif.

Sprint Planning dibatasi maksimum delapan jam untuk setiap *Sprint* yang berdurasi maksimal satu bulan. Untuk

Sprint yang lebih pendek, batasan waktunya biasanya lebih singkat. *Scrum Master* harus bisa memastikan bahwa acara ini dilaksanakan dan setiap peserta yang hadir benar-benar memahami tujuannya. *Scrum Master* mengedukasi Tim *Scrum* supaya bisa melaksanakannya dalam batasan waktu yang telah ditentukan.

2. *Daily Scrum and Sprint Execution*

Daily Scrum adalah kegiatan dengan batasan waktu maksimum selama 15 menit agar Tim *Developer* dapat mensinkronisasikan pekerjaan seluruh anggotanya, serta bisa membuat perencanaan untuk 24 jam ke depan. Hal ini dilakukan dengan meninjau pekerjaan mulai dari acara *Daily Scrum* terakhir dan memperkirakan pekerjaan yang dapat dilakukan sebelum melakukan *Daily Scrum* berikutnya.

Daily Scrum dilaksanakan pada waktu dan tempat yang sama setiap hari supaya tidak rumit pengaturannya. Pada saat pertemuan, Tim *Developer* harus menjelaskan hal-hal berikut ini:

1. Apa yang telah dilakukan sehingga *Sprint Goal* tercapai?
2. Apa yang akan dilakukan hari ini untuk mencapai *Sprint Goal*?
3. Apakah ada hambatan yang dapat menghalangi untuk mencapai *Sprint Goal*?

3. *Sprint Review Meeting*

Sprint Review diadakan di akhir *Sprint* untuk meninjau Inkremen dan perubahan *Product Backlog* jika diperlukan. Pada *Sprint Review*, Tim *Scrum* dan *stakeholder* berkolaborasi untuk membahas apa yang telah dikerjakan dalam *Sprint* yang baru saja diselesaikan. Dari pembahasan tersebut, peserta *Sprint Review* berkolaborasi menentukan apa yang dapat dikerjakan di *Sprint* berikutnya, untuk optimalisasi nilai produk. Pertemuan ini bersifat informal, dan presentasi dari Inkremen diharapkan dapat mengumpulkan masukan dan menumbuhkan semangat kolaborasi.

Acara ini diberi batasan waktu maksimum selama empat jam untuk *Sprint* yang berdurasi satu bulan. Untuk *Sprint* yang lebih pendek, maka batasan waktunya juga biasanya dibuat menjadi lebih singkat. *Scrum Master* harus bisa memastikan pelaksanaan acara ini berjalan lancar, dan setiap peserta yang hadir pada pertemuan tersebut benar-benar memahami maksud dan tujuannya, tanpa ada kebingungan sama sekali.

Pada titik manapun, jumlah pekerjaan yang tersisa hingga akhir tujuan pengembangan dapat dijumlahkan. *Product Owner* memantau total sisa pekerjaan ini setidaknya di setiap *Sprint Review*. *Product Owner* membandingkan kondisi saat ini dengan jumlah sisa pekerjaan di *Sprint Review* sebelumnya guna meninjau perkembangan menuju tujuan akhir dengan waktu yang diharapkan. Informasi ini transparan untuk setiap *stakeholder*.

4. *Sprint Retrospective Meeting*

Sprint Retrospective adalah waktu dimana Tim *Scrum* dapat melakukan inspeksi dan membuat perencanaan lanjutan tentang upaya-upaya yang harus dilakukan untuk peningkatan di *Sprint* berikutnya.

Sprint Retrospective dilangsungkan setelah *Sprint Review* selesai dan sebelum *Sprint Planning* berikutnya. Ini adalah acara dengan batasan waktu maksimum selama tiga jam untuk *sprint* yang berdurasi satu bulan.

Batasan setiap *sprint* berbeda-beda, maka *sprint* yang lebih pendek, batasan waktunya biasanya lebih singkat. *Scrum Master* harus bisa memastikan bahwa acara ini dilaksanakan dan setiap peserta pertemuan memahami tujuannya. *Scrum Master* mengedukasi Tim *Scrum* untuk melaksanakannya dalam batasan waktu yang telah ditentukan. *Scrum Master* berpartisipasi sebagai rekan kerja yang bertanggungjawab terhadap berjalannya proses *Scrum*.

Tujuan yang ingin dicapai dari acara *Sprint Retrospective* adalah:

- a. Melakukan *review* terhadap personil, hubungan antar personil, proses, dan peralatan pada *Sprint* yang telah selesai.
- b. Mengidentifikasi dan mengurutkan hal-hal utama yang berjalan baik, dan hal-hal yang berpotensi untuk ditingkatkan; dan,
- c. Membuat rencana implementasi, dengan tujuan peningkatan cara-cara kerja Tim *Scrum*.

Scrum Master mendukung Tim *Scrum* untuk membuat peningkatan kinerja dengan kerangka kerja *Scrum*, sehingga *sprint* berikutnya dapat dilakukan dengan lebih efektif dan menyenangkan. Pada tahap *Sprint Retrospective*, Tim *Scrum* dapat merencanakan cara untuk meningkatkan kualitas dari produk, dengan merubah definisi “Selesai” sebagaimana dibutuhkan.

Di akhir *Sprint Retrospective*, Tim *Scrum* harus dapat mengidentifikasi peningkatan-peningkatan yang harus diimplementasikan pada *sprint* berikutnya. *Sprint Retrospective* memberi kesempatan secara formal kepada Tim *Scrum* untuk lebih fokus pada kegiatan *review* dan adaptasi.

5. Backlog Refinement Meeting

Product Backlog Refinement adalah kegiatan menambahkan rincian, mengestimasi dan mengurutkan item di dalam *Product Backlog*. Kegiatan ini dilakukan secara terus-menerus, di mana *Product Owner* dan Tim *Developer* berkolaborasi untuk merinci item *Product Backlog*. Pada saat *Product Backlog Refinement*, item ditinjau-ulang dan dilakukan revisi. Tim *Scrum* sendiri yang menentukan bagaimana dan kapan proses *refinement* diadakan. *Refinement* biasanya mencakup tidak lebih dari 10% dari kapasitas Tim *Developer*. Item *Product Backlog* dapat diperbarui kapanpun juga oleh *Product Owner* atau siapapun atas arahan dari *Product Owner* kapan saja diinginkannya.

Kebanyakan PBI pada awalnya memerlukan perbaikan karena ukurannya begitu besar dan belum dipahami secara penuh oleh Tim *Scrum*. Oleh karena itu, Tim *Scrum* masih memerlukan waktu untuk bisa memahami dan bisa menghasilkan produk.

Dalam *meeting* ini Tim *Scrum* memperkirakan seberapa besar upaya yang harus dilakukan untuk bisa melengkapi seluruh item *Product Backlog*, dan menyediakan informasi teknis dalam membantu *Product Owner* untuk menetapkan skala prioritas.

Seorang *Scrum Master* yang handal bisa menolong Tim *Scrum* untuk mengidentifikasi pekerjaan-pekerjaan yang masih belum jelas, menjadi pekerjaan yang jelas, sehingga bisa dilakukan pekerjaan berikutnya termasuk pengujian dan *refactoring*.

Sebenarnya *meeting* ini tidak memiliki nama yang formal, dan kadang disebut dengan *Backlog Grooming*, *Backlog Maintenance*, atau *Story Time*.

5.4 Artefak *Scrum*

Artefak *Scrum* merepresentasikan pekerjaan atau nilai, bertujuan untuk menyediakan transparansi, dan kesempatan-kesempatan untuk peninjauan dan adaptasi. Artefak yang didefinisikan oleh *Scrum* secara khusus dirancang untuk meningkatkan transparansi dari informasi penting dalam pembuatan produk, dengan begitu semua pihak dapat memiliki pemahaman yang sama terhadap artefak.

1. *Product Backlog*

Product Backlog adalah daftar terurut, dari setiap hal yang mungkin dibutuhkan dalam pembuatan produk, dan juga merupakan sumber utama, dari daftar kebutuhan mengenai semua hal yang perlu dilakukan terhadap produk. *Product Owner* bertanggungjawab terhadap *Product Backlog*, termasuk isinya, ketersediaannya, dan urutannya.

Product Backlog harus selalu diperbaharui sesuai dengan kebutuhan sampai dengan target akhir pekerjaan, bisa jadi *Product Backlog* ini tidak pernah selesai karena produk terus menerus dikembangkan. Pada awal pembuatannya hanya terjabar daftar kebutuhan yang hanya diketahui dan dipahami pada saat itu. *Product Backlog* berkembang seiring dengan berkembangnya produk dan lingkungan dimana produk tersebut digunakan. *Product Backlog* bersifat dinamis, senantiasa berubah sampai produk dapat dikatakan layak, kompetitif di pasar, dan bermanfaat bagi penggunaannya. Selama produk masih eksis digunakan maka *Product Backlog* juga akan tetap eksis.

Product Backlog menjabarkan semua fitur, fungsi, kebutuhan, penyempurnaan dan perbaikan terhadap produk yang akan dirilis. Item *Product Backlog* memiliki atribut deskripsi, urutan, estimasi dan nilai bisnis.

Bersamaan dengan digunakannya produk dan semakin bertambahnya nilai dari produk, dan bertambahnya masukan dari pasar, *Product Backlog* semakin berkembang menjadi lebih besar. Daftar kebutuhan tidak pernah berhenti berubah, sehingga *Product Backlog* dapat dikatakan sebagai artefak yang hidup. Perubahan dalam kebutuhan bisnis, keadaan pasar, ataupun teknologi dapat menyebabkan perubahan pada *Product Backlog*.

Sering kali dibuat lebih dari satu Tim *Scrum* untuk mengerjakan satu produk yang sama. Satu *Product Backlog* digunakan untuk menggambarkan pekerjaan selanjutnya terhadap sebuah produk. Bisa ditambahkan pula sebuah atribut, untuk mengelompokkan item *Product Backlog*.

Item *Product Backlog* pada urutan yang lebih atas biasanya lebih jelas dan lebih detail dibandingkan item di bawahnya. Estimasi dengan presisi tinggi diberikan berdasarkan tingkat kejelasan dan detail yang tinggi. Semakin bawah urutan dari item *Product Backlog*, maka semakin rendah pula tingkat kedetailannya. Item *Product Backlog* yang akan dikerjakan oleh Tim *Developer* untuk *sprint* yang mendatang di-*refine* supaya setiap item yang dikerjakan dapat di-“Selesai”kan dalam satu *sprint*. Item *Product Backlog* yang dianggap dapat di-“Selesai”-kan oleh Tim *Developer* dalam satu *sprint*, maka dikatakan “Siap” untuk diseleksi pada saat *Sprint Planning*. Item *Product Backlog* biasanya akan memiliki tingkat transparansi yang tinggi karena adanya aktifitas *refinement* ini.

Tim *Developer* bertanggungjawab terhadap seluruh estimasi. *Product Owner* dapat mempengaruhi Tim *Developer* dengan cara memberi bantuan untuk memahami *Product Backlog* dan membuat pengecualian terhadap *Product Backlog*, namun orang-orang yang akan mengerjakan item *Product Backlog*-lah yang akan membuat estimasi final.

Berbagai macam praktik proyeksi terhadap *trending* telah digunakan untuk memperkirakan kemajuan, seperti *burndown chart* atau *burnup chart*. Hal ini telah terbukti berguna. Namun hal ini tidak menggantikan pentingnya peran empirisme. Di lingkungan yang kompleks, apa yang akan terjadi di masa mendatang tidak dapat diketahui

sebelumnya. Hanya apa yang telah terjadi yang dapat digunakan untuk membuat keputusan di masa mendatang.

2. *Product Backlog Item (PBI)*

Beberapa hal yang diperhatikan dalam pencatatan produk sebagai berikut

1. Menunjukkan “Apa” daripada “Bagaimana” yang diinginkan pengguna
2. Sering menulis form *User Story*
3. Mempunyai ketentuan
4. Memiliki kriteria *item-specific acceptance*
5. Dijalankan secara tim, idealnya dalam setiap unit (*story point*)
6. Dijalankan oleh 2-3 orang, 2-3 hari, atau lebih kecil untuk tim yang lebih pakar

3. *Sprint Backlog*

Sprint Backlog adalah sekumpulan item *Product Backlog* yang telah dipilih untuk dikerjakan di *Sprint*, juga di dalamnya rencana untuk mengembangkan potongan tambahan produk dan merealisasikan *Sprint Goal*. *Sprint Backlog* adalah perkiraan mengenai fungsionalitas apa yang akan tersedia di Inkremen selanjutnya dan pekerjaan yang perlu dikerjakan untuk menghantarkan fungsionalitas

tersebut menjadi potongan tambahan produk yang “Selesai”.

Sprint Backlog menampilkan semua pekerjaan yang dibutuhkan untuk mencapai *Sprint Goal* yang dibuat oleh Tim Pengembang.






















Sprint Backlog adalah sebuah rencana yang cukup detail, di mana perubahan-perubahannya di tengah *Sprint* bisa dipahami saat *Daily Scrum Meeting*. Tim Pengembang memodifikasi *Sprint Backlog* sepanjang *Sprint* berlangsung, dan *Sprint Backlog* dapat berubah kapanpun juga sepanjang *Sprint*. Perubahan ini terjadi seiring dengan berkerjanya Tim Pengembang sesuai rencana pada saat itu, dan semakin meningkatnya wawasan tim untuk mencapai tujuan *Sprint*.

Dengan bertambahnya pekerjaan baru, Tim Pengembang menambahkannya ke dalam *Sprint Backlog*. Dengan dikerjakannya atau diselesaikannya pekerjaan, estimasi sisa pekerjaan juga diperbaharui. Ketika ada elemen dari perencanaan tidak dibutuhkan lagi, maka elemen tersebut dikeluarkan dari *Sprint Backlog*. Hanya Tim Pengembang yang dapat merubah *Sprint Backlog* pada saat *Sprint* sedang berjalan.

Sprint Backlog sangat transparan, menggambarkan secara *real-time* pekerjaan yang akan diselesaikan oleh Tim Pengembang pada saat *Sprint*, dan ia sepenuhnya menjadi milik Tim Pengembang.

Di titik manapun dalam sebuah *Sprint*, jumlah sisa pekerjaan dalam *Sprint Backlog* harus dapat dijumlahkan. Tim *Developer* harus bisa memantau sisa pekerjaan ini, setidaknya di setiap *Daily Scrum*, untuk memproyeksikan kemungkinan pencapaian *Sprint Goal*. Dengan memantau sisa pekerjaan ini sepanjang *Sprint*, maka Tim *Developer*

akan bisa mengelola perkembangan pekerjaan.

Committed Backlog Items	Tasks Not Started	Tasks In Progress	Tasks Completed
	  		 
	  		
	     		
			

Gambar 5. *Sprint Backlog*

4. *Sprint Task*

- 1. Menentukan bagaimana cara untuk mencapai apa yang ditetapkan sebagai PBI
- 2. Memerlukan waktu 1 hari kerja atau kurang
- 3. Sisa pekerjaan diestimasi ulang setiap hari, biasanya dalam beberapa jam
- 4. Selama Sprint Execution, a *point person* diperbolehkan menjadi relawan

5. Tanggung jawab utama untuk task (tugas)
6. Dimiliki oleh keseluruhan ti; diharapkan melalui kolaborasi

5. *Sprint Burndown Chart*

1. Menunjukkan total jam tugas tim yang tersisa di dalam satu *Sprint*
2. Mengestimasi ulang secara harian, agar bisa naik sebelum turun
3. Memberikan fasilitas kepada tim *self-organization*
4. Variasi mewah seperti melakukan rincian berdasarkan *point person* atau penambahan trend garis, dan cenderung mengurangi efektivitas tetapi mendorong kolaborasi
5. Kelihatannya seperti ide bagus dalam keseharian *Scrum*, tetapi dalam prakteknya jarang digunakan sebagai laporan manajemen. *Scrum Master* harus memutuskan penggunaan *chart* jika membebani pekerjaan terhadap tim *self-organization*.

6. *Product/Release Burndown Chart*

1. Melacak upaya *Product Backlog* yang tersisa dari satu *sprint* ke *sprint* berikutnya
2. Bisa menggunakan alternatif unit seperti *Story Points* untuk sumbu Y
3. Menggambarkan trend historis untuk menyesuaikan perkiraan

5.5 Inkremen

Inkremen (tambahan potongan produk) merupakan penggabungan dari seluruh item *Product Backlog* yang diselesaikan pada *Sprint* yang berjalan dan nilai-nilai dari Inkremen *sprint-sprint* sebelumnya.

Pada akhir *Sprint*, inkremen terbaru harus “Selesai”, yang artinya berada dalam kondisi yang berfungsi penuh dan memenuhi definisi “Selesai” yang dibuat oleh Tim *Scrum*. Terlepas apakah *Product Owner* akan merilis produknya, produk harus selalu berada dalam kondisi yang berfungsi penuh.

5.6 Transparansi Artefak

Transparansi merupakan prinsip penting di dalam *Scrum*. Keputusan-keputusan untuk mengoptimalkan nilai produk dan mengendalikan risiko dibuat berdasarkan keadaan artefak hingga saat ini. Pada titik di mana transparansi berada pada tingkat tinggi, keputusan yang dibuat semakin dapat dipercaya. Pada titik di mana transparansi berada pada tingkat rendah, keputusan dapat dimanipulasi, nilai produk akan menurun dan resiko akan meningkat.

Scrum Master harus bekerja dengan *Product Owner*, Tim *Developer* dan pihak-pihak lain untuk bersama-sama memahami apakah seluruh artefak sudah sepenuhnya transparan. Ada banyak praktik untuk menangani belum sepenuhnya transparansi; dalam keadaan ini, *Scrum Master* harus membantu semua pihak untuk menerapkan praktik yang sesuai terhadap hilangnya transparansi. *Scrum Master*

dapat mendeteksi hilangnya transparansi dengan meninjau semua artefak, mengamati pola-pola yang terjadi, menyimak apa yang dikatakan, dan melihat perbedaan antara apa yang diharapkan dengan hasil yang sebenarnya.

Tugas seorang *Scrum Master* adalah bekerja bersama Tim *Scrum* dan organisasi yang terlibat untuk meningkatkan tingkat transparansi dari artefak-artefak yang digunakan. Pekerjaan ini biasanya membutuhkan pembelajaran, pendekatan persuasif, serta perubahan. Transparansi tidak terjadi dalam semalam, melainkan sebuah perjalanan jangka panjang.

5.7 Definisi “Selesai”

Ketika sebuah item *Product Backlog* atau Inkremen dikatakan “Selesai”, maka setiap pihak harus sepakat dan sama-sama mengerti dengan apa yang dimaksud dengan “Selesai”, tanpa adanya kesalahpahaman sama sekali. Walaupun definisi ini berbeda-beda antar tim *Scrum*, sesama anggota tim harus memiliki pemahaman yang sama mengenai pekerjaan yang harus diselesaikannya guna memastikan adanya transparansi. Ini adalah definisi “Selesai” untuk Tim *Scrum* dan ini digunakan untuk memeriksa apakah pekerjaan untuk mengembangkan Inkremen dianggap selesai.

Definisi yang sama akan membimbing Tim *Developer* untuk mengetahui berapa banyak item *Product Backlog* yang bisa diambil pada saat *Sprint Planning*. Tujuan dari setiap *Sprint* adalah untuk menghantarkan Inkremen, yang berpotensi untuk dirilis, yang memenuhi definisi “Selesai” terkini yang dibuat oleh Tim *Scrum*.

Tim *Developer* menghantarkan Inkremen yang berfungsi setiap *Sprint*. Inkremen ini dapat digunakan, supaya *Product Owner* dapat merilis produk tersebut sesegera mungkin jika diinginkan. Apabila definisi “Selesai” untuk sebuah Inkremen adalah bagian dari konvensi, standar atau panduan pengembangan dari organisasi, setiap Tim *Scrum* harus mengikuti seluruhnya sebagai *minimum requirement*.

Apabila “Selesai” untuk sebuah Inkremen bukan merupakan bagian dari konvensi, standar atau panduan pengembangan dari organisasi, Tim *Developer* harus membuat definisi “Selesai” yang pantas untuk produk yang dikembangkannya. Apabila ada beberapa Tim *Scrum* yang mengembangkan sistem atau produk yang sama, seluruh Tim *Developer* di setiap Tim *Scrum* harus menentukan definisi “Selesai” yang sama bersama-sama.

Setiap Inkremen merupakan gabungan dari Inkremen *Sprint-sprint* sebelumnya dan diuji secara teliti, untuk memastikan bahwa setiap Inkremen dapat berfungsi secara penuh sebagaimana mestinya.

Seiring dengan bertambah dewasanya Tim *Scrum*, mereka diharapkan untuk membuat definisi “Selesai” yang lebih baik dan ketat lagi demi peningkatan kualitas. Produk atau sistem manapun harus memiliki definisi “Selesai” yang merupakan sebuah standar untuk pekerjaan yang akan dilakukan.

5.8 Poin Penting

Di dalam *framework Scrum* peran, artefak, acara dan

aturan sifatnya tidak dapat diubah walaupun penggunaan *Scrum* secara sebagian sangat memungkinkan, namun berakibat bahwa hasilnya bukan merupakan *Scrum*. Bisa dikatakan *Scrum* kalau seluruh komponen ada dan berfungsi dengan baik sebagai pembungkus untuk teknik, metodologi maupun praktik lain.

BAB 6

SISTEM INFORMASI

6.1 Istilah Sistem Informasi

H. M. Jogianto (2006) memberikan definisi tentang Sistem informasi, sebagai berikut:

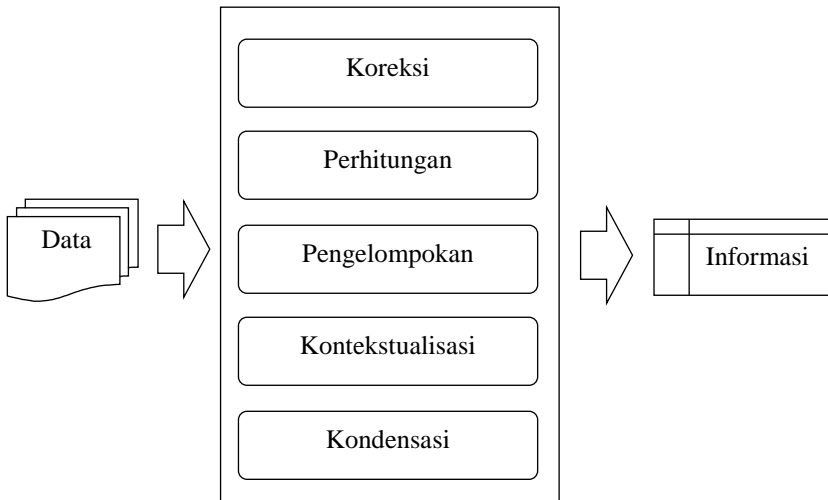
“Sistem adalah suatu jaringan kerja dari prosedur-prosedur yang saling berhubungan, berkumpul bersama-sama untuk melakukan suatu kegiatan atau untuk menyelesaikan sasaran yang tertentu”

Kenneth Laudon C (2012) mendefinisikan Sistem Informasi sebagai berikut:

“a set of interrelated components that collect (or retrieve), process, store, and distribute information to support decision making and control in an organization. In addition to supporting decision making, coordination and control, information systems may also help managers and workers analyze problems, visualize complex subjects, and create new products.”

Berdasarkan kedua definisi tersebut, dapat dikatakan bahwa pengertian sistem informasi bisa dilihat dari dua sisi yaitu sisi prosedur maupun dari sisi komponen pendukungnya. Intinya adalah sebuah sistem yang memanfaatkan data dengan peralatan teknologi yang

memadai sehingga bisa menghasilkan informasi yang dapat digunakan dalam menunjang berbagai proses bisnis mulai dari level operasional sampai dengan level pengambilan keputusan tertinggi.



Gambar 6.1 Hubungan Data dan Informasi

Dalam era digital seperti saat ini sistem informasi tidak terlepas dari bantuan perangkat komputer dan jaringan internet/intranet, sehingga sering disebut juga dengan sebutan lengkap *Computer Based Information System*. Dengan demikian bahwa sebuah sistem informasi sudah pasti mengacu pada suatu sistem informasi yang menggunakan komputer sebagai alat bantu, dan harus bisa diakses melalui jaringan komputer global.

Seiring dengan perkembangan teknologi informasi terutama perangkat *mobile* seperti *tablet*, *smartphone*, *laptop*, maka sistem informasi saat ini dituntut untuk bisa

diakses dengan cepat dari dan kapan saja.

Sistem informasi memiliki beberapa ciri atau karakteristik yaitu komponen sistem (*componenents*), batasan sistem (*boundary*), lingkungan sistem (*environments*), penghubung sistem (*interface*), Masukan sistem (*input*), pengolahan (*processing*), keluaran sistem (*output*), sasaran (*objective*) dan tujuan (*goal*)

Sebagaimana yang ditunjukkan pada Gambar 6.1, sebuah sistem informasi akan memasukkan data, kemudian memprosesnya sedemikian rupa sesuai dengan jenis atau karakter masing-masing sistem, sehingga akhirnya dihasilkan informasi yang berkualitas yaitu valid bebas dari kesalahan, tepat waktu, dan relevan dengan kebutuhan proses bisnis.

6.2 Sumber Daya Teknologi Informasi

Sebagaimana sumber daya lain, sumber daya teknologi informasi memiliki peranan yang sangat penting dalam menunjang kesuksesan penerapan sistem informasi di dalam suatu *enterprise*. Pemanfaatan teknologi informasi secara tepat dapat mendukung sepenuhnya keseluruhan rangkaian proses bisnis dalam sebuah *enterprise*.

Sumber daya teknologi informasi pada dasarnya terdiri dari tiga komponen utama sebagai berikut.

1. Information Technology

Teknologi informasi adalah salah satu pendukung utama suatu *enterprise*. Infrastruktur teknologi informasi yang dibutuhkan untuk mendukung kelancaran proses bisnis secara menyeluruh.

Infrastruktur ini mencakup *software*, *hardware* dan *communication technology*.

2. *Information*

Sumber pengambilan kebijakan dan keputusan oleh suatu *enterprise* yang bersasar dari data-data yang terkumpul dari dalam dan luar *enterprise*.

3. *People*

Yang menjadi kunci utama sebagai pelaku sistem yang telah disediakan. karena pada dasarnya sebuah sistem bukanlah sebuah robot pintar yang bisa berjalan sendiri dan mengambil keputusan sendiri.

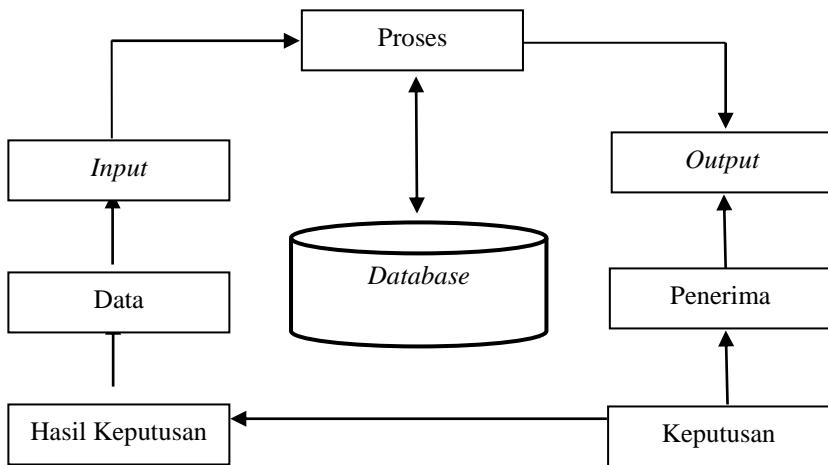
Keterlibatan orang dalam menjalankan sistem berbasis komputer sangat dominan, dan sangat menentukan kualitas dan keberlangsungan sumber daya lainnya. Oleh karena itu, dalam pengembangan arsitektur *enterprise*, pembahasan tentang *who* artinya siapa yang terlibat menjadi perhatian khusus sehingga dimasukkan ke dalam pembahasan *framework*.

6.3 Teknologi Pendukung

Secara umum komponen utama yang menyusun sebuah sistem informasi berbasis komputer dan jaringan komputer dapat dijelaskan melalui siklus sistem informasi seperti pada Gambar 6.2.

Teknologi *database* (penyimpanan data) merupakan *core technology* dari suatu sistem informasi karena data dan informasi akan disimpan dan dikelola sedemikian rupa secara efisien dan efektif di dalamnya, sehingga akan

menjadi sumber daya yang sangat penting bagi berbagai macam sistem informasi yang terkait. Oleh karena itu, dalam pembuatan sistem informasi harus dipilih teknologi *database* yang *reliable* disesuaikan dengan kapasitas data yang harus disediakan, terutama untuk *enterprise* yang menjalankan bisnis dengan menggunakan data-data sangat penting berbagai pihak seperti bisnis perbankan.



Gambar 6.2 Siklus Sistem Informasi

Komponen pendukung sebuah sistem informasi berbasis komputer adalah sebagai berikut.

1. *Software*

Program aplikasi komputer yang secara langsung mendukung jalannya proses bisnis, dimana dalam pembuatannya memerlukan *programming language*, *database management system* (DBMS), *operating system*, sistem sekuriti, dll.

2. *Hardware*

Perangkat keras merupakan perangkat dasar berupa barang elektronik yang diperlukan untuk menjalankan berbagai *software* pendukung sebuah sistem informasi, seperti diantaranya komputer, *harddisk*, *flashdisk*, printer, *scanner*, dll.

3. *Communication*

Teknologi komunikasi merupakan teknologi infrastruktur untuk mengfungsikan dan mengoptimalisasikan *software* dan *hardware*. Teknologi ini mencakup berbagai alat untuk membangun suatu jaringan komputer baik intranet atau internet. Teknologi yang bisa digunakan saat ini ada banyak alternatifnya, tidak hanya teknologi jaringan berbasis kabel, termasuk teknologi *wireless* yang sangat memudahkan proses komunikasi antar berbagai alat komputer seperti *Wifi*, jaringan GSM, dll.

6.4 Tipe Sistem Informasi

Dalam suatu *enterprise* terdapat banyak pihak-pihak yang akan memanfaatkan teknologi informasi untuk menjalankan proses bisnisnya masing-masing. Berbagai jenis sistem informasi dibutuhkan disesuaikan dengan tingkat manajerial atau spesifikasi bisnis yang dilakukan. Dengan demikian, sistem informasi bisa dikelompokkan berdasarkan kebutuhan fungsi manajerial *enterprise*.

Sistem informasi yang dibutuhkan untuk mendukung proses bisnis *enterprise* secara umum bisa dikelompokkan ke dalam empat kategori sebagai berikut:

1. TPS (*Transaction Processing System*)

Sistem yang sifatnya transaksional dan operasional, dan merupakan jenis sistem informasi yang akan memberikan masukan data (*penyedia raw data*) bagi sistem informasi yang lainnya. Sistem ini pada dasarnya hanya mendukung kegiatan operasional bisnis yang menyangkut berbagai macam transaksi. Oleh karena itu, kedudukan sistem ini sangat penting karena sebagai fondasi sistem yang menjadi sumber data bagi sistem informasi yang lain di atasnya, dan akan sangat menentukan kualitas informasi yang dihasilkan oleh sistem informasi MIS, DSS dan EIS. Dengan demikian, proses validasi data merupakan hal yang mutlak dilakukan dalam pembuatan sistem ini, agar dijamin tidak terjadi kesalahan pada proses pemasukan data.

2. MIS (*Management Information System*)

Sistem informasi yang berada satu level di atas TPS. Sistem ini yang mendukung kegiatan bisnis pada level manajerial dengan memanfaatkan data-data yang telah diproses melalui TPS. Data yang dikumpulkan oleh TPS, selanjutnya akan diolah lebih lanjut, sehingga menghasilkan suatu informasi dalam mengontrol pelaksanaan kegiatan bisnis. Untuk mengolah data menjadi informasi dalam sistem informasi ini, biasanya cukup menggunakan fasilitas pengolahan *query* dengan bantuan *Structure Query Language* (SQL). *Query* yang dibangun disesuaikan dengan kebutuhan bisnis setiap sistem sebagaimana karakteristik proses bisnisnya masing-masing.

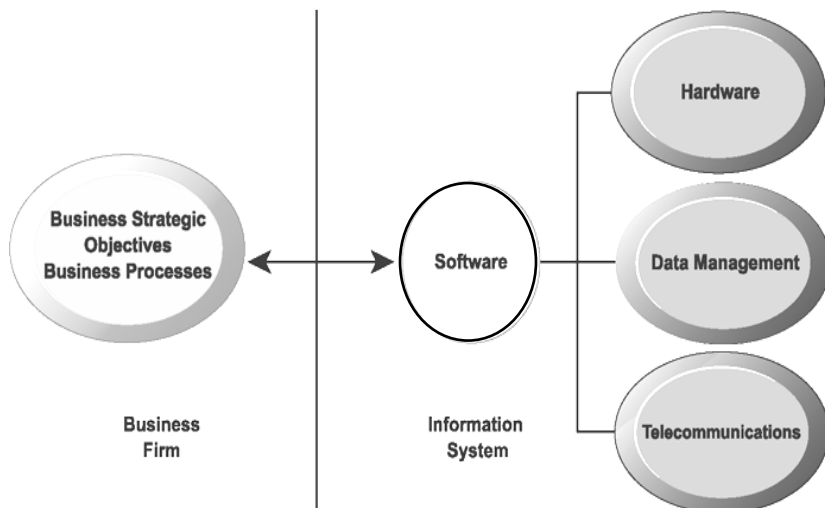
3. DSS (*Decision Support System*)

Sistem ini mendukung kegiatan yang bersifat analitis dengan memanfaatkan bongkahan data berukuran besar untuk diproses lebih lanjut, sehingga menghasilkan

pengetahuan tentang apa sebenarnya yang terkandung dalam bongkahan data tersebut. Untuk pengolahan datanya bisa digunakan teknologi *Data Mining*, sehingga bisa menggali informasi yang penting dari suatu bongkahan data yang sangat penting dalam proses pengambilan keputusan. Bongkahan data merupakan kumpulan data dalam kurun waktu yang cukup lama, perlu dilakukan proses denormalisasi terhadap struktur datanya agar pemrosesan lebih cepat. Dalam hal ini istilah *Data Warehouse* merupakan salah satu pekerjaan yang dilakukan dalam membangun DSS. Dengan demikian *Data Warehouse* dan *Data Mining* merupakan dua komponen utama yang saling melengkapi dalam membentuk sistem informasi jenis ini.

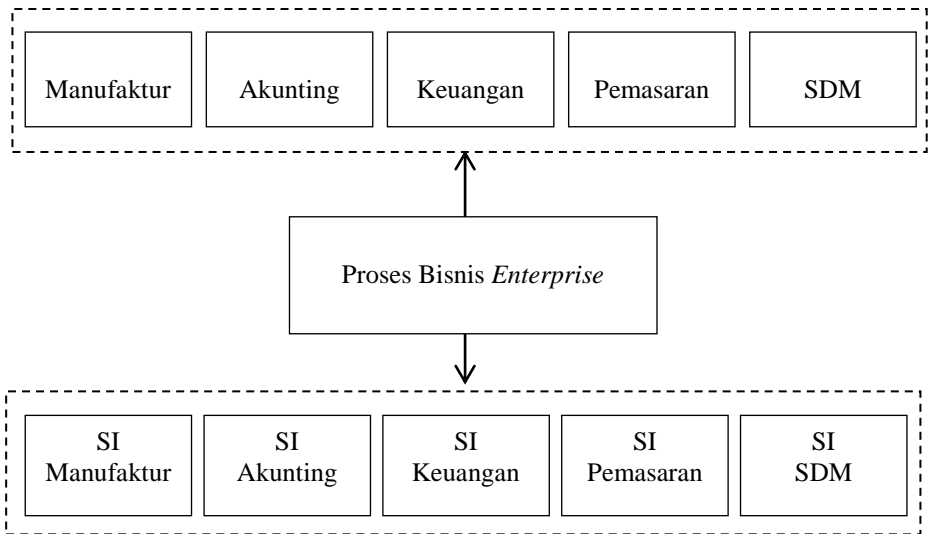
4. EIS (*Executive Information System*)

Sistem yang digunakan untuk kepentingan para manajer puncak dalam suatu *enterprise* agar bisa memonitor kinerja bisnis secara keseluruhan dan *up-to-date* hanya dengan melihat informasi yang ditampilkan dalam bentuk grafik. Berkaitan dengan sistem informasi jenis ini, dikenal juga istilah *War Room* yaitu mengacu pada suatu ruangan khusus yang dipergunakan untuk pengolahan data *enterprise* secara menyeluruh baik data dari dalam maupun dari luar *enterprise*, yang ditampilkan secara visual dalam berbagai bentuk grafik, agar informasi yang ditampilkan benar-benar dapat dipahami secara mudah dan cepat oleh para manajer puncak dalam membantu pengambilan keputusan, kebijakan, strategi bisnis.



Gambar 6.3 Hubungan Sistem Informasi Dan Proses Bisnis

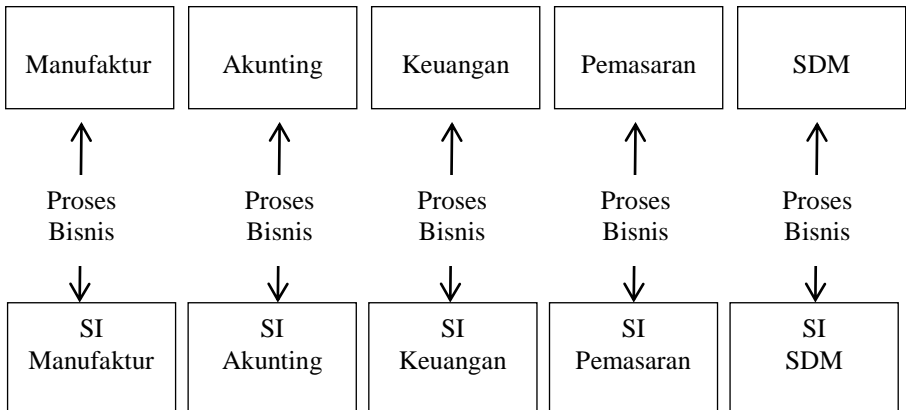
Gambar 6.3 menunjukkan secara umum bahwa suatu *enterprise* sangat membutuhkan keberadaan sistem informasi berbasis komputer yang mencakup berbagai proses bisnis yang dijalankan sehingga bisa mendukung kelancaran proses bisnisnya dan meningkatkan kinerja *enterprise* secara keseluruhan. Idealnya tidak ada satu proses bisnis pun yang tidak didukung oleh *software* sistem informasi.



Gambar 6.4 Sistem Informasi Konvensional

Sebagaimana yang ditunjukkan pada Gambar 6.4 bahwa konsep sistem informasi yang bersifat konvensional merupakan sistem yang terkotak-kotak satu sama lain, masing-masing sistem mandiri walaupun secara kebutuhan sistem informasi yang tersedia lengkap dan berada dalam satu lingkungan *enterprise*^[3]. Dengan demikian, hal ini menyebabkan tidak terjadinya *data sharing* di antara sistem yang ada, dan akan mengakibatkan berbagai masalah diantaranya sebagai berikut:

1. Duplikasi data di dalam sistem yang berbeda.
2. Bisa saja sistem informasi yang dibutuhkan lengkap tetapi tidak bisa diintegrasikan.
3. Manajer puncak akan kesulitan untuk mendapatkan informasi yang menyeluruh secara *enterprise*.
4. Pemeliharaan sistem lebih kompleks.



Gambar 6.5 Sistem Informasi *Enterprise*

Sedangkan dalam sistem informasi *enterprise*, seluruh sistem informasi yang diperlukan secara keseluruhan harus lengkap dan diintegrasikan menjadi satu kesatuan sistem yang saling terkait satu sama lain, baik dari aspek data, aplikasi maupun infrastrukturnya, sehingga semua permasalahan dalam sistem informasi konvensional bisa dipecahkan, seperti yang dijelaskan secara konsep pada Gambar 6.5.

Gambar 6.5 menjelaskan bahwa seluruh proses bisnis *enterprise* didukung oleh sistem informasi terkait yang merupakan satu kesatuan sistem yang saling terhubung satu sama lain (tidak terpisah seperti pada gambar 1.5). Dari sisi data, setiap sistem bisa menggunakan data yang dibuat oleh sistem yang lain, dan tidak boleh terjadi duplikasi data yang akan menghasilkan informasi yang tidak valid dan tidak berkualitas. Sedangkan, dari sisi aplikasi setiap sistem memiliki aplikasi yang berbeda, tetapi bisa memiliki

komponen sistem yang dipakai bersama, untuk memudahkan proses pemeliharaan sistem secara keseluruhan. Dengan kemajuan teknologi *software* saat ini, keseluruhan sistem yang diperlukan tidak harus berada dalam suatu *server* yang sama secara fisik, tetapi sistem tersebar di *server* yang berbeda dengan tetap terjaga menjadi suatu sistem informasi *enterprise* yang terpadu.

Enterprise Resource Planning (ERP) merupakan salah satu contoh kongkrit implementasi dari *integrated enterprise information system*. ERP saat ini lebih merupakan suatu paket aplikasi sistem informasi *enterprise* yang mendukung keseluruhan proses bisnis *enterprise* terutama *enterprise* dengan kategori manufaktur. Berbeda dengan jenis *software* lainnya, ERP merupakan paket *software* yang bisa diubah sesuai dengan kebutuhan masing-masing *enterprise* baik ERP yang bersifat *open source* maupun yang bersifat *proprietary*. Karena proses bisnis yang didukung oleh ERP bersifat umum, maka banyak *enterprise* yang lebih menyesuaikan proses bisnisnya dengan aplikasi yang tersedia di ERP.

ERP saat ini tidak hanya untuk kepentingan *enterprise* yang berbentuk manufaktur, tetapi sudah berkembang menjadi beberapa jenis ERP, yang bisa mendukung jenis *enterprise* lainnya seperti rumah sakit, perguruan tinggi, sekolah, lembaga pemerintah, dll. ERP *open source* seperti Compiere, OpenBravo, OpenERP, dll. Sedangkan ERP *proprietary* seperti SAP, Web Dynamic, dll.

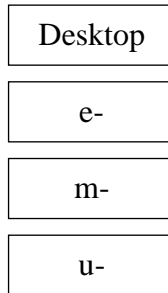
Untuk menentukan ERP mana yang lebih baik digunakan, tentunya harus dilakukan kajian berupa perencanaan arsitektur *enterprise* terlebih dahulu untuk mengetahui karakter proses bisnisnya, keperluan data, aplikasi dan infrastruktur teknologi. Dengan demikian,

berdasarkan kajian tersebut bisa diambil keputusan paket ERP mana yang sebaiknya diimplementasikan, disesuaikan dengan ketersediaan dana dan kebutuhan fungsinya.

ERP yang dikhususkan untuk mendukung kelancaran proses bisnis di lingkungan pemerintahan lebih dikenal dengan istilah e-Governance. Dalam rangka mewujudkan tata-kelola pemerintahan yang lebih bersih dan transparan, penerapan e-Governance mulai dijalankan di berbagai level pemerintahan, walaupun belum mencakup keseluruhan proses bisnis. Sebagai contoh adalah Sistem Informasi Rumah Sakit (SIRS) berbasis *open source* yang disediakan oleh Kementerian Kesehatan untuk pengelolaan rumah sakit khususnya rumah sakit milik pemerintah.

6.5 Evolusi Sistem Informasi

Seiring dengan perkembangan teknologi informasi, istilah sistem informasi mengalami evolusi seperti pada Gambar 6.6 yaitu dari sistem yang berbasis desktop lalu menjadi segala sesuatu sistem pengolahan data berbasis jaringan komputer yang penamaannya dimulai dengan awalan “e”. Pada awalnya muncul istilah *email* sebagai bentuk digital *mail* menggantikan *mail* konvensional. Kemudian, berkembanglah berbagai jenis sistem informasi yang masing-masing sehingga muncul istilah-istilah seperti e-Learning, e-Library, e-Commerce, e-KTP dll. Dengan demikian, muncullah secara umumnya istilah e-Business yang mengacu kepada sistem informasi berbasis web supaya sistem bisa diakses dari mana saja dan kapan saja.



Gambar 6.6 Evolusi Sistem Informasi

Huruf “e” sendiri sebenarnya berasal dari kata Bahasa Inggris “electronic”. Secara harfiah berarti segala sesuatu sistem informasi yang menggunakan perangkat elektronik. Kenyataanya huruf “e” bermakna bukan hanya sekedar elektronik, tetapi lebih luas yaitu mengandung pengertian perangkat komputer yang terkoneksi melalui jaringan internet. Dengan demikian huruf “e” bermakna segala sesuatu sistem informasi yang memanfaatkan komputer dan jaringan internet dalam proses pengolahan datanya. Oleh karena itu, saat ini penamaan sebuah sistem informasi lebih populer menggunakan awalan “e” dari pada hanya sekedar nama sistem informasi.

Seiring dengan masyarakat *gadget* berupa *smartphone* atau *tablet* yang didukung dengan tersedianya berbagai *software open source*, sehingga memungkinkan akses ke sistem informasi melalui internet semakin bervariasi. Dengan demikian penggunaan sistem informasi bisa dilakukan dari mana saja dan kapan saja. Muncullah istilah *mobile computing*. Namun demikian dalam perkembangannya istilah sistem informasi dengan diawali huruf “m” sebagai pengganti dari “e” kurang populer.

Selain istilah *mobile* muncul juga istilah *ubiquitous* terkait perkembangan peralatan apa saja yang bisa dipakai dan tidak harus selalu berbentuk komputer (*wearable computing*) pada umumnya, tetapi bisa berbentuk jam tangan, kacamata, dll. Sehingga muncul istilah aplikasi yang bisa akses ke suatu sistem informasi melalui perangkat seperti ini yang disebut dengan istilah yang dimulai dengan huruf “u”.

BAB 7

REQUIREMENT ENGINEERING

7.1 Fungsional dan Non-Fungsional

Kebutuhan *software* secara umum dapat diklasifikasikan menjadi dua kelompok yaitu sebagai berikut:

7. Kebutuhan fungsional yang menyatakan layanan apa saja yang seharusnya sebuah *software* sediakan termasuk input, output, dan proses yang diperlukan. Dengan kata lain, apa yang bisa *software* perbuat terhadap penggunaanya.
8. Kebutuhan non-fungsional, biasa mencakup *software* secara keseluruhan seperti batasan waktu/biaya, standarisasi (seperti *tool* pengembangan yang digunakan, bahasa pemrograman, lisensi, dll)

Kebutuhan terhadap *software* terbagi menjadi dua bagian yaitu kebutuhan pengguna dan kebutuhan sistem.

7.2 Spesifikasi Kebutuhan

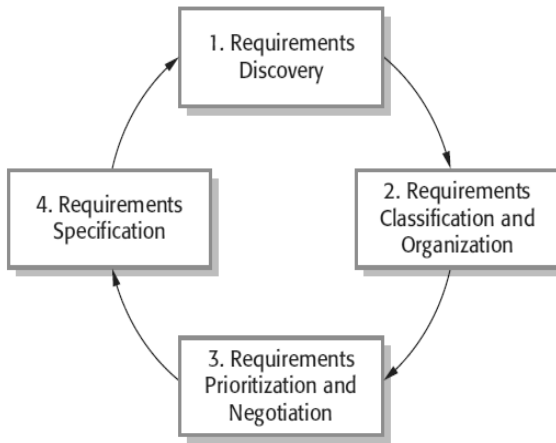
Proses pembuatan dokumen atau penulisan kebutuhan *software* yang disepakati bersama antara pengguna dan pengembang. Penulisannya harus sejelas mungkin, tidak ada kesamaran, mudah dimengerti oleh berbagai pihak terkait, lengkap dan konsisten.

Dokumentasi kebutuhan selain berbentuk narasi berupa penjelasan dengan menggunakan bahasa alami, gunakan format standar penulisan yang sederhana, bila perlu bisa menggunakan *text highlighting* seperti huruf tebal, tulisan miring, dan warna.

Penulisan dokumentasi kebutuhan bisa juga menggunakan notasi-notasi grafis untuk membantu visualisasi kebutuhan software supaya mudah dipahami berbagai pihak, seperti menggunakan *Unified Modeling Language* (UML). Penggunaan notasi tidak dibatasi hanya UML, tetapi bisa menggunakan alat bantu (*tool*) apa saja sepanjang membantu pemahaman tentang kebutuhan *software*.

7.3 Elisitasi & Analisis Kebutuhan

Kebutuhan terhadap software dapat diproses melalui tahapan seperti pada Gambar 7.1, dimana prosesnya berjalan secara iteratif dimulai dari penemuan kebutuhan dan diakhiri dengan dokumentasi kebutuhan. Satu iterasi ke iterasi berikutnya merupakan penyempurnaan terhadap kebutuhan *software*.



Gambar 7.1 Proses Elisitasi & Analisis Kebutuhan

7.4 Validasi Kebutuhan

Ada beberapa teknik untuk melakukan validasi kebutuhan yaitu sebagai berikut:

1. *Requirement reviews*

Kebutuhan dianalisis secara sistematis oleh suatu tim review untuk mengecek kesalahan dan kemungkinan tidak konsisten.

2. *Prototyping*

Dibuatkan contoh aplikasi yang bisa ditunjukkan kepada pengguna untuk mencoba dan menggali kebutuhan pengguna.

3. *Test-case generation*

Kebutuhan software harus bisa diuji. Jika ujicoba sulit dilakukan biasanya berarti bahwa kebutuhan

tersebut sulit dipenuhi sehingga perlu dipertimbangkan kembali.

7.5 Manajemen Kebutuhan

Semakin kompleks sebuah software, umumnya kebutuhannya juga semakin rumit untuk dirumuskan, sehingga memerlukan pengelolaan yang sistematis supaya kebutuhan software bisa ditetapkan secara benar dan lengkap. Selama proses pengembangan software, seringkali pengguna mengungkapkan kebutuhan berubah-ubah yang bisa berakibat pada penyelesaian software.

Ada tiga prinsip yang bisa dilakukan dalam pengelolaan perubahan kebutuhan, yaitu sebagai berikut:

1. *Problem analysis and change specification*
Menganalisis masalah yang menyebabkan perubahan kebutuhan dan spek *software*.
2. *Change analysis and costing*
Menganalisis perubahan kebutuhan terhadap biaya pengembangan sejauh mana pengaruhnya, apakah masih bisa ada dalam cakupan atau tidak.
3. *Change implementation*
Perubahan seperlunya dokumen kebutuhan software, termasuk perubahan rancangan dan implementasi *software*.

BAB 8

PERANCANGAN ARSITEKTUR

8.1 *Architectural Views*

Perancangan *software* fokus kepada bagaimana membuat suatu *software* sebagai penjabaran dari hasil proses analisis. Perancangan menggunakan notasi yang berbeda dengan yang digunakan pada proses analisis. UML menyediakan berbagai diagram yang bisa digunakan dalam proses perancangan maupun analisis sistem.

Pandangan tentang rancangan arsitektur *software* terdiri dari empat tipe yaitu:

1. *Logical view*

View ini menunjukkan abstraksi suatu sistem, terkait langsung dengan kebutuhan sistem

2. *Process view*

View ini berguna untuk membuat keputusan tentang karakteristik fungsional sistem seperti kinerja sistem.

3. *Development view*

View yang menunjukkan komposisi *software* yang akan diimplementasikan oleh *developer*. *View* ini berguna bagi manajer *software* , *database administrator*, dan *programmer*

4. *Physical view*

View yang menunjukkan bagaimana sistem diterapkan secara riil baik secara hardware maupun *software*.

Dalam *framework Zachman* istilah *view* dikenal dengan *perspective*, yaitu *scope/planner/contextual*, *business model/owner/conceptual*, *system model/designer/logical*, dan *technology model/builder/physical*.

8.2 Architectural Pattern

Perancangan *software* sebaiknya menggunakan salah satu pola arsitektur yang umum digunakan dalam pengembangan *software*. Pola yang sering digunakan adalah *Model-View-Controller* atau MVC. Bahasa pemrograman umumnya sudah mendukung pola MVC ini seperti Struts untuk Java, CodeIgniter untuk PHP, dll.

Pola MVC membagi *software* menjadi tiga bagian penting yaitu

1. *Model*

Bagian pengelolaan data yang perlu diproses oleh aplikasi, terutama bagian yang berhubungan dengan data yang disimpan dalam server *database*.

2. *View*

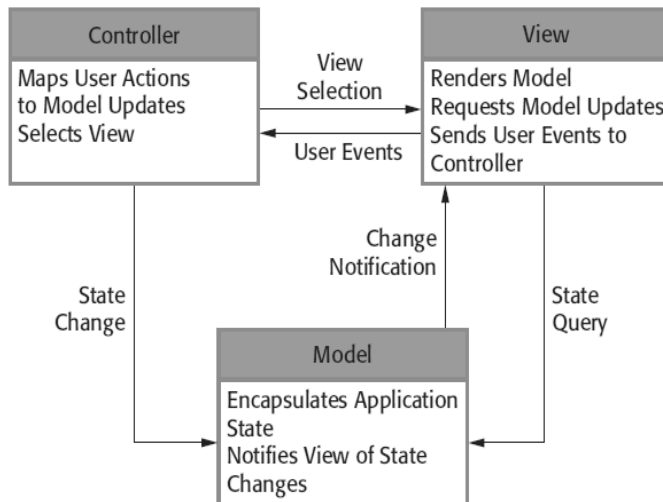
Bagian yang terkait langsung dengan pengguna, biasa berupa tampilan antarmuka yang menghubungkan antara penggunaan dan sistem.

3. *Controller*

Bagian penghubung antara bagian *View* dan *Model*. Bagian ini juga yang mengimplementasikan logika pemrograman yang diperlukan oleh suatu aplikasi.

Sedangkan arsitektur sistem menggunakan konsep *Client-Server* dimana *Client* di sisi pengguna biasayan

menggunakan *web browser*, sedangkan Server sebagai penyedia layanan termasuk data. *Client* dan Server terkoneksi melalui jaringan komputer baik internet maupun intranet.



Gambar 8.1 MVC

8.3 Arsitektur Aplikasi

Aplikasi sebuah sistem informasi dibuat dengan tujuan untuk memenuhi kebutuhan bisnis organisasi. Arsitektur aplikasi bisa dengan membuat aplikasi yang benar-benar dikembangkan dari nol. Sebagai alternatif, bisa juga dengan memanfaatkan *software* yang sudah ada seperti berbagai *software* ERP baik yang bersifat *proprietary* maupun *open source*.

Sebuah *software* sebagian besar terdiri dari aplikasi yang diimplementasikan dengan menggunakan bahasa pemrograman tertentu.

Arsitektur aplikasi merupakan gabungan dokumen dari proses dan dokumen terstruktur berupa model, pola proses bisnis, dll) untuk merancang berbagai aplikasi bisnis yang dibutuhkan. Dengan mendefinisikan kumpulan aplikasi yang dibutuhkan, hal ini akan memudahkan cepatnya pengembangan dan penyampaian aplikasi dengan cara yang sistematis dan teratur.

8.4 Arsitektur Data

Bagian dari *software* yang terkait dengan pengelolaan data. Biasanya menggunakan *software* khusus *Database Management System* (DBMS) supaya pengelolaan data bisa dilakukan secara efektif dan efisien.

Arsitektur ini merupakan struktur untuk mendokumentasikan secara rinci data apa saja yang benar-benar diperlukan untuk memperlancar jalannya proses bisnis suatu organisasi. Disamping memperjelas hubungan antar data yang mendukung proses bisnis yang dijalankan melalui *software* sistem informasi.

Arsitektur data memperjelas hubungan bisnis dan data yang diperlukan oleh organisasi. Dalam sebuah *software* sistem informasi arsitektur ini merupakan bagian terpenting yang merupakan sumber informasi yang dibutuhkan organisasi dalam menjalankan bisnisnya.

8.5 Rancangan Tampilan

Dalam *software* sistem informasi tampilan suatu *software* akan sangat berpengaruh pada kelangsungan *software* tersebut (*life time*) dalam jangka waktu yang lama. Oleh karena itu, perancangan tampilan harus dipertimbangkan dengan memperhatikan apa yang diinginkan oleh calon penggunaanya.

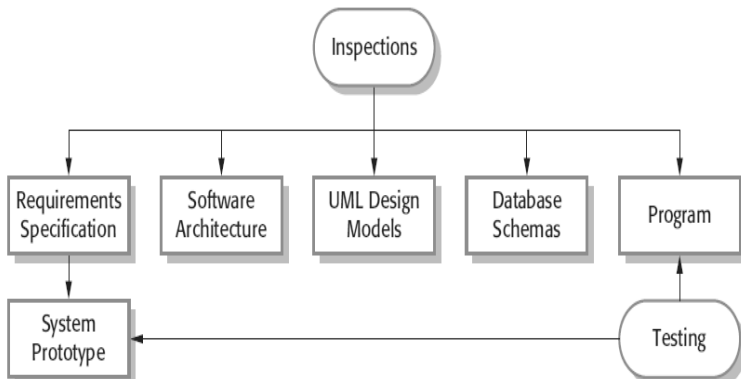
Perancangan tampilan sebuah *software* bisa dilakukan dengan menggunakan metode tertentu diantaranya dengan *Kansei Engineering* untuk mendapatkan tampilan *interface* seperti bagaimana yang diinginkan oleh pengguna dan diterjemahkan ke dalam berbagai elemen rancangan tampilan *software*.

BAB 9

UJICoba

9.1 Inspeksi

Pengembangan *software* tidak akan lepas dari kejadian yang tidak diinginkan walaupun sudah dilakukan secara hati-hati. Oleh karenanya, perlu dilakukan ujicoba dahulu sebelum *software* benar-benar digunakan oleh pengguna dalam pekerjaannya. Tidak ada satupun *software* yang dibuat oleh siapapun yang secara otomatis selesai tanpa adanya kesalahan apapun. Semakin kompleks dan semakin besar cakupan suatu *software*, maka konsekwensinya akan semakin banyak hal-hal yang harus dilakukan dalam ujicoba. Hal-hal yang harus diujicoba seperti yang dijelaskan pada Gambar 9.1.



Gambar 9.1 Inspeksi dan Ujicoba

Ujicoba dilakukan untuk memastikan bahwa *software* berjalan sebagaimana mestinya tanpa terjadi kesalahan apapun secara sistem, sehingga bisa aman untuk diimplementasikan.

Tujuan ujicoba terdiri dari:

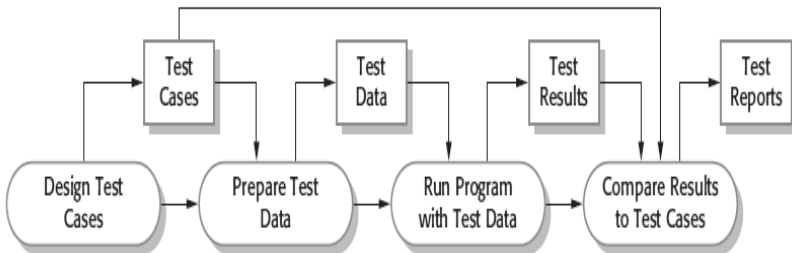
1. Menunjukkan bahwa *developer* dan pengguna bahwa *software* memenuhi kebutuhan yang ditetapkan bersama.
2. Menemukan situasi dimana ada cara kerja *software* yang salah, tidak diinginkan, atau tidak sesuai spek.

9.2 Ujicoba Pengembangan

Ujicoba dalam pengembangan dilakukan dalam beberapa bagian sistem. Sedangkan tahapan ujicoba dilakukan seperti pada Gambar 9.2.

1. Ujicoba unit
Pengujian bagian terkecil dari suatu software yang dikembangkan.
2. Ujicoba kasus
Pengujian berdasarkan kasus tertentu yang sangat diperkirakan sangat potensial terjadinya kesalahan.
3. Ujicoba komponen
Pengujian komponen software yang merupakan integrasi dari bagian-bagian di dalamnya yang lebih kecil.
4. Ujicoba sistem

Pengujian keseluruhan dari suatu sistem software dan pengintegrasian seluruh komponen sistem sehingga dapat dipastikan software berjalan tanpa masalah berarti.



Gambar 9.2 Proses Ujicoba *Software*

9.3 User Testing

Ada tiga tipe untuk melakukan ujicoba oleh pengguna, yaitu sebagai berikut:

1. *Alpha*

Pengguna bekerjasama dengan tim *developer* melakukan ujicoba *software* di tempat *developer*.

2. *Beta*

Pengguna mencoba *software* sebagai masa percobaan, dan untuk mencoba menemukan keganjalan lainnya bersama-sama tim *developer*.

3. *Acceptance*

Pengguna menjalankan *software* dengan sebenarnya untuk memastikan bahwa *software*

sudah tidak ada masalah dan dapat digunakan sebagaimana mestinya.

Dalam tahapan ujicoba *software* ada dua proses penting yaitu verifikasi dan validasi atau sering dikenal dengan istilah V&V yaitu:

1. Verifikasi

Apakah *software* telah dibuat dengan benar? Mencakup proses pengembangan yang dilakukan telah melalui tahapan yang sesuai dengan standar.

2. Validasi

Apakah telah dibuat *software* yang benar? Mencakup *software* yang dibuat telah sesuai dengan apa yang dibutuhkan.

Ada dua metode ujicoba validasi *software* yang biasa dilakukan yaitu:

1. *Black-Box*

Pengujian *software* dengan memandangnya sebagai sebuah kotak hitam yang tidak perlu untuk melihat isinya, dengan demikian cukup melakukan ujicoba dari bagian luar *software*.

Beberapa teknik yang digunakan dalam ujicoba ini adalah:

- *Decision (branch) Coverage*
- *Condition Coverage*
- *Path Analysis*
- *Execution Time*

- *Algorithm Analysis*

2. *White-Box*

Pengujian *software* dengan memandang bagian dalam *software* yaitu *source code*, sehingga pengujian dilakukan lebih rinci terkait dengan keseluruhan bagian dari *source code*.

Beberapa teknik yang digunakan dalam ujicoba ini adalah:

- *Equivalen Partitioning*
- *Boundry Value Analysis*
- *Cause Effect Graph*
- *Random Data Selection*
- *Feature Test*

BAB 10

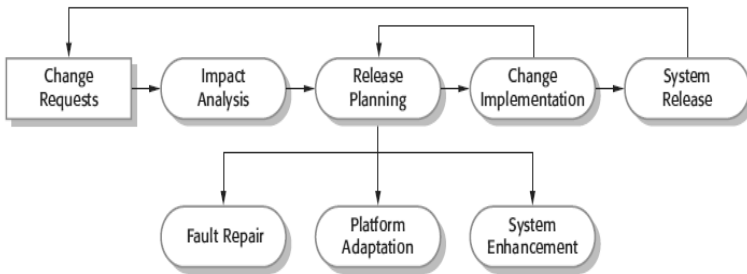
EVOLUSI *SOFTWARE*

10.1 Proses Evolusi

Pengembangan software tidak selesai ketika suatu software selesai dibuat dan dipakai oleh pengguna, tetapi masih terus berlanjut selaman *software* tersebut digunakan. Proses bisnis akan terjadi perubahan-perubahan yang menyebabkan *software* harus dilakukan penyesuaian. Dengan demikian, software akan terus mengalami evolusi selama *software* tersebut masih diperlukan dan digunakan oleh penggunanya.

Dalam pengembangan software sistem informasi, evolusi merupakan hal yang penting karena software senantiasa harus bisa mendukung proses bisnis, dimana apabila terjadi perubahan maka sudah selayaknya software pendukungnyapun harus disesuaikan. Apalagi yang mendukung sistem yang kompleks yang memerlukan biaya yang tidak sedikit.

Evolusi *software* dilakukan seperti pada Gambar 10.1. Proses evolusi dimulai dari datangnya permintaan dari pengguna tentang perubahan software karena ada perubahan proses bisnis. Selanjutnya developer harus menganalisis permintaan tersebut dan dihitung kemungkinannya, biaya, maupun waktu yang diperlukan. Sampai akhirnya *developer* melepas software yang telah dimodifikasi kepada pengguna.



Gambar 10.1 Proses Evolusi

10.2 Pemeliharaan *Software*

Pemeliharaan software adalah proses yang umum dilakukan setelah diserahkan kepada pengguna. Biasanya pemeliharaan dilakukan terhadap adanya kesalahan mulai dari error coding, peningkatan kinerja, sampai dengan penambahan kebutuhan. Perubahan dilakukan dengan merubah software yang ada, dan bisa juga menambahkan komponen software yang baru bila diperlukan.

Ada tiga jenis pemeliharaan software, sebagai berikut:

1. *Fault repairs*

Kesalahan tentang *coding* relatif lebih mudah dibandingkan dengan kesalahan pada tahap perancangan, karena kesalahan perancangan berakibat terhadap kesalahan coding. Kesalahan pada waktu analisis kebutuhan merupakan kesalahan yang paling fatal.

2. *Environmental adaption*

Pemeliharaan jenis ini dibutuhkan ketika terjadi perubahan terkait lingkungan sistem software seperti hardware, sistem operasi, atau software pendukung lainnya. Software perlu disesuaikan dengan perubahan lingkungan pendukungnya.

3. *Functionality addition*

Pemeliharaan ini dilakukan pada saat kebutuhan sistemnya berubah karena terjadi perubahan organisasi atau perubahan proses bisnis. Pemeliharaan jenis ini skalanya biasanya lebih besar dibandingkan dengan jenis lainnya.

Dari ketiga jenis pemeliharaan tersebut yang paling sering dilakukan adalah *functionality addition* sebanyak 65%, sedangkan *fault repair* 17% dan *environmental adaption* 18%.

BAB 11

EXTREME PROGRAMMING

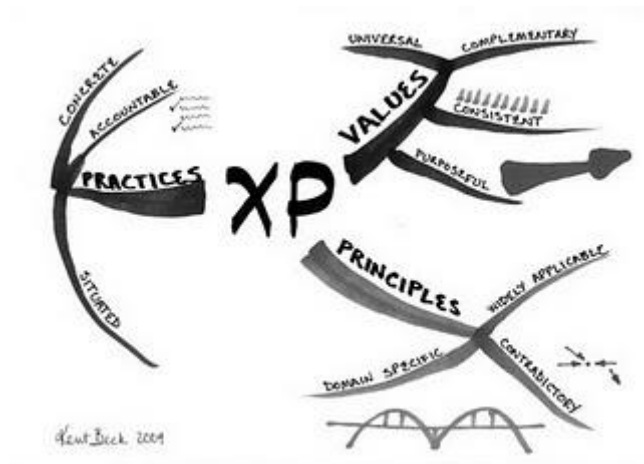
11.1 Sejarah XP

Proyek pengembangan *software* yang pertama kali menerapkan *Extreme Programming* (XP) adalah *C3 (Chrysler Comprehensive Compensation) Project* dari Chrysler.

Proyek ini adalah sebuah proyek penggajian 10.000 karyawan Chrysler, terdiri dari kira-kira 2000 class dan 30.000 method. Proyek yang dimulai pada pertengahan dekade 90-an ini terancam gagal karena rumitnya sistem yang dibangun dan kegagalan pada saat *testing*. Chrysler kemudian mempekerjakan Kent Beck, seorang pakar *software engineering*. Di kemudian hari Kent Beck menjadi terkenal sebagai pencetus awal dari XP. Kent Beck bersama rekannya Ron Jeffries dengan kewenangan yang diberikan oleh Chrysler melakukan berbagai perubahan di *C3 Project* untuk membuatnya lebih efisien, adaptif, dan fleksibel. Hal yang paling penting pada waktu itu adalah harus mampu memenuhi permintaan utama dari Chrysler, untuk melakukan *launching software* yang diinginkan oleh Chrysler dalam waktu tidak lebih dari dua tahun sejak saat Kent Beck dikontrak.

Kent Beck dan Ron Jeffries pada akhirnya berhasil menyelesaikan target Chrysler dengan menerapkan berbagai metode dalam proses pengembangan *software* tersebut. Kumpulan metode inilah yang kemudian dikenal sebagai model atau pendekatan XP dalam pengembangan *software*, seperti pada Gambar 11.1. Begitu sederhananya metode-

metode tersebut sehingga bagi orang yang belum menerapkan XP, sehingga seolah-olah hanya merupakan sekumpulan dari ide-ide lama yang dinilai terlalu sederhana dan tidak akan memberikan efek apapun pada sebuah proyek pengembangan *software*.



Gambar 11.1 *Extreme Programming*

Kent Beck sendiri menyadari bahwa XP tidak selalu cocok untuk setiap proyek pengembangan *software*. Kelebihan XP adalah sangat cocok digunakan untuk proyek-proyek *software* yang memiliki sifat *dynamic requirements*, artinya bahwa XP sangat mendukung pengembangan *software* yang memerlukan adaptasi cepat dalam mengatasi perubahan-perubahan yang terjadi selama proses pengembangan *software*. Disamping itu, XP juga cocok untuk proyek dengan jumlah anggota tim *developer* tidak terlalu banyak (sekitar 10-20 orang) dan berada pada lokasi yang sama.

11.2 Tujuan XP

Tujuan utama XP sebenarnya adalah menurunkan biaya dari adanya perubahan *software*. Dalam metodologi pengembangan sistem tradisional, kebutuhan sistem ditentukan pada tahap awal pengembangan proyek dan bersifat tetap (tidak diperkenankan terjadi perubahan). Dengan demikian, seandainya harus ada perubahan, maka biaya terhadap adanya perubahan kebutuhan yang terjadi pada tahap selanjutnya akan menjadi mahal.

XP diarahkan untuk menurunkan biaya dari adanya perubahan dengan memperkenalkan nilai-nilai basis dasar, prinsip dan praktis. Dengan menerapkan XP, pengembangan suatu sistem haruslah lebih fleksibel terhadap perubahan.

Penerapan XP dapat dilakukan dalam kondisi pengembangan *software* sebagai berikut :

1. Keperluan terhadap *software* mengalami perubahan dengan cepat
2. Resiko tinggi dan ada proyek dengan tantangan yang baru
3. Jumlah anggota Tim *Programmer* hanya sekitar 2-10 orang
4. Agar dapat mengotomatiskan ujiboca *software* yang akan direlease
5. Mengoptimalkan keterlibatan pelanggan dalam pengembangan *software* secara langsung

11.3 Variabel XP

Variabel yang berlaku dalam pengembangan *software* berbasis XP ada empat buah, antara lain sebagai berikut:

1. *Cost* (biaya).

Dengan meningkatkan biaya, maka dapat menciptakan program yang lebih baik. Sebaliknya pengurangan biaya untuk proyek tidak akan menyelesaikan masalah klien. Tetapi, biaya yang tiak terbatas juga akan menimbulkan kerusakan.

2. *Time* (waktu)

Dengan meningkatkan waktu maka mampu membuat *software* yang berkualitas dan dengan *feature-feature* yang lebih banyak. Akan tetapi waktu yang berlebihan tidak baik, karena dapat merusak terhadap diri sendiri. Waktu yang sedikit juga tidak baik karena kualitas yang dihasilkan akan jauh dari yang diharapkan.

3. *Quality* (mutu)

Mutu merupakan suatu variabel pengendali yang sangat “menakutkan” karena merupakan suatu hal wajar semestinya diberikan kepada klien.

4. *Scope* (Area)

Scope merupakan varibel primer. Jika dilakukan pengurangan *scope*, maka bisa berpengaruh pada pengurangan biaya, sekaligus bisa pula meningkatkan kualitas.

11.4 Kunci utama XP

Kent Beck sebagai penggagas awal XP mendefinisikan lima kunci utama dari XP, yaitu :

1. *Communication* (Komunikasi)

XP memfokuskan pada hubungan komunikasi yang

baik antar anggota tim. Para anggota tim harus membangun saling pengertian, mereka juga wajib saling berbagi pengetahuan dan keterampilan dalam mengembangkan perangkat lunak. Ego dari para *programmer* yang biasanya cukup tinggi harus ditekan dan mereka harus membuka diri untuk bekerjasama dengan programmer lain dalam menuliskan kode program. Komunikasi menjadi hal yang sangat menentukan dalam sebuah tim pengembangan *software*.

Apabila komunikasi antar pihak-pihak yang berkepentingan dalam pengembangan *software* gagal diwujudkan, maka masalah-masalah yang tidak diharapkan bisa terjadi seperti: proses bisnis yang dibuat tidak sesuai, anggota tim menghadapi masalah yang tidak dapat diselesaikan, atasan mendapatkan laporan kejutan sehari sebelum *deadline*, dsb.

2. *Courage* (Keberanian)

Para anggota tim dan penanggungjawab pengembangan perangkat lunak harus selalu memiliki keyakinan dan integritas dalam melakukan tugasnya. Integritas ini harus selalu dijaga bahkan dalam kondisi adanya tekanan dari situasi sekitar. Untuk dapat melakukan sesuatu dengan penuh integritas, terlebih dahulu para anggota tim memiliki rasa saling percaya.

Rasa saling percaya inilah yang coba dibangun dan ditanamkan oleh XP pada berbagai aspeknya. *Courage* diperlukan untuk mengatakan bahwa target *deadline* yang diinginkan customer tidak mungkin tercapai, untuk mengambil keputusan saat code yang sudah dibuat ternyata harus dibuang karena kesalahan informasi.

3. *Simplicity* (Kesederhanaan)

Lakukan semua dengan sederhana. Hal tersebut adalah salah satu nilai dasar dari XP. Gunakan metode yang pendek dan simpel, jangan terlalu rumit dalam membuat desain, hilangkan fitur yang tidak ada gunanya, dan berbagai proses penyederhanaan lain akan selalu menjadi nilai utama dari setiap aspek XP. Kesederhanaan mengacu pada desain sistem yang akan dibuat. Berlawanan dengan disiplin *software development* lainnya, XP menganjurkan desain yang berevolusi sepanjang proses pengembangan. Seringkali *programmer* berpikir terlalu kompleks dalam menyelesaikan suatu permasalahan.

Pada saat yang lain mereka berpikir bagaimana supaya sistem menjadi fleksibel untuk masa depan. Dalam mengambil keputusan demikian biasanya hanya berdasar pada spekulasi saja, di mana pada akhirnya perkiraan tersebut meleset, sistem tidak pernah diuntungkan dengan desain fleksibel yang super kompleks tersebut. Satu prinsip yang berkaitan dengan ini adalah “*You aren’t gonna need it*“, di mana sangat dianjurkan agar dalam membangun sesuatu terlebih dahulu harus memastikan bahwa hal ini memang dibutuhkan.

4. *Feedback* (Umpan Balik)

Berikan selalu *feedback* kepada sesama anggota tim maupun pihak-pihak lain yang terlibat dalam pengembangan perangkat lunak. Utarakan selalu pikiran anda dan diskusikan kesalahan-kesalahan yang muncul selama proses pengembangan. Dengarkan selalu pendapat rekan yang lain.

Feedback digunakan untuk mengetahui bagian mana yang salah atau bagian mana yang bisa ditingkatkan lagi dari *software* yang dikembangkan. *Feedback* diperlukan

untuk mengetahui kemajuan dari proses dan kualitas dari *software* yang dibangun. Informasi yang diperoleh dari *feedback* harus dikumpulkan setiap interval waktu yang singkat secara konsisten. Ini dimaksudkan agar hal-hal yang menjadi masalah dalam proses pengembangan dapat diketahui sedini mungkin dan klien dapat membuat keputusan berdasarkan informasi tersebut.

5. *Quality Work* (Kualitas Kerja)

Semua nilai di atas berujung pada sebuah kondisi dimana pekerjaan dilakukan dengan berkualitas. Dengan proses yang berkualitas maka akan muncul pula implikasi perangkat lunak yang berkualitas sebagai hasil akhir.

11.6 Prinsip Dasar XP

Dalam pengembangan *software*, penerapan XP memiliki lima prinsip utama, yaitu:

1. *Rapid Feedback*

Dari sisi ilmu psikologi, waktu antara sebuah aksi dengan *feedback*-nya sangat penting untuk dipelajari. Dalam sebuah proyek XP, *developer* bisa memperoleh *feedback* sesegera mungkin, menginterpretasikannya, lalu mengambil inti sarinya dan meletakkannya ke dalam sistem *software*. Frekuensi *feedback* dari klien terhitung harian, bukan bulanan, dan *feedback* dari *developer* terhitung menitan, bukan mingguan.

2. *Assume Simplicity*

Hanya mendesain untuk masalah saat ini dan menghemat waktu 98% dari masalah tersebut dan hanya

menekuni sekitar 2% untuk bagian yang sulit. XP berencana untuk masa depan sehingga desainnya dapat di-*reuse*, lakukan pekerjaan yang penting, dan percayalah bahwa untuk kekompleksitasan dapat ditambahkan kemudian.

3. *Incremental Change*

Pembagian masalah menjadi beberapa bagian dengan sedikit kecil perubahan saja. Dengan demikian, perubahan-perubahan yang terjadi pada pengembangan *software* dengan XP melalui tahapan-tahapan kecil dan dalam waktu yang singkat.

4. *Embracing Change*

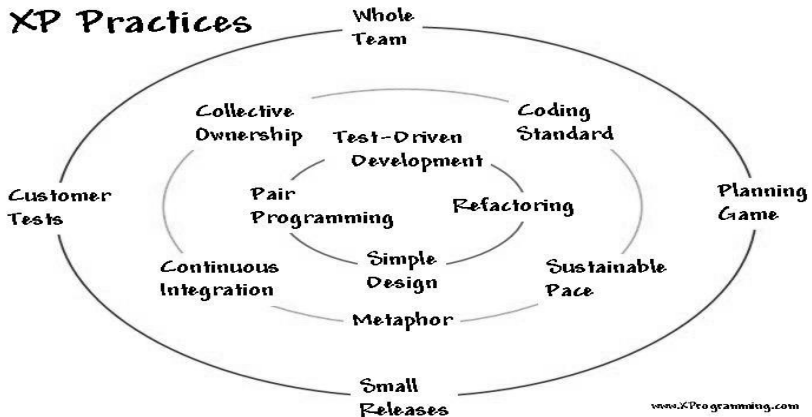
Senantiasa berusaha mencari dan menyediakan terlebih dahulu sebanyak mungkin alternatif solusi terbaik untuk menyelesaikan masalah-masalah penting yang dihadapi dengan tepat.

5. *Quality Work*

Setiap anggota dalam *developer* menginginkan pekerjaan yang baik, layak dan berkualitas. Kualitas yang dimaksud adalah kualitas yang ekstrim. Karena tanpa adanya kualitas tertentu, maka tidak akan melakukan pekerjaan tersebut, akhirnya hasil yang diinginkan tidak akan sempurna, dan proyek pengembangan *software* akan berantakan.

11.7 Aspek Pengembangan XP

Aspek dasar XP terdiri dari beberapa praktek yang diterapkan oleh Kent Beck dan Ron Jeffries pada *C3 Project* adalah sebagaimana yang ditunjukkan Gambar 11.2.



Gambar 11.1 Aspek Pengembangan XP

1. *The Planning Game*

Pendekatan XP dalam perencanaan sangat mirip dengan metode yang diterapkan pada *RAD (Rapid Application Development)*. Proses pendek dan cepat, mengutamakan aspek teknik, memisahkan unsur bisnis dengan unsur teknis dan pertemuan intensif antara klien dengan *developer*. Pada XP, proses ini menggunakan terminologi “game” karena Beck menyarankan untuk menggunakan teknik *score card* dalam menentukan *requirements*. Semakin sulit aspek teknis yang dibutuhkan semakin tinggi pula skor pada kartu rencana tersebut.

2. *Small Releases*

Setiap *release* dilakukan dalam lingkup sekecil mungkin pada XP. Setiap *developer* menyelesaikan sebuah

unit atau bagian dari perangkat lunak maka hasil tersebut harus segera dipresentasikan dan didiskusikan dengan klien. Jika memungkinkan untuk menerapkan unit tersebut pada perusahaan, hal itu juga dapat dilakukan sekaligus sebagai tes awal dari penerapan keseluruhan sistem. Kendati demikian hal ini tidak selalu perlu dilakukan karena harus dihitung terlebih dahulu sumberdaya yang dibutuhkan. Apakah lebih menguntungkan langsung melakukan tes terhadap unit tersebut atau melakukan tes setelah unit tersebut terintegrasi secara sempurna pada sistem.

3. *Metaphor*

Metaphor pada dasarnya sama dengan arsitektur perangkat lunak. Keduanya menggambarkan visi yang luas terhadap tujuan dari pengembangan perangkat lunak. Beck sendiri seperti para penandatangan Agile Manifesto lainnya bercita-cita menyederhanakan proses pengembangan perangkat lunak yang saat ini sudah dianggap terlalu rumit. Arsitektur yang saat ini banyak berisi diagram dan kode semacam UML dianggap terlalu rumit untuk dimengerti, terutama oleh klien. *Metaphor*, walaupun mirip dengan arsitektur lebih bersifat naratif dan deskriptif. Dengan demikian diharapkan komunikasi antara klien dengan developer akan berlangsung lebih baik dan lancar dengan penggunaan *metaphor*.

4. *Simple Design*

Sebagai salah seorang penandatangan Agile Manifesto, Beck adalah seorang yang tidak menyukai desain yang rumit dalam sebuah pengembangan perangkat lunak. Tidak heran jika dia memasukkan *Simple Design* sebagai salah satu

unsur XP. Pada XP desain dibuat dalam lingkup kecil dan sederhana. Tidak perlu melakukan antisipasi terhadap berbagai perubahan di kemudian hari. Dengan desain yang simpel apabila terjadi perubahan maka membuat desain baru untuk mengatasi perubahan tersebut dapat dengan mudah dilakukan dan resiko kegagalan desain dapat diperkecil.

5. *Refactoring*

Refactoring adalah salah satu aspek paling khas dari XP. *Refactoring* seperti didefinisikan oleh Martin Fowler adalah "Melakukan perubahan pada kode program dari perangkat lunak dengan tujuan meningkatkan kualitas dari struktur program tersebut tanpa mengubah cara program tersebut bekerja". *Refactoring* sendiri sangat sesuai untuk menjadi bagian XP karena *Refactoring* mengusung konsep penyederhanaan dari proses desain maupun struktur baris kode program. Dengan *Refactoring* tim pengembang dapat melakukan berbagai usaha untuk meningkatkan kualitas program tanpa kembali mengulang-ulang proses desain. Fowler adalah salah satu kolega dekat dari Kent Beck karena itu tidak mengherankan bahwa cara berpikir mereka terhadap proses pengembangan perangkat lunak sangat mirip satu dengan lainnya.

6. *Testing*

XP menganut paradigma berbeda dalam hal tes dengan model pengembangan perangkat lunak lainnya. Jika pada pengembangan perangkat lunak lainnya *testing* baru dikembangkan setelah perangkat lunak selesai menjalani proses coding maka pada XP tim pengembang harus membuat terlebih dahulu tes yang hendak dijalani oleh

perangkat lunak. Berbagai model tes yang mengantisipasi penerapan perangkat lunak pada sistem dikembangkan terlebih dahulu. Saat proses *coding* selesai dilakukan maka perangkat lunak diuji dengan model tes yang telah dibuat tersebut. Pengetesan akan jauh lebih baik apabila dilakukan pada setiap unit perangkat lunak dalam lingkup sekecil mungkin daripada menunggu sampai seluruh perangkat lunak selesai dibuat.

7. *Pair Programming*

Pair programming adalah melakukan proses menulis program dengan berpasangan. Dua orang *programmer* saling bekerjasama di komputer yang sama untuk menyelesaikan sebuah unit. Dengan melakukan ini maka keduanya selalu dapat berdiskusi dan saling melakukan koreksi apabila ada kesalahan dalam penulisan program. Aspek ini mungkin akan sulit dijalankan oleh para *programmer* yang memiliki ego tinggi dan sering tidak nyaman untuk berbagi komputer bersama rekannya.

8. *Collective Ownership*

Tidak ada satupun baris kode program yang hanya dipahami oleh satu orang programmer. XP menuntut para programmer untuk berbagi pengetahuan untuk tiap baris program bahkan beserta hak untuk mengubahnya. Dengan pemahaman yang sama terhadap keseluruhan program, ketergantungan pada *programmer* tertentu ataupun berbagai hambatan akibat perbedaan gaya menulis program dapat diperkecil. Pada level yang lebih tinggi bahkan dimungkinkan para *programmer* dapat bertukar unit yang dibangunnya.

9. *Coding Standards*

Pair programming dan *collective ownership* hanya akan dapat berjalan dengan baik apabila para *programmer* memiliki pemahaman yang sama terhadap penulisan kode program. Dengan adanya *coding standards* yang telah disepakati terlebih dahulu maka pemahaman terhadap program akan menjadi mudah untuk semua *programmer* dalam tim. Hal ini dapat diterapkan sebagai contoh pada penamaan variabel dan penggunaan tipe data yang sama untuk tiap elemen semua record atau array pada program.

10. *Continuous Integration*

Melakukan *build* setiap hari kerja menjadi sebuah model yang disukai oleh berbagai tim pengembang perangkat lunak. Hal ini terutama didorong oleh keberhasilan penerapan sistem ini oleh Microsoft dan telah sering dipublikasikan. Dengan melakukan *build* sesering mungkin berbagai kesalahan pada program dapat dideteksi dan diperbaiki secepat mungkin. Apabila banyak tim pengembang perangkat lunak meyakini bahwa *build* sekali sehari adalah minimum maka pada XP hal tersebut adalah maksimum. Pada XP tim disarankan untuk melakukan *build* sesering mungkin misalnya setiap 4 jam atau bahkan lebih cepat lagi.

11. *40-hours Week*

Kent Beck berpendapat bekerja 8 jam sehari dan 5 hari seminggu adalah maksimal untuk tiap *programmer*. Lebih dari itu *programmer* akan cenderung membuat berbagai

error pada baris-baris kode programnya karena kelelahan.

12. On-Site Customer

Sebuah pendekatan klasik, di mana XP menganjurkan bahwa ada anggota dari klien yang terlibat pada proses pengembangan perangkat lunak. Yang lebih penting lagi ia harus ada di tempat pemrograman dan turut serta dalam proses *build* dan *test* yang dilakukan. Apabila ada kesalahan dalam pengembangan diharapkan klien dapat segera memberikan masukan untuk koreksinya.

Pada kenyataannya, bisa saja para *developer* menerapkan *practices* yang berbeda Pada *practice Planning Game* sendiri memiliki tiga fase seperti pada Tabel 11.1, yaitu *exploration phase*, *commitment phase*, dan *steering phase*. *Planning game* ini adalah pertemuan antara dua pihak, yaitu dari pihak *developer* dan pihak bisnis, dibahas mulai membuat *story* oleh *on-site customer* sampai *re-estimate* oleh pihak *development*.

Tidak seperti pada metodologi lainnya, pada fase-fase XP tidak terdapat sebuah fase ataupun bagian dari satu fase yang melakukan dokumentasi formal selama proses pengembangan. Satu-satunya dokumentasi adalah penulisan *requirements* pada *index card* yang disebut dengan *stories*. *Stories* ini ditulis pada fase *exploration*, di mana satu *function* yang akan diimplementasikan akan ditulis dalam satu *index card*. Selanjutnya pada proses pengembangan apabila satu *function* pada *stories* telah berhasil diimplementasikan, *index card*-nya akan dibuang.

Tabel 11.1 Fase pada *planning game* (sebelum modifikasi)

NO	PHASE	BUSINESS	DEVELOPMENT
I	Exploration phase	<i>Write story</i>	-
		-	<i>Estimate story</i>
		<i>Split story</i>	-
II	Commitment phase	<i>Sort by</i>	-
		-	<i>Sort by risk</i>
		-	<i>Sort by velocity</i>
		<i>Choose</i>	-
III	Steering phase	<i>Iteration</i>	-
		-	<i>Recovery</i>
		<i>New story</i>	<i>Re-estimate</i>

Hal ini bisa menjadi kelemahan XP karena tanpa dokumentasi formal maka proses pengembangan ini akan kembali seperti proses yang tidak terpola dan primitif. Apabila ditarik kepada model CMM, maka proses yang tanpa dokumentasi formal ini akan masuk ke level paling bawah yaitu lebel *initial*. Namun jika terdapat dokumentasi formal yang cukup berat, diperkirakan metodologi ini tidak menjadi ringan lagi sehingga tidak masuk dalam kategori *agile*.

BAB 12

PEMODELAN UML

12.1 Pemodelan *Software*

Pemodelan sistem informasi adalah penggambaran *software* yang mendukung sistem informasi tertentu dalam bentuk pemetaan berdasarkan pada aturan tertentu. Suatu sistem informasi semakin kompleks maka akan semakin memerlukan pemodelan yang rinci sebelum *software* yang mendukungnya dibuat.

Tujuan utama dari pengembangan aplikasi dengan menggunakan notasi UML (*Unified Modeling Language*), adalah sebagai berikut:

1. Menyediakan bahasa pemodelan visual yang ekspresif, lengkap, mudah dimengerti oleh pihak-pihak yang terlibat, dan siap pakai untuk pengembangan *software*.
2. Menyediakan fasilitas komunikasi berbagai pihak yang terlibat dalam pengembangan *software*, terutama antar *developer* dan pengguna, untuk menghindari kesalahpahaman tentang *software*.
3. Mendukung spesifikasi *software* yang independen artinya tidak tergantung pada salah satu bahasa pemrograman dan metodologi pengembangan tertentu
4. Menyediakan basis formal yang standar untuk pemahaman bahasa pemodelan
5. Menyediakan model yang dapat memberikan gambaran bagaimana *software* akan berjalan
6. Mendorong pengembangan *software* yang menggunakan prinsip-prinsip orientasi objek

7. Mendukung konsep-konsep pengembangan *software* untuk level lebih tinggi seperti komponen, kolaborasi, *framework* dan *pattern*

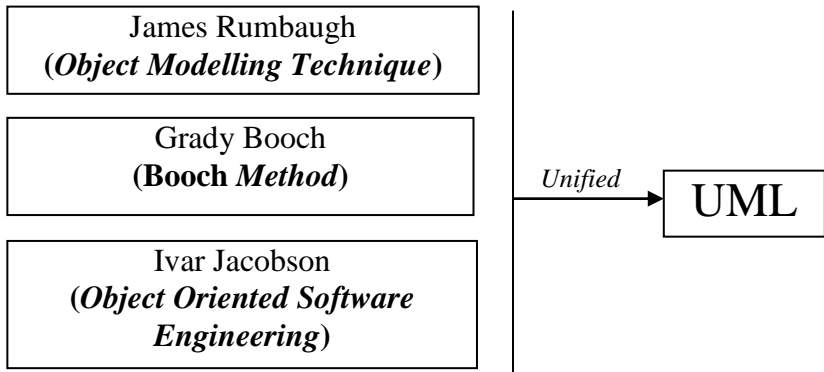
12.2 Unified Modeling Language

UML adalah notifikasi standar untuk menggambarkan keadaan suatu sistem *software*. UML pertama kali diperkenalkan pada tahun 1996 dengan versi 0.9, 0.91 oleh tiga serangkai seperti pada Gambar 11.1. UML dibuat berdasarkan pada notasi-notasi pemodelan versi-versi sebelumnya yaitu:

1. J. Rumbaugh sebagai pencetus OMT (*Object Modeling Technique*),
2. G. Booch sebagai pencetus Booch *Method*,
3. Jacobson sebagai pencetus OOSE.

Karena banyaknya bahasa pemodelan yang berkembang saat itu, maka dibuatlah bahasa pemodelan baru yang menggabungkan beberapa bahasa pemodelan sebelumnya. Pada tahun 1997 UML versi 1.1 diakui sebagai standar notifikasi OMG (*Object Management Group*).

Kegunaan atau manfaat suatu diagram pada pemodelan adalah untuk formalisasi ekspresi model objek secara koheren, presisi dan mudah dirumuskan. Pemodelan berorientasi objek memerlukan kaskas untuk mengekspresikan model. UML menyediakan sejumlah diagram untuk mempresentasikan suatu sistem secara visual dengan pemodelan berorientasi objek yang dilakukan, sehingga sangat membantu mempermudah pemahaman terhadap sistem dan pemeliharaan sistem.



Gambar 11.1 Tiga Pelopor UML

UML bukan metodologi, tetapi merupakan salah satu bahasa pemodelan yang memungkinkan *developer* melakukan permodelan secara visual, yaitu penekanan pada penggambaran, bukan didominasi oleh narasi. Permodelan visual membantu untuk menangkap struktur dan kelakuan dari objek, mempermudah penggambaran interaksi antara element dalam sistem, dan mempertahankan konsistensi antara disain dan implementasi dalam pemrograman.

Pengajuan UML oleh OMG memegang peranan penting dalam dunia *software*, terutama dalam pengembangan *software* berdasarkan orientasi objek. Spesifikasi UML secara lengkap dapat diakses melalui situs web <http://www.omg.org>.

12.3 Kategori Diagram

UML menyediakan banyak diagram yang diperlukan

dalam pengembangan software. Secara garis besar diagram yang ada di UML dikelompokkan menjadi dua jenis yaitu sebagai berikut:

1. Diagram Struktur

Diagram jenis ini untuk memvisualisasi, menspesifikasi, membangun, dan mendokumentasi *software* yang bersifat statik. Yang termasuk ke dalam diagram seperti ini adalah:

- a. *Class Diagram*
- b. *Object Diagram*
- c. *Component Diagram*
- d. *Deployment Diagram*

2. Diagram Perilaku (*Behaviour*)

Diagram jenis ini untuk memvisualisasi, menspesifikasi, membangun dan mendokumentasi *software* yang bersifat dinamis. Diagram perilaku terdiri dari:

- a. *Usecase Diagram*
- b. *Sequence Diagram*
- c. *Collaboration Diagram*
- d. *Statechart Diagram*
- e. *Activity Diagram*

12.4 Tips Pembuatan Diagram

Dalam membuat suatu diagram untuk melakukan pemodelan sistem, sebaiknya:

1. Lakukan pemilihan nama yang hati-hati untuk komponen-komponen diagram

2. Lakukan penomoran pada komponen-komponen paling utama
3. Hindari diagram yang terlalu kompleks
4. Perbaiki penampilan diagram sehingga
 - a. Benar secara teknis
 - b. Dapat diterima oleh pemakai
 - c. Tidak canggung untuk dipajang
5. Pastikan bahwa diagram-diagram itu konsisten
6. Diagram sebaiknya dimuat dalam satu halaman
 - a. Pemakai dapat melihat tanpa susah payah
 - b. Sistem yang dimodelkan tidak sangat kompleks. Bila sistem secara instrinsik kompleks, maka lebih dahulu lakukan dekomposisi di level lebih tinggi sehingga masing-masing diagram dapat dimuat di satu halaman
 - c. Sebaiknya menggunakan tool yang diotomasi yang telah dapat menguji kebenaran sintaks dan sebagian simantiks, serta hasilnya dapat digunakan untuk proses berikutnya dengan mudah

Daftar Pustaka

- Booch Grady, Maksimchuk R. A, et. al., 2007, Object-Oriented Analysis and Design with Applications Third Edition, Addison-Wesley
- Eeles Peter, Houston Kelli, Kozaczynski Wojtek, 2006, Builing J2EE Application with The Rational Unified Process, Addison-Wesley
- Laplante Philip A, 2007, What Engineer Should Know About Software Engineering, CRC Press
- Munawar, 2005, Pemodelan Visual, Penerbit Graha Ilmu
- Pressman Roger S., 2015, Software Engineering (A Practioner's approach) Seventh Edition, McGraw Hill Publisher
- Rosenberg Doug, Scott Kendall, 2005, Applying Use Case Driven Object Modelling with UML, Addison-Wisley
- Rizky Soetam, 2011, Konsep Dasar Rekayasa Perangkat Lunak {Software Engineering}, Prestasi Pustaka Publisher
- Rosa A.S, Salahuddin M., 2011, Rekayasa Perangkat Lunak (Terstruktur dan Berorientasi Objek), Penerbit Modula
- Soliq, 2006, Pemodelan Sistem Informasi Berorientasi Objek dengan UML, Penerbit Graha Ilmu
- Sommerville Ian, 2011, Software Engineering Ninth Edition, Addison-Wesley
- Surendro Kridanto, 2009, Pengembangan Rencana Induk Sistem Informasi, Penerbit INFORMATIKA

Biodata Penulis

Penulis saat ini bekerja di Pusat Penelitian Informatika (P2I) – Lemaga Ilmu Pengetahuan Indonesia (LIPI), lulus pendidikan tinggi di bidang informatika untuk program S1, S2, dan S3 masing-masing pada tahun 1993, 1995, dan 2004. Saat ini disamping sebagai PNS mengajar di beberapa perguruan tinggi swasta di Kota Bandung. Penelitian yang diminati yaitu *Software Engineering*, *Kansei Engineering*, *Human Computer Interaction*, dan Sistem Informasi.