

LECTURE NOTES

Software Engineering

Minggu 01

Introduction to Software Engineering

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO1 – Menjelaskan konsep dari proses model piranti lunak

Outline Materi (Sub-Topic) :

1. *Software Engineering*
2. *Software Costs*
3. *FAQs about Software Engineering*
4. *What are the Costs of Software Engineering?*
5. *What are Software Engineering Methods?*
6. *What is CASE (Computer-Aided Software Engineering)*
7. *What are the Attributes of Good Software?*
8. *What are the Key Challenges Facing Software Engineering?*
9. *Professional and Ethical Responsibility*
10. *Kesimpulan*

ISI MATERI

1. *Software Engineering*

Software engineering atau rekayasa perangkat lunak merupakan salah satu cabang keilmuan pada bidang teknologi informasi yang berkembang sangat pesat. Ilmu ini sangat diperlukan untuk memastikan bahwa software yang dibangun memiliki kualitas yang baik. *Software engineering* berfokus pada pembelajaran teori, metode atau alat-alat pendukung yang secara profesional mengembangkan sebuah piranti lunak (*software*). Pada masa ini, hampir semua aspek kehidupan manusia telah menggunakan komputer, misalnya penggunaan perangkat lunak pada bidang:

- Politik: ada aplikasi untuk perhitungan hasil pilpres, kampanye kandidat capres, dll
- Ekonomi: terdapat aplikasi untuk jual beli barang, perhitungan saham
- Hiburan: aplikasi pemutar video, musik, dll
- Perbankan: adanya aplikasi *e-banking*, *mobile banking*, dll

Apalikasi tersebut tentunya sangat mendukung kebutuhan manusia, sehingga *software* sebagai bagian penting dari komputer mendapatkan perhatian khusus dalam pengembangannya. Bahkan pada negara-negara yang telah maju, bahkan di Indonesia sendiri, hamper semua kegiatan telah bergantung pada *software*.

2. *Software Costs*

Software cost adalah biaya yang berhubungan dengan proses pembuatan dan pemeliharaan perangkat lunak, dapat termasuk:

- Biaya lisensi
- Biaya pembuatan

- Biaya pemeliharaan

Contoh biaya lisensi, adalah biaya untuk pembelian aplikasi, misalnya pembelian software untuk basis data, system operasi, anti virus, dan lain-lain.

Jika dibandingkan dengan biaya yang dikeluarkan untuk membeli *hardware*, biaya yang dikeluarkan untuk pengembangan *software* selalu mendominasi keseluruhan pengembangan sebuah sistem. Dan melihat lebih jauh kedalam tahapan pengembangan *software* (mulai dari analisis kebutuhan, perencanaan, perancangan, pembuatan *software*, hingga maintenance), biaya yang dikeluarkan pada tahapan maintenance memakan porsi yang paling besar. Bahkan biaya maintenance dapat berkali-kali lipat dari biaya pembuatan *software* itu sendiri. Salah satu focus pembelajaran dalam *software engineering* adalah pengembangan *software* yang *cost-effective*.

3. FAQs about *Software Engineering*

Berikut ini adalah beberapa pertanyaan yang seringkali muncul terkait dengan *software engineering*:

a. Apa itu *software*?

Software adalah program komputer dan semua dokumentasi yang terkait dengan program tersebut. Dokumentasi dapat berupa kebutuhan-kebutuhan pada tahapan analisis, model-model hasil perancangan, database, maupun dokumen cara menggunakan program tersebut.

Berdasarkan penggunaannya, *software* dapat dikategorikan menjadi dua tipe:

- *Software generic*, *software generic* dikembangkan untuk dipasarkan pada kalayak umum yang memiliki berbagai jenis pengguna.

- Contoh:
 - aplikasi untuk mendukung pekerjaan sehari-hari: Microsoft seperti Word atau Excel.
 - Aplikasi basis data
 - Aplikasi untuk anti virus
- *Software bespoke (custom)*, *software custom* merupakan *software* yang dikembangkan untuk user secara spesifik berdasarkan kebutuhan user tersebut.

Dalam pembuatannya, *software* dapat dibuat dengan mengembangkan program baru, melakukan konfigurasi terhadap program general, atau menggunakan kembali (reuse) sistem *software* yang sudah ada.

Contoh:

- Aplikasi *internet banking* yang digunakan sesuai untuk bank tertentu
- Aplikasi keuangan pada perusahaan tertentu yang dibuat berdasarkan kebutuhan bisnis yang spesifik pada perusahaan tersebut
- Aplikasi company profile dengan menggunakan web pada perusahaan atau organisasi tertentu.

b. **Apa itu *software engineering*?**

Software engineering seperti dibahas sebelumnya adalah cabang ilmu teknik yang berfokus pada semua aspek pengembangan sebuah *software*. Seorang *software engineer* harus mengadopsi pendekatan yang sistematis dan terorganisir dalam pekerjaannya, serta menggunakan tool dan teknik yang sesuai untuk masing-masing masalah yang dihadapi.

Perkembangan software engineering juga sangat pesat sekali, dan saat ini terdapat banyak sekali metode pengembangan software engineering, seperti:

- Rational Unified Process (RUP)
- Scrum
- Extreme Programming (XP)
- Crystal
- DSDM

c. **Apa perbedaan antara *software engineering* dan *computer science*?**

Perbedaan utama dari *software engineering* dan *computer science* adalah fokus pembelajarannya. *Computer science* fokus pada teori dan konsep yang fundamental, sedangkan *software engineering* berfokus pada praktek dalam pengembangan dan penyampaian sebuah *software*.

Pada dunia industri, perbedaan ini tidak terlalu terlihat, dan saling melengkapi satu sama lain, walau pun di dalam materi pengajarannya terdapat hal-hal yang dapat digunakan, yang satu bersifat konsep, sedangkan lainnya bersifat pada praktik.

d. **Apa perbedaan antara *system engineering* dan *software engineering*?**

Perbedaan antara *system engineering* dan *software engineering* adalah *system engineering* meliputi semua aspek pada pengembangan sebuah sistem, seperti hardware, *software* dan sistem itu sendiri. Sedangkan *software engineering* hanya berfokus pada komponen *software*, proses dari sebuah sistem yang meliputi pengembangan infrastruktur, control, aplikasi serta database sistem.

Di dalam praktiknya, kedua bidang ini sangat saling mendukung. Biasanya system engineering dilakukan terlebih dahulu sebelum software engineering.

Contoh:

Di dalam proyek pengembangan perangkat lunak, sebelum seorang developer membuat program pada komputer, tentunya diperlukan instalasi pada system komputer yang digunakan, seperti:

- Instalasi komponen perangkat keras
- Instalasi jaringan
- Instalasi system operasi

e. Apa itu proses *software*?

Proses pada sebuah *software* merupakan aktifitas yang bertujuan untuk membuat atau mengembangkan sebuah *software*. Berikut ini adalah aktifitas umum dalam pengembangan *software*:

- Spesifikasi, apa saja yang harus dilakukan oleh sistem dan batasan-batasan pengembangannya.
- Pengembangan, pembuatan *software* itu sendiri.
- Validasi, pengujian terhadap *software* yang telah dibuat, apakah sudah memenuhi kebutuhan dari user.
- Evolusi, perubahan dari *software* sebagai bentuk dari tanggapan atas permintaan terhadap perubahan.

f. **Apa itu model proses *software*?**

Model proses dari sebuah *software* merupakan proses *software* yang telah disederhanakan yang menunjukkan suatu perspektif yang lebih spesifik. Contoh perspektif dari proses:

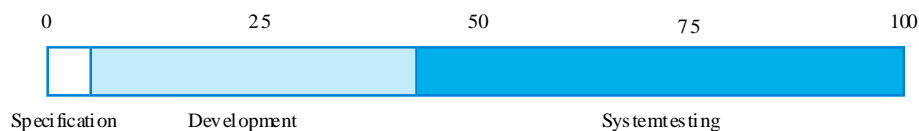
- *Workflow*, urutan aktifitas
- *Data flow*, aliran informasi
- *Role/action*, pengguna aplikasi

Terdapat beberapa proses model umum yang telah dikembangkan yang dapat digunakan langsung dalam pengembangan sebuah *software*:

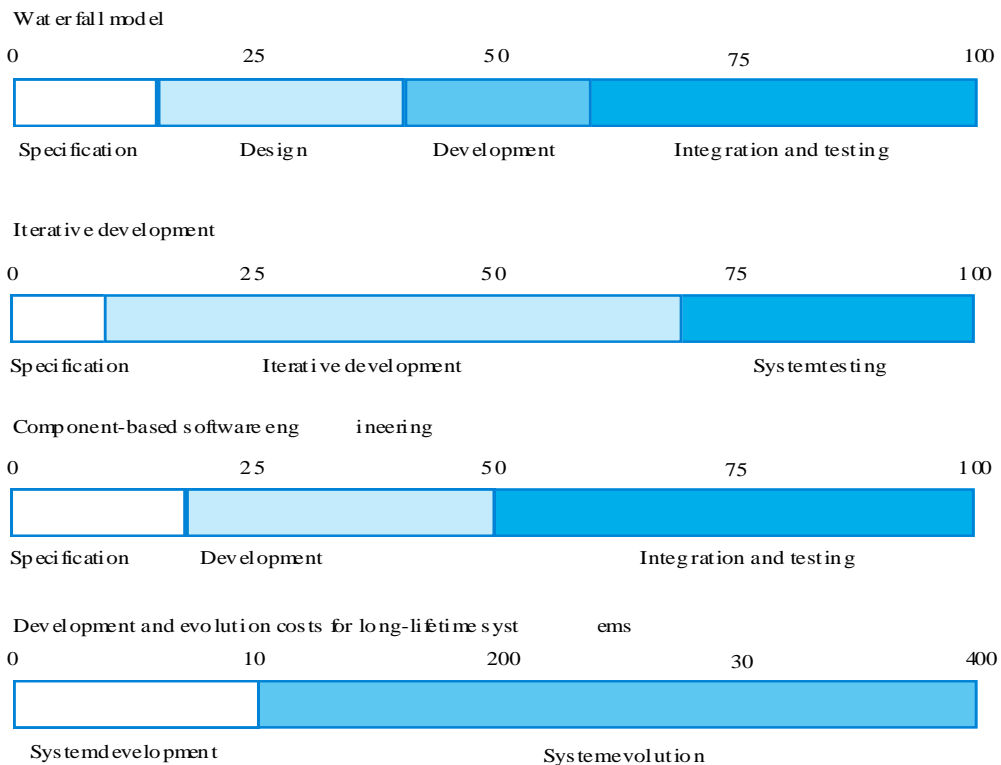
- *Waterfall*
- *Iterative development*
- *Component-based software engineering*

4. *What are the Costs of Software Engineering?*

Biaya dari *software engineering* biasanya dihabiskan 60% untuk proses pembuatannya, dan 40% untuk proses pengujiannya.



Biaya pengembangan *software* tentu saja akan beragam, bergantung pada jenis dan ukuran dari *software* yang akan dikembangkan. Sedangkan biaya distribusi dari sebuah *software* bergantung pada model pengembangan yang digunakan seperti yang digambarkan pada ilustrasi di bawah ini:



5. What are Software Engineering Methods?

Metode *software engineering* merupakan pendekatan terstruktur dalam pengembangan *software* yang meliputi model sistem, notasi, aturan, saran perancangan, serta panduan dari proses.

Contoh:

Scrum merupakan salah satu metode software engineering, yang awalnya merupakan suatu framework yang dikembangkan menjadi metode. Di dalam Scrum, terdapat beberapa komponen, yaitu:

- Peran
 - *Product Owner*
 - *Developer*
 - *Scrum Master*

- *Events*
 - *Sprint*
 - *Sprint Planning*
 - *Daily Scrum*
 - *Sprint Review*
 - *Sprint Retrospective*
- *Artifacts*
 - *Product Backlog*
 - *Sprint Backlog*
 - *Increment*

6. What is CASE (Computer-Aided Software Engineering)

CASE merupakan sistem *software* yang bertujuan untuk menyediakan dukungan otomatisasi terhadap aktifitas-aktifitas dalam pengembangan *software*. *Case* dapat dibagi menjadi dua kategori:

- Upper *CASE*, tool untuk mendukung aktifitas pada proses awal pengembangan *software*, seperti analisis kebutuhan dan perancangan model.
- Lower *CASE*, tool yang dikembangkan untuk mendukung aktifitas lebih lanjut dalam pengembangan sebuah *software*, mulai dari pemrograman, debugging serta pengujian program.

Contoh dari CASE misalnya:

- Aplikasi untuk membuat proses perencanaan
- Aplikasi untuk menyediakan fungsi untuk konfigurasi
- Alat bantu testing

7. *What are the Attributes of Good Software?*

Sebuah *software* harus memenuhi kebutuhan fungsionalitas dan performance dari *user*, dan memiliki sifat *maintainable*, *dependable*, *efficient*, serta *acceptable*.

- *Maintainable*: *software* dapat dikembangkan (berevolusi) untuk memenuhi kebutuhan akan perubahan.
- *Dependability*: *software* harus dapat dipercaya.
- *Efficiency*: *software* harus dapat menghemat resource yang dimiliki.
- *Acceptability*: *software* harus dapat diterima oleh user.

Contoh:

Misalnya Anda sebagai seorang software engineer diminta oleh pelanggan untuk membangun suatu aplikasi ERP, maka Anda harus memerhatikan komponen berikut di dalam pembangunan software, misalnya:

- Tampilan aplikasi yang dapat dimengerti oleh pengguna
- Software dapat dikembangkan lagi dengan membuat aturan-aturan dari system yang dapat dibuat parameter, tidak dibuat *hard coded* di dalam program Anda.

- Aplikasi sudah dapat diterima dalam artian misalnya sudah lolos uji *user acceptance test* sebelum digunakan.

8. *What are the Key Challenges Facing Software Engineering?*

Tantangan yang dihadapi dalam *software engineering*, diantaranya adalah:

- *Heterogeneity*. Mengembangkan teknik dalam mengembangkan *software* yang dalam meliputi platform dan lingkungan eksekusi yang beragam.
- *Delivery*. Mengembangkan teknik penyampaian *software* kepada user yang cepat (tidak memakan waktu lama).
- *Trust*. Mengembangkan teknik yang dapat mendemonstrasikan bahwa *software* dapat dipercaya.

9. *Professional and Ethical Responsibility*

Berikut ini adalah beberapa isu dalam tanggung jawab profesional seorang *software engineer*:

- Kerahasiaan. Seorang *software engineer* harus dapat menghormati kerahasiaan karyawan atau klien, meskipun tidak ada persetujuan formal yang ditanda tangani.
- Kompetensi. Seorang *software engineer* harus dapat menunjukan kompetensi yang dimiliki.
- Hak kekayaan intelektual. *Software engineer* harus memperhatikan hukum yang mengatur penggunaan properti intelektual seperti paten, *copyright*, dll.
- Penyalahgunaan komputer. *Software engineer* tidak boleh menyalahgunakan kemampuan teknisnya.

Misalnya:

Anda seorang software engineer yang membuat aplikasi untuk system penggajian di mana Anda mengetahui daftar gaji dari basis data pelanggan Anda, maka Anda tidak boleh menyebarkan informasi ini kepada pihak mana pun.

KESIMPULAN

Sebagai seorang *software engineer*, Anda harus mengetahui prinsip-prinsip dasar dari software, misalnya definisi, metode pembangunan *software*, bagaimana membuat *software* yang dapat digunakan oleh pelanggan. Anda juga harus mengetahui bagaimana etika sebagai seorang *software engineer* di dalam menangani pekerjaan yang dilakukan.

DAFTAR PUSTAKA

- Introduction to *Software Engineering*,
<http://www.youtube.com/watch?v=Z6f9ckEElsU>
- *Software engineering : a practitioners approach* : Chapter 3/ Pages 30, Chapter 4/Pages 40, Chapter 5/Pages 66, Chapter 6/Pages 87
- *Software Engineering Incremental Model*,
<http://www.youtube.com/watch?v=9cBkihYP1rY>
- Video Water Fall, V model,, <http://www.youtube.com/watch?v=KaPC0gsEQ68>
- *Software engineering : a practitioners approach* : Chapter 8/Pages 131
Chapter 9/Pages 166, Chapter 10/ Pages 184, *Requirements Engineering / Specification*., <http://www.youtube.com/watch?v=wEr6mwquPLY>
- Collaborative Requirements Management,,
<http://www.youtube.com/watch?v=tEXizjE05LA>
- UML 2.0 Tutorial,, <http://www.youtube.com/watch?v=OkC7HKtiZC0>
- *Software engineering : a practitioners approach* : Chapter 12/Pages 224, Chapter 13/Pages 262, Chapter 14/Pages 285, Chapter 15/Pages 317, 16/Pages 347, 17/Pages 371, dan 18/Pages 391
- Introduction to *Software Architecture*.,
<http://www.youtube.com/watch?v=x30DcBfCJRI>
- Component-based game engine,,
http://www.youtube.com/watch?v=_K4Mc3t9Rtc
- *Software design pattern*., http://www.youtube.com/watch?v=ehGl_V6lWJw
- *Software engineering : a practitioners approach* : Chapter 19/Pages 412 & chapter 20/Pages 431
- Introduction to *software quality assurance*.,
http://www.youtube.com/watch?v=5_cTi5xBIYg
- *Software reliability* ,, http://www.youtube.com/watch?v=ww51aF_qODA Lean Six Sigma and IEEE standards for better *software engineering*.,
<http://www.youtube.com/watch?v=oCkPD5YvWqw>

LECTURE NOTES

Software Engineering

Minggu 2

Software Process Model

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO1 – Menjelaskan konsep dari proses model piranti lunak

Outline Materi (Sub-Topic) :

1. *Software Process Structure*
2. *Process Models*
3. *Agile Development*
4. *Human Aspects of Software Engineering*
5. Studi Kasus

ISI MATERI

1. *Software Process Structure*

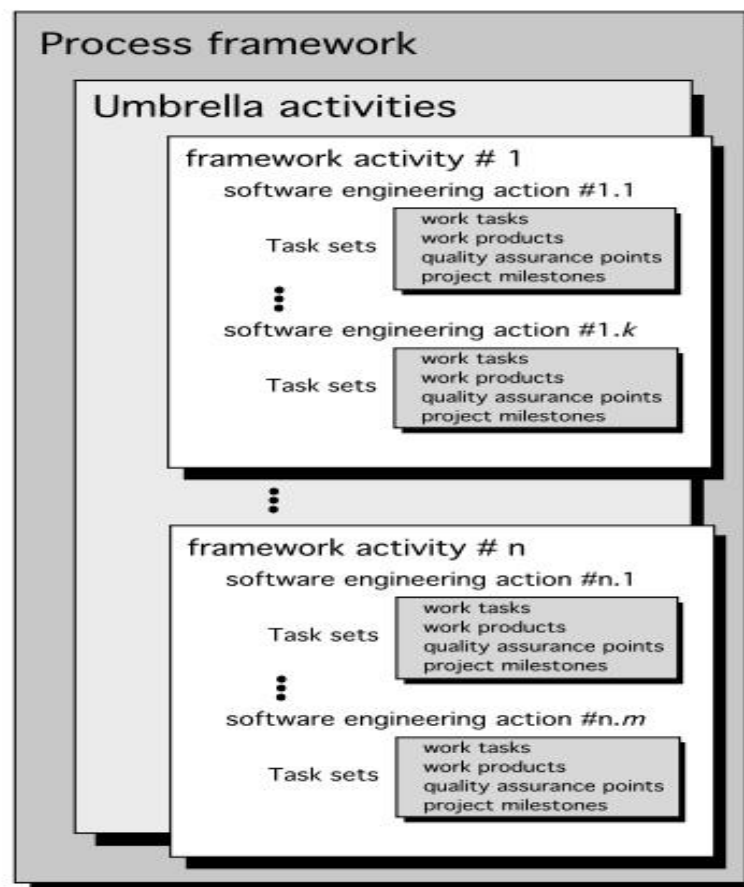
Software Process Structure berhubungan dengan struktur atau komponen yang dijadikan dasar di dalam pengembangan suatu *software*.

Misalnya:

Anda akan membuat suatu aplikasi kepegawaian pada perusahaan, maka Anda dapat mengikut salah satu proses pengembangan software, yang dapat terdiri dari kerangka kerja atau framework, metode dan kegiatan-kegiatan di dalamnya yang mendukung proses pembuatan software.

Berikut ini adalah kerangka proses dari pengembangan sebuah *software*:

Software process



Pada setiap kerangka aktivitas akan terdapat sekumpulan tugas yang harus dijalankan dalam satu proses pengembangan *software*. Sekumpulan tugas tersebut dapat didefinisikan sebagai kerja nyata yang dilakukan untuk mencapai tujuan dari software engineering. Sekumpulan tugas tersebut dapat berupa:

- Daftar tugas yang harus diselesaikan
- Daftar produk kerja yang akan dihasilkan
- Daftar filter penjaminan kualitas yang harus dipakai

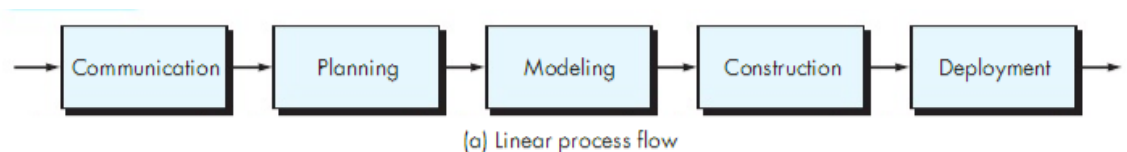
Contoh kegiatan di dalam proses pengembangan perangkat lunak misalnya:

- Mendesain basis data
- Mendesain user interface
- Mendesain suatu class
- Mendesain beberapa fungsi yang digunakan

Aliran proses dari struktur proses sebuah software dapat dibagi menjadi beberapa tipe:

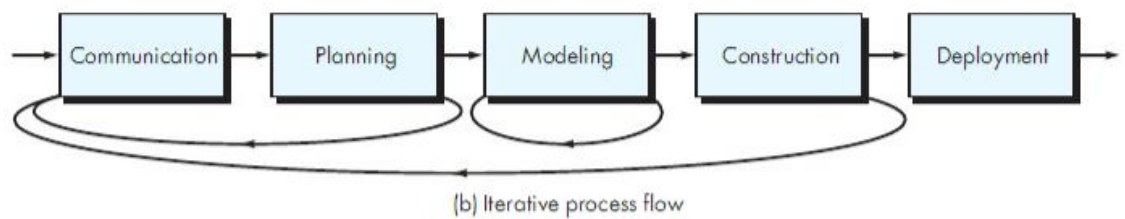
- Aliran proses linear.

Pada aliran proses linear, proses dilakukan dalam satu *flow* dari awal hingga akhir pada satu garis lurus. Pada gambar dibawah ini, dapat dilihat bahwa aliran proses dimulai dari proses komunikasi, dilanjutkan pada proses perencanaan, pemodelan, pembuatan *software* hingga deployment. Pada aliran proses linear, tidak memungkinkan untuk dilakukan aliran balik dari proses yang telah dilakukan ke proses sebelumnya.



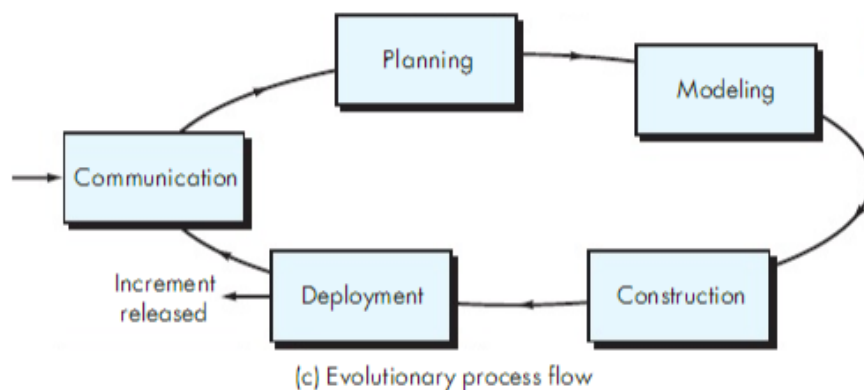
- Aliran proses iterative (berulang)

Aliran proses iterative merupakan aliran proses yang dapat dilakukan secara berulang. Pada contoh dibawah ini, setelah melewati proses komunikasi dan perencanaan, apabila dibutuhkan melakukan komunikasi ulang, proses dapat kembali ke tahapan sebelumnya. Begitu pula proses perulangan pada tahapan itu sendiri.



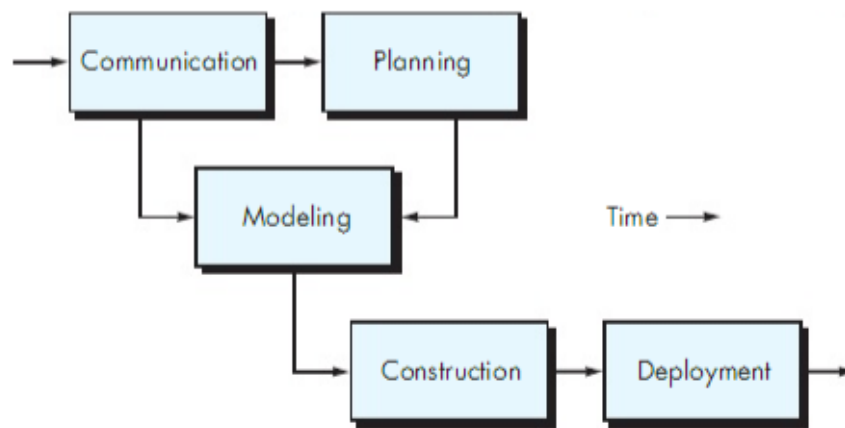
- Aliran proses evolusioner

Aliran proses evolusioner merupakan proses yang melakukan iterasi terhadap satu siklus proses yang ada. Seperti tampak pada gambar, sebuah software dapat dibagi menjadi beberapa modul, dimana satu modul akan melewati satu fase proses dari komunikasi hingga *deployment*, kemudian dimulai lagi untuk komunikasi jika dibutuhkan perbaikan atau lanjut ke modul selanjutnya. Berikut ini adalah ilustrasi dari aliran proses evolusioner:



- **Aliran proses parallel**

Aliran proses parallel menunjukkan proses yang dapat dijalankan secara bersamaan antara 2 atau lebih proses. Seperti pada contoh dibawah ini, planning dan modeling dilakukan secara parallel setelah proses komunikasi dilakukan.



(d) Parallel process flow

2. *Process Models*

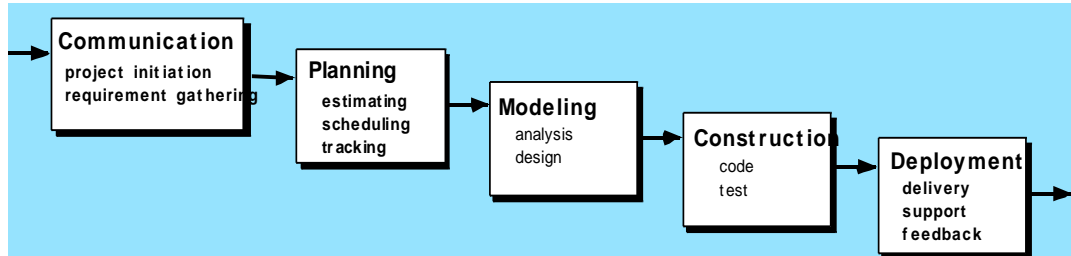
Proses model dibagi menjadi dua jenis: *prescriptive* dan *agile*. *Software* process model *prescriptive* menganjurkan pendekatan yang teratur dan cenderung kaku. Hal ini menimbulkan beberapa pertanyaan, seperti:

- Jika model *prescriptive* menuntut model yang terstruktur dan teratur, apakah model tersebut cocok digunakan untuk mengembangkan *software* yang membutuhkan perubahan?
- Sebaliknya, jika proses model *prescriptive* yang lebih tradisional ini diganti dengan model yang kurang terstruktur, apakah mungkin untuk melakukan koordinasi dan koherensi kerja yang baik dalam pengembangan sebuah *software*?

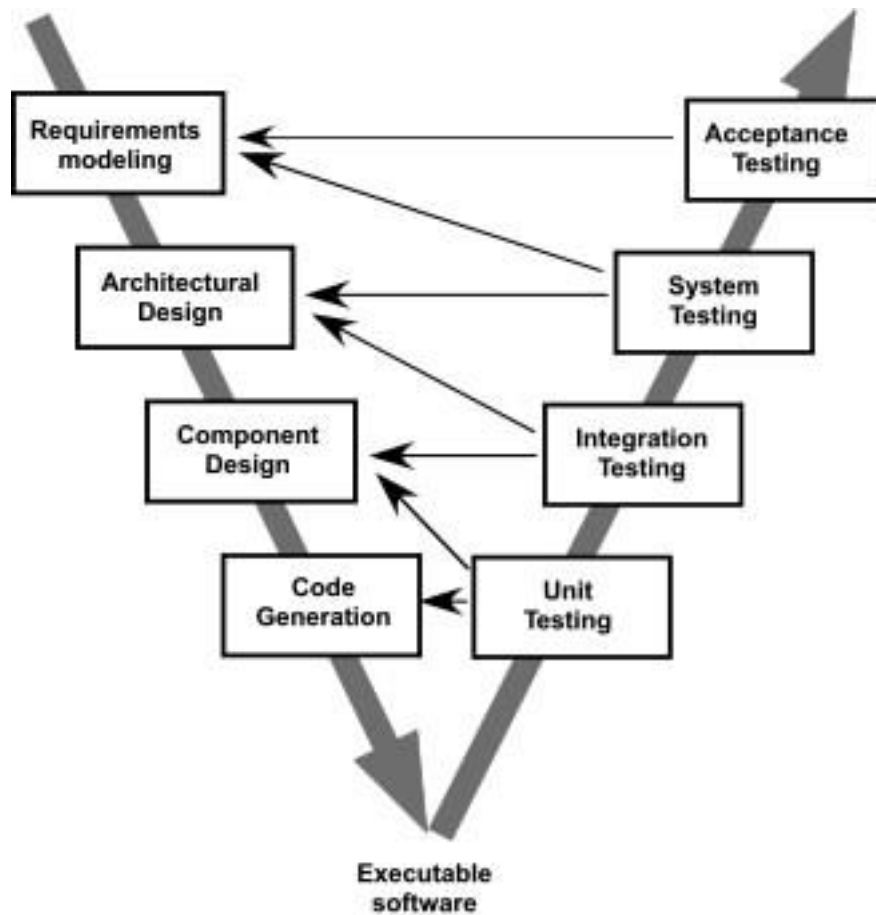
Silahkan dilakukan diskusi mengenai kedua pertanyaan diatas.

Berikut ini adalah contoh proses model *prescriptive*:

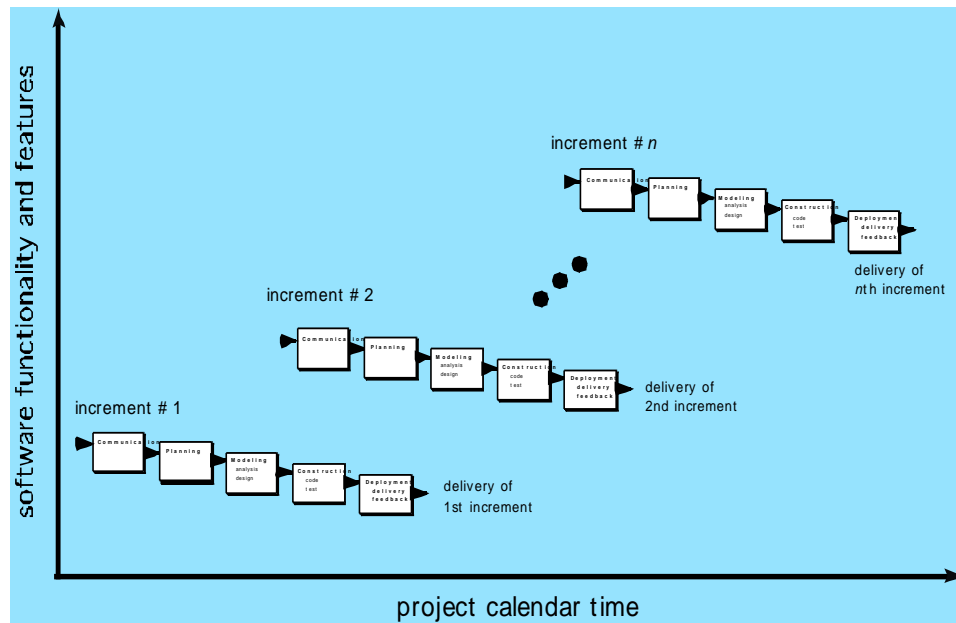
A. Waterfall Model



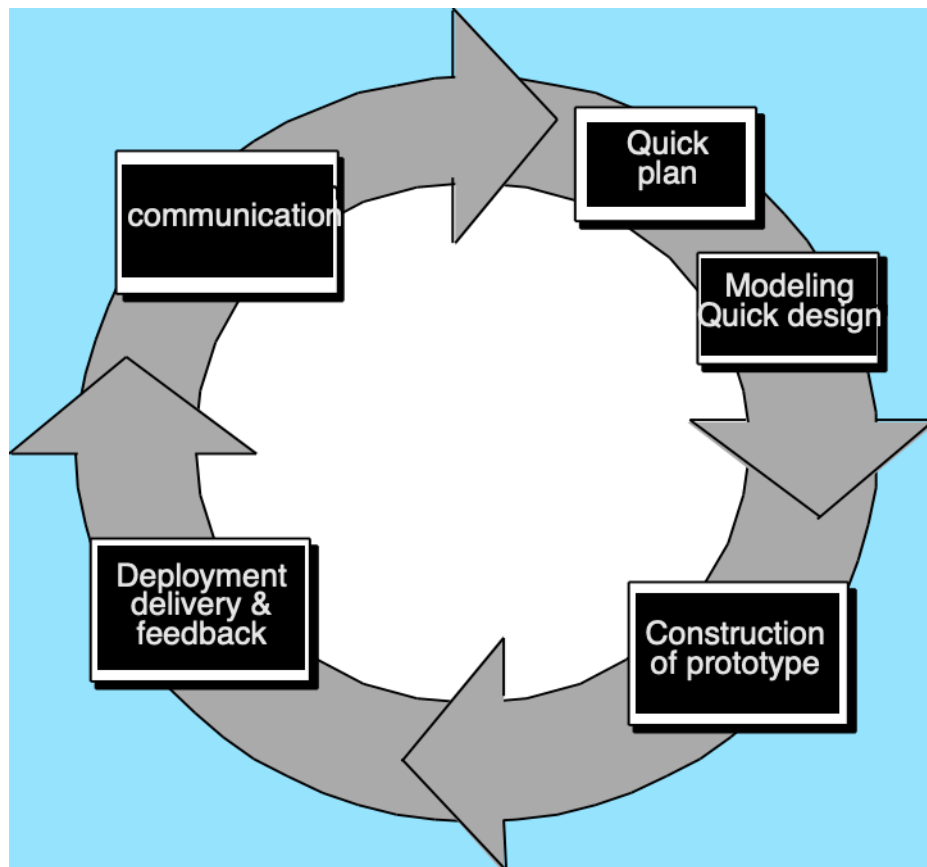
B. V- Model



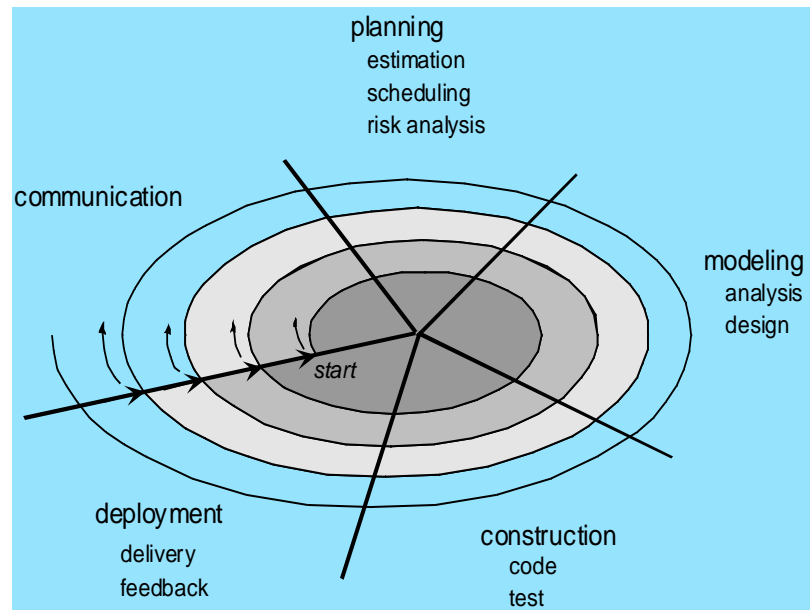
C. Incremental Model



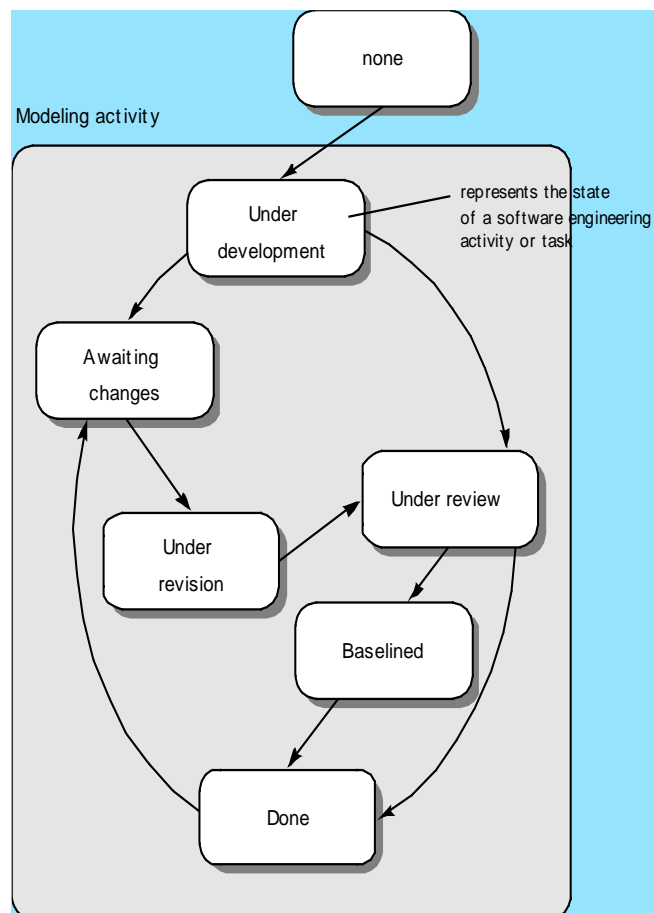
D. Model evolutioner: Prototyping



E. Model evolutioner: *Spiral*



F. Model evolutioner: *Concurrent*

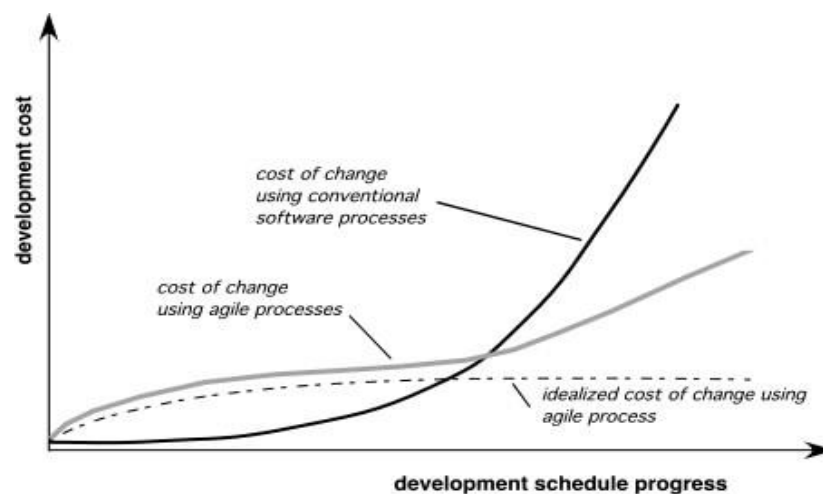


3. Agile Development

Agility merupakan:

- Efektif (cepat dan *adaptive*) dalam merespon perubahan yang ada
- Komunikasi yang efektif di antara *stakeholders*
- Melibatkan customer dalam tim pengembang
- Mengorganisasikan tim sehingga dapat mengontrol kinerja dari pekerjaan
- Cepat, dan secara *incremental software* disampaikan kepada *customer*

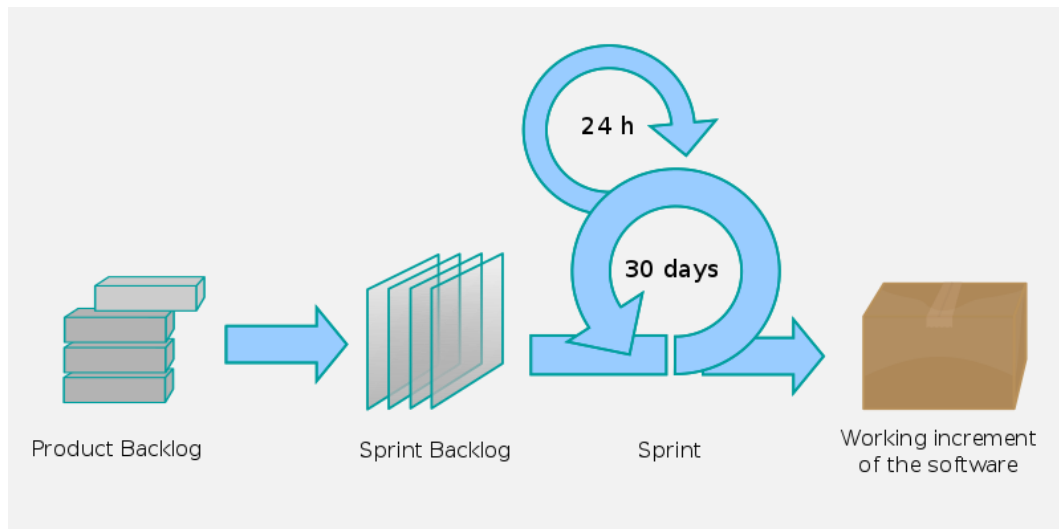
Berikut ini adalah diagram yang menggambarkan hubungan antara *agility* dan biaya yang dikeluarkan untuk mengakomodasi perubahan:



Ada beberapa tipe proses model *agile*:

- *Extreme programming*
- *Adaptive software development*
- *Dynamic system development method*
- *Scrum*
- *Crystal*
- *Feature Driven Development*

Masing-masing proses model di atas memiliki kelebihan dan kekurangan. Kita juga dapat melakukan kombinasi di dalam penggunaan metode tersebut, misalnya kombinasi antara implementasi Scrum dan Extreme Programming.



Contoh Proses Scrum

Di dalam menjalankan proyek pengembangan perangkat lunak berbasis agile, para software engineer harus mengikuti panduan pelaksanaan agile seperti yang tertuang pada agile manifestore berikut:

“Kami menemukan cara yang lebih baik untuk mengembangkan perangkat lunak dengan melakukan dan membantu sesama untuk menggunakannya.

Melalui usaha ini kami telah dapat menghargai:

Individu dan interaksi lebih dari proses dan sarana perangkat lunak

Perangkat lunak yang bekerja lebih dari dokumentasi yang menyeluruh

Kolaborasi dengan klien lebih dari negosiasi kontrak

Tanggap terhadap perubahan lebih dari mengikuti rencana

Demikian, walaupun kami menghargai hal di sisi kanan, kami lebih menghargai hal di sisi kiri.”

4. Human Aspects of Software Engineering

Karakteristik dari software engineer yang dikategorikan sebagai sifat yang efektif:

- Memiliki kesadaran akan tanggung jawab individual
- Memiliki kesadaran yang tajam
- Jujur
- Ketahanan dalam menghadapi tekanan
- Memiliki rasa keadilan yang tinggi
- Memberikan perhatian pada detail
- Pragmatis

Studi Kasus

Sebelum memulai mengerjakan proyek pengembangan perangkat lunak, Anda harus menentukan terlebih dahulu mengenai metode apa yang dipakai apakah menggunakan metode waterfall dan agile. Jika business requirement dapat ditentukan secara jelas pada saat awal pengembangan, maka waterfall cocok untuk digunakan, sedangkan jika business requirement tidak dapat ditentukan dengan jelas di awal dan terdapat ketidakpastian yang tinggi, maka pendekatan agile lebih baik untuk digunakan.

DAFTAR PUSTAKA

- Software engineering : a practitioners approach : Chapter 3/ Pages 30, Chapter 4/Pages 40, Chapter 5/Pages 66, Chapter 6/Pages 87
- Software Engineering Incremental Model,
<http://www.youtube.com/watch?v=9cBkihYP1rY>
- Video Water Fall, V model,, <http://www.youtube.com/watch?v=KaPC0gsEQ68>
- https://id.wikipedia.org/wiki/Berkas:Scrum_process.svg

LECTURE NOTES

Software Engineering

Minggu 3

Requirement Modelling

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO2 – Menjelaskan konsep dari proses model piranti lunak

Outline Materi (Sub-Topic) :

1. *Requirements Engineering*
2. *Eliciting Requirement*
3. *Developing Use Case*
4. *Negotiating Requirement*
5. *Validating Requirements*
6. Studi Kasus

ISI MATERI

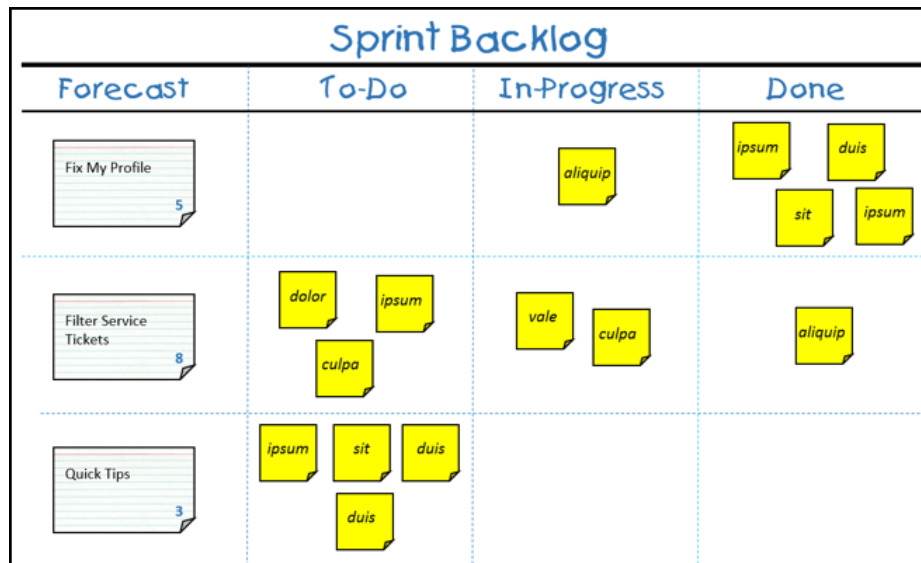
1. *Requirements Engineering*

Requirement engineering berhubungan bagaimana seorang *software engineer* melakukan proses untuk mendapatkan kebutuhan bisnis dari pengguna perangkat lunak dan mengelola kebutuhan tersebut untuk disiapkan menjadi proses selanjutnya, yaitu desain dan pengembangan. Seorang *software engineer* harus memiliki pengetahuan dan ketrampilan di dalam hal melakukan proses *requirement engineering* ini.

Requirement engineering merupakan suatu proses yang sangat penting di dalam proses pembangunan perangkat lunak. Banyak kasus yang terjadi akibat terjadi perbedaan persepsi atau ketidakkonsistenan antara kebutuhan pelanggan dan hasil dari perangkat lunak yang dibangun. Jika proses ini tidak dilakukan secara benar, maka akan terjadi potensi kegagalan implementasi perangkat lunak.

Pada proyek yang berbasis agile, *requirement engineering* ini dapat dilakukan dengan menggunakan konsep product backlog. Product backlog merupakan kumpulan dari requirement menggunakan konsep user stories. Berikut contoh dari product backlog menggunakan papan Kanban. Manajemen requirement ini menggunakan konsep backlog refinement.

Bahasan-bahasan selebihnya pada dokumen ini, lebih menitik beratkan proses pembentukan requirement dengan menggunakan notasi UML. Penggunaan notasi UML sangat baik digunakan untuk melakukan proses requirement untuk tipe pengembangan menggunakan konsep waterfall.



Requirement engineering dapat dikategorikan menjadi beberapa model:

- Model berdasarkan *scenario*, contoh: *use case diagram*, *user story*, *activity diagram*, *use case scenario*, dll. Pada pemodelan berdasarkan *scenario* bertujuan untuk membantu mendefinisikan *actor* dari sistem dan apa yang harus dilakukan oleh sistem tersebut.

Use case diagram merupakan salah satu model yang sering dipakai untuk menentukan fitur-fitur dari system yang akan dibangun. Use case menunjukkan fitur atau fungsionalitas yang ada pada system, misalnya:

- Fungsi untuk melihat saldo
- Fungsi untuk menarik uang di ATM
- Fungsi untuk transfer antar rekening
- Fungsi untuk melakukan administrasi perubahan akun

b. Model berdasarkan kelas, contoh: *class diagram*, *collaboration diagram*, dll. Model berdasarkan kelas menggambarkan beberapa hal berikut:

- *Object* dari sistem yang akan dimanipulasi
- Operasi (metode atau *service*) yang akan diterapkan terhadap *object* untuk mendapatkan efek dari manipulasi yang dilakukan.
- Hubungan antar *object*
- Kolaborasi yang muncul antara kelas yang didefinisikan.

Contoh:

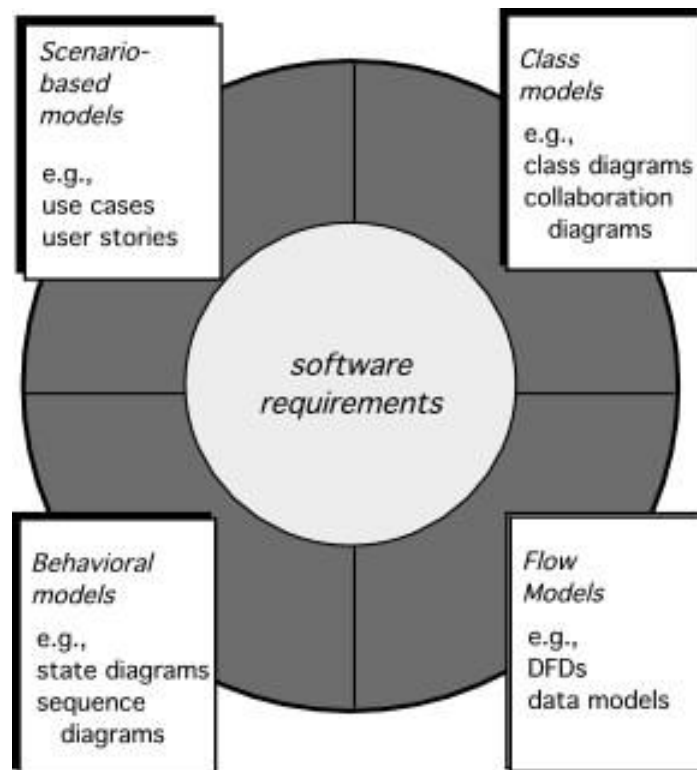
Jika kita ingin membuat suatu system penjualan yang melibatkan data produk, pelanggan dan supplier. Dengan menggunakan model berdasarkan kelas, kita dapat menggambarkan model-model untuk produk, pelanggan dan supplier sebagai kelas-kelas dan obyek yang saling berhubungan.

c. Model berdasarkan *behavior*, contoh *state diagram*, *sequence diagram*, dll. *Model behavior* mengindikasikan bagaimana *software* akan memberikan respon terhadap event atau stimulus external. Untuk menciptakan model ini, proses analisis harus dilakukan melalui tahapan berikut:

- Evaluasi semua *use case* untuk memahami secara keseluruhan urutan interaksi di dalam sistem.
- Identifikasikan sistem yang menggerakkan urutan interaksi dan memahami bagaimana event-event tersebut dikaitkan dengan *object specific*.
- Membuat *sequence* untuk setiap *use case*
- Membangun *state diagram* untuk sistem

- Melakukan *review* terhadap model *behavioral* untuk verifikasi akurasi dan konsistensi

d. Model berdasarkan aliran, contoh: data flow diagram, data *models*, dll.



2. *Eliciting Requirement*

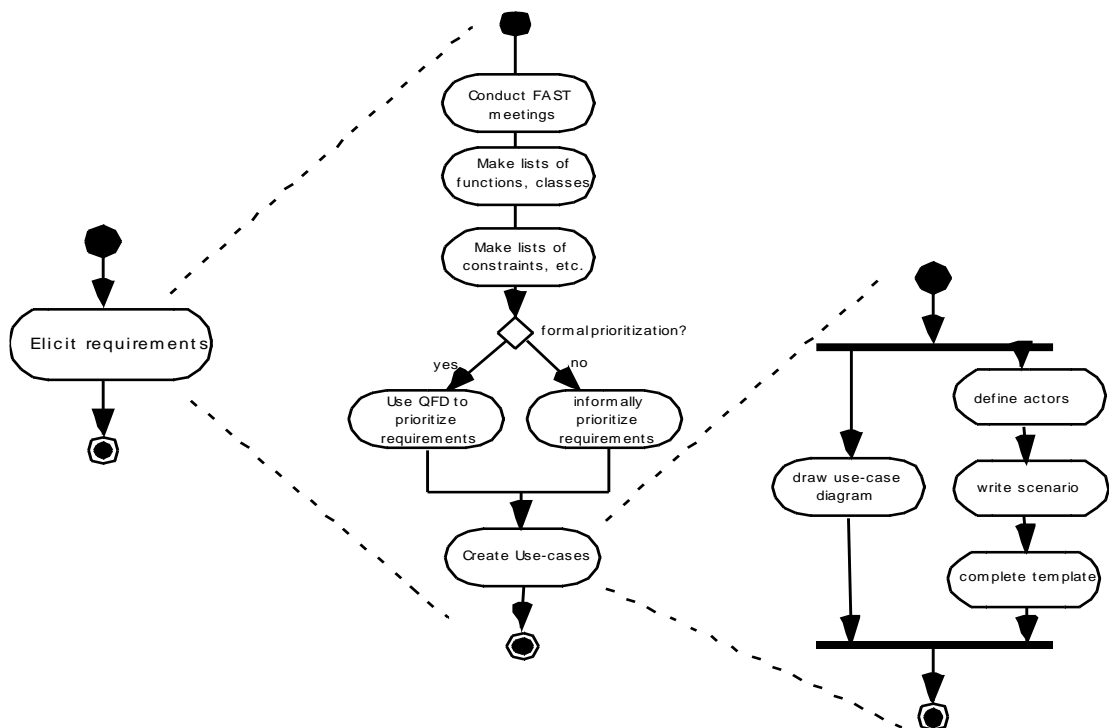
Mendapatkan (*elicit*) *requirement* merupakan salah satu kegiatan dalam *requirement engineering*. Sebelum membahas lebih dalam mengenai *requirement elicitation*, perlu dipahami secara keseluruhan apa saja kegiatan dalam *requirement engineering*:

- *Inception*, menanyakan sekumpulan pertanyaan yang telah dibuat mengenai pemahaman dasar mengenai masalah, siapa saja orang yang menginginkan solusi, bentuk dari solusi yang diinginkan, dan *efektifitate* dari komunikasi dan kolaborasi awal antara *customer* dan *developer*.

- *Elicitation*, mendapatkan semua kebutuhan dari semua *stakeholder* yang terlibat.
- *Elaboration*, membuat model analisis yang mengidentifikasi kebutuhan data, fungsi dan behavioral.
- *Negotiation*, persetujuan mengenai cara pengiriman sistem yang *realistic* untuk developer dan *customer*.
- *Specification*, dapat berupa dokumen tertulis, sekumpulan model, matematika formal, kumpulan dari user scenario, atau prototype.
- *Validation*, mekanisme *review* yang mencari error dalam konten, inkonsistensi, kebutuhan yang tidak realistic/memiliki konflik.
- *Requirement management*, manajemen untuk mengatur keseluruhan kegiatan yang ada dalam *requirement engineering*.

Dalam melakukan eliciting requirements, dibutuhkan sebuah pertemuan / meeting yang diadakan dan dihadiri oleh kedua pihak (*software engineer* dan *customer*). Dalam pertemuan tersebut, akan ditentukan aturan dalam persiapan dan bentuk partisipasi masing-masing pihak. Tujuan utama dari kegiatan eliciting requirements adalah untuk :

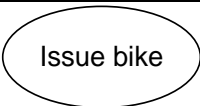
- Identifikasi masalah
- Mengajukan elemen-elemen solusi
- Negosiasi metode-metode yang berbeda
- Menspesifikasikan kebutuhan solusi diawal

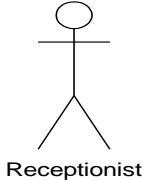

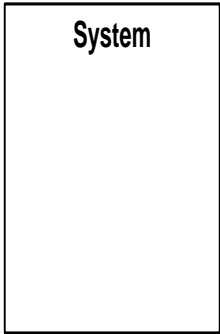


3. Developing Use Case

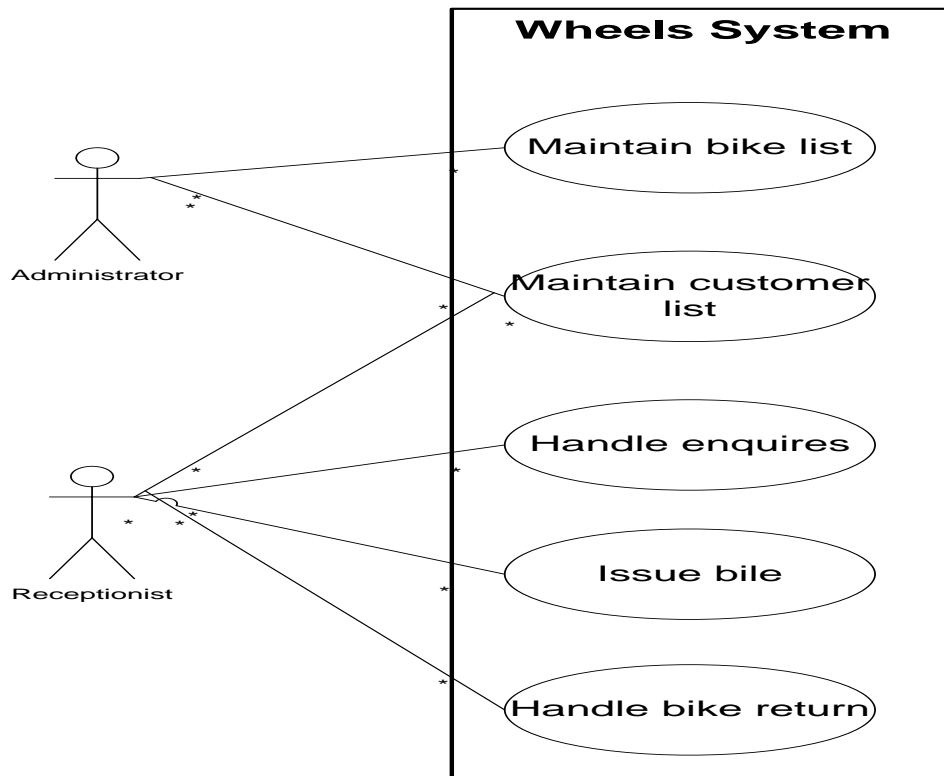
Diagram *use case* menggambarkan fungsionalitas dari sistem dan interaksi antara *user* dengan sistem. Model *use case* menyediakan fasilitas untuk mengorganisasikan, menstruktur dan mendokumentasikan informasi pada tahap pendefinisian kebutuhan sistem.

Berikut ini adalah notasi yang digunakan untuk menggambarkan diagram use case:

Nama	Notasi	Keterangan
<i>Use case</i>		Berbentuk elips dengan keterangan nama <i>use case</i> di tengah elips. Satu <i>use case</i> akan menggambarkan satu fungsionalitas yang dimiliki oleh sistem dan

		digambarkan di dalam notasi batasan sistem.
Aktor	 <p>Receptionist</p>	Semua aktor yang berhubungan langsung dengan sistem akan digambarkan dengan notasi ini.
Hubungan <i>use case</i>		Garis lurus digunakan untuk menghubungkan antara aktor dengan <i>use case</i> yang terkait.
Batasan sistem	 <p>System</p>	Batasan sistem berupa persegi panjang dengan nama sistem di atas. Semua <i>use case</i> yang teridentifikasi akan digambarkan di dalam batasan ini.

Berikut ini adalah contoh dari *use case* diagram untuk sebuah sistem “Wheel”, sistem penyewaan sepeda.



Pada gambar diagram *use case* di atas, terlihat bahwa terdapat dua actor yang akan menggunakan sistem; administrator dan *receptionist*. Administrator dapat menjalankan fungsionalitas untuk melakukan pemeliharaan terhadap daftar sepeda dan pelanggan. Sedangkan receptionist dalam mengakses use case untuk memelihara daftar pelanggan, melakukan penyewaan sepeda, pengembalian sepeda, dan pendataan sepeda.

4. *Negotiating Requirement*

Dalam melakukan negosiasi terhadap requiremen dari sebuah software beberapa hal yang harus diperhatikan adalah:

- Mengidentifikasi *stakeholder* utama (kunci), yang merupakan orang-orang yang akan terkait dalam negosiasi
- Menentukan “*win condition*” untuk setiap *stakeholders*. Kondisi menang disini tidak selalu jelas.
- Negosiasi, melakukan negosiasi langsung terhadap kebutuhan yang memberikan solusi “*win win*”.

5. *Validating Requirements*

Berikut ini adalah pertanyaan yang dapat digunakan untuk melakukan validasi terhadap requirement yang telah dirancang:

- Apakah setiap *requirement* dapat dicapai dalam lingkungan teknis dari sistem atau produk
- Apakah *requirement* dapat diuji ketika telah dikembangkan?
- Apakah *requirement* model secara layak merefleksikan informasi, fungsi dan *behavior* dari sistem yang akan dibuat?
- Sudahkah *requirement* dipartisi sehingga dapat menunjukkan detail informasi secara progresif.
- Sudahkah *pattern requirement* digunakan untuk menyederhanakan model *requirement*. Apakah semua *pattern* secara layak divalidasi? Apakah *pattern* konsisten dengan kebutuhan customer?

Studi Kasus

Di dalam implementasi proyek pengembangan perangkat lunak, tentunya diperlukan dokumentasi yang harus dilakukan, misalnya dalam bentuk:

- Functional Specification Document (FSD)
- Software Requirement Specification (SRS)

Pembentukan dokumen ini dapat mengikuti panduan yang telah dijelaskan, diawali dengan pembentukan use case.

Untuk metode pengembangan dengan agile, dimulai dari pembentukan product backlog dan product backlog item. Product backlog sendiri pada dasarnya merupakan suatu daftar requirement dalam bentuk yang fitur produk yang akan dibangun.

DAFTAR PUSTAKA

- Software engineering : a practitioners approach : Chapter 8/Pages 131
Chapter 9/Pages 166, Chapter 10/ Pages 184,
- Requirements Engineering / Specification,, <http://www.youtube.com/watch?v=wEr6mwquPLY>
- Collaborative Requirements Management,,
<http://www.youtube.com/watch?v=tEXizjE05LA>
- UML 2.0 Tutorial,, <http://www.youtube.com/watch?v=OkC7HKtiZC0>
- Scrum.org

LECTURE NOTES

Software Engineering

Minggu 4

Design Engineering

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO2 – Menjelaskan praktek *software engineering*

Outline Materi (Sub-Topic) :

1. *Architectural Design*
2. *Component-Level Design*
3. *User Interface Design*
4. *Pattern Based Design*
5. *WebApp Design*
6. *Mobile App Design*
7. Studi kasus

ISI MATERI

Proses desain atau design engineering dilakukan setelah fase *requirement engineering*. Proses ini juga sangat penting sebelum dilakukan proses implementasi atau *coding*. Pada proses pengembangan perangkat lunak dengan metode *agile*, proses desain ini dapat dilakukan pada setiap iterasi yang terjadi.

1. *Architectural Design*

Software architecture adalah sebuah desain umum suatu proses pada sebuah *software system*, meliputi:

- Pembagaian *software* ke dalam subsistem
- Memutuskan bagaimana saling berhubungan
- Menentukan alat penghubung

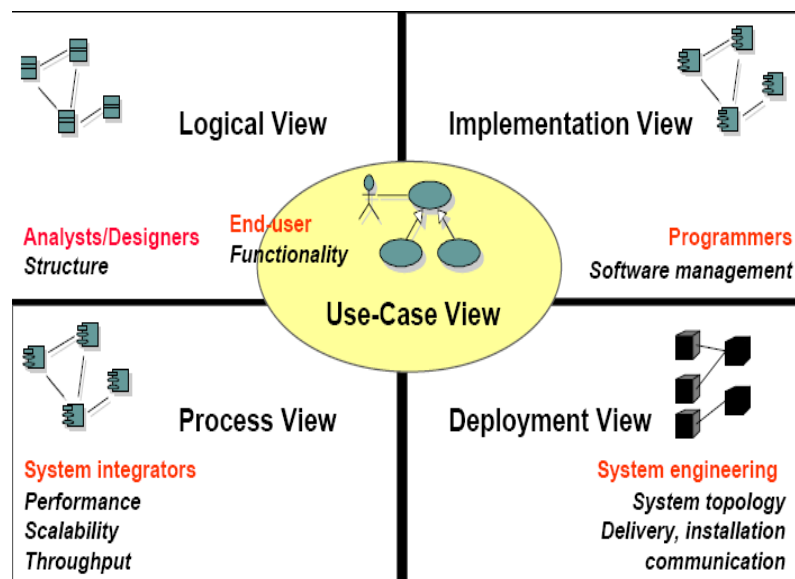
Pentingnya arsitektur sebuah *software*:

- Kenapa kita perlu mengembangkan arsitektur:
 - Agar setiap orang bisa mengerti mengenai sistem yang ada.
 - Untuk membiarkan *user* bekerja secara individual terhadap sebuah sistem
 - Persiapan untuk perluasan *system*
 - Menyediakan fasilitas *reuse* and *reusability*

Struktur dan pandangan arsitektural:

- *View* menampilkan aspek aspek yang terdapat pada *software architecture* yang menunjukkan spesifikasi *software*.
- *Architectural structures*
 - Sebuah sistem *family* yang terkait dengan *pattern*
 - sebuah *vocabulary* dari komponen dan *connector type*
 - Suatu batasan dimana dapat kombinasikan

Architectural structures dapat disebut juga dengan *architectural style*



➤ *Architecture view*

■ *Use Case View*

- Analisa *use case* adalah teknik untuk meng-*capture* proses bisnis dari perspektif *user*.
- Aspek statis di-*capture* dalam *use case diagram*
- Aspek dinamis di-*capture* dalam *interaction diagram*, *statechart diagram* dan *activity diagram*

■ *Design View*

- Meliputi *class-class*, *interface*, dan *collaboration* yang mendefinisikan *vocabulary system*
- Mendukung kebutuhan fungsional *system*
- Aspek statis di-*capture* dalam *class diagram* dan *object diagram*
- Aspek dinamis di-*capture* dalam *interaction diagram*, *statechart diagram* dan *activity diagram*

■ *Process View*

- Meliputi *thread* dan pendefinisian proses-proses *concurrency* dan *synchronization*
- Menunjukkan *performance*, *scalability* dan *throughput*

- Aspek statis dan dinamis di-*capture* dengan *design view*, tetapi lebih menekankan pada *active class*

■ **Implementation View**

- Meliputi komponen dan *file* yang digunakan untuk menghimpun dan me-*release system physic*
- Menunjukkan *configuration management*
- Aspek statis di-*capture* dalam *component diagram*
Aspek dinamis di-*capture* dalam *interaction diagram*, *statechart diagram* dan *activity diagram*

■ **Deployment View**

- Meliputi node yang membentuk topologi *hardware system*
- Menunjukkan pendistribusian, *delivery*, dan pengistallan
- Aspek statis di-*capture* dalam *deployment diagram*
- Aspek dinamis di-*capture* dalam *interaction diagram*, *statechart diagram*, *activity diagram*

Definisi design oleh IEEE6 10.12-90 adalah sebagai berikut : “proses pendefinisian arsitektur, komponen, *interface* dan karakteristik lain dari sistem atau komponen” dan “ hasil dari proses itu”. Di tampilan sebagai proses, *software design* adalah aktivitas terus menerus dari *software engineering* yang mana *software requirements* dianalisa dalam rangka untuk menghasilkan deskripsi dari struktur *internal software* yang berperan sebagai basis untuk konstruksinya. Lebih pastinya, sebuah *software design* (hasilnya) harus dapat mendeskripsikan *arsitektur software*. Karenanya, bagaimana *software* dipecah dan disusun menjadi komponen-komponen, dan tampilan antara komponen-komponen tersebut, harus juga dapat mendeskripsikan komponen pada tingkatan detil yang menyediakan konstruksi mereka.

Software design memainkan peranan penting dalam membangun *software*. *Software design* mengijinkan *software engineers* untuk membuat beberapa model yang membentuk sejenis blueprint dari solusi menjadi implementasi.

➤ **Aktivitas *Software design***

Dalam daftar standar *software life cycle process* seperti pada *Software Life Cycle Processes*, *software design* terdiri atas dua aktivitas yang sangat sesuai antara *software requirements analysis* dan *software construction*:

***Software architectural design* (sering disebut *top level design*):**

- Menggambarkan *software toplevel structure* dan mengorganisasi dan mengidentifikasi berbagai komponen.

***Software detailed design*:**

- Menggambarkan tiap komponen secara cukup mengijinkan untuk konstruksinya.

➤ **General Concepts design**

Software bukan satu-satunya media yang melibatkan desain. Dalam pemahaman secara umum, kita dapat melihat desain sebagai bentuk pemecahan masalah. Sebagai contoh, kita mengambil konsep dari masalah yang tidak mempunyai solusi nyata, sangat menarik sebagai bagian untuk memahami batasan dari desain. Sejumlah ide dan konsep lain juga menarik untuk memahami desain dalam pemahaman umum: tujuan, batasan, alternatif, representasi dan solusi.

➤ ***Software Design Process***

Software design secara umum terdiri atas proses dua langkah:

- ***Architectural Design***

Architectural design mendeskripsikan bagaimana *software* dipecah dan disusun menjadi beberapa komponen (*the software architecture*)

- ***Detailed Design***

Detailed design mendeskripsikan perilaku khusus komponen tersebut. Hasil dari proses tersebut merupakan kumpulan dari model-model dan artefak yang merekam keputusan utama yang telah diambil

➤ ***Enabling Techniques***

1. Prinsip dari *Software Design*, juga disebut dengan teknik penyediaan, adalah ide utama berdasarkan pada berbagai pendekatan dan konsep yang berbeda dari *software design*.

2. Macam *Enabling Techniques* sebagai berikut :

- *Abstraction*
- *Coupling and cohesion*
- *Decomposition and modularization*
- *Encapsulation/information hiding*
- *Separation of interface and implementation*
- *Sufficiency, completeness and primitiveness*

➤ ***Abstraction***

- *Abstraction* adalah karakteristik dasar dari sebuah entitas yang membedakan entitas tersebut dari entitas yang lain
- *Abstraction* mendefinisikan batasan dalam pandangan *viewer*
- *Abstraction* bukanlah pembuktian nyata, hanya menunjukkan intisari / pokok dari sesuatu

➤ ***Coupling and cohesion***

Coupling didefinisikan sebagai kekuatan hubungan antara *module*, sementara *cohesion* didefinisikan bagaimana elemen-elemen membuat modul tersebut saling berkaitan.

➤ ***Decomposition modularization***

Pendekomposisian dan pemodularisasian *software* besar menjadi sejumlah *software* independen yang lebih kecil, biasanya dengan tujuan untuk

menempatkan fungsionalitas dan responsibilitas pada komponen yang berbeda.

➤ ***Encapsulation***

Encapsulation adalah menyembunyikan implementasi dari client, sehingga client hanya tergantung pada interface.

➤ ***The number of key issues crosscutting***

■ ***Concurrency***

Bagaimana *software* dapat membedakan proses, *task*, *threads*, *synchronisasi* dan *scheduling*

■ ***Control and handling of events***

Bagaimana sebuah *software* dapat mengatur data dan *flow control*

■ ***Distributions of components***

Bagaimana sebuah *software* dapat didistribusikan dan semua komponen saling berkomunikasi

➤ ***The number of key issues crosscutting***

■ ***Error and Exception handling and Fault tolerance***

Bagaimana sebuah *software* dapat mengenali sebuah *error* dan mengetahui bagaimana cara mengatasinya

■ ***Interaction and presentation***

Bagaimana sebuah *software* dapat berinteraksi dengan *user* dan dapat menampilkan keinginan *user*

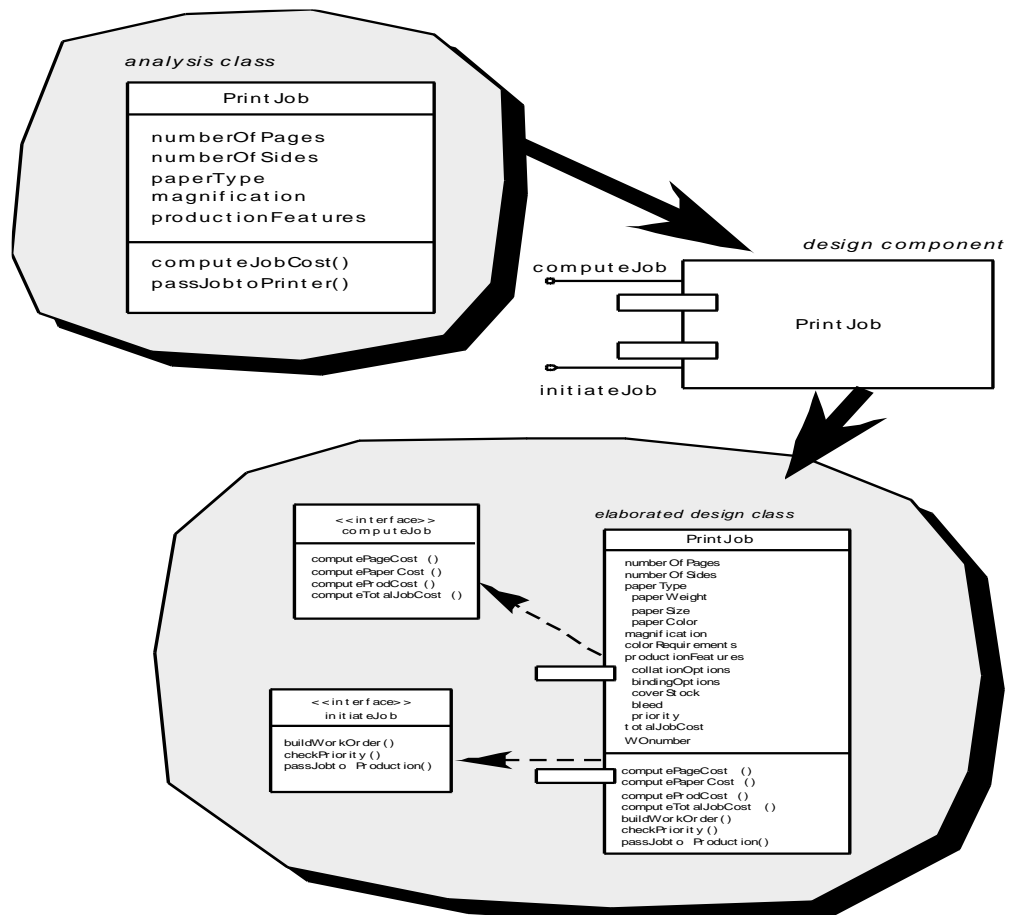
■ ***Data persistence***

Seberapa lama data akan disimpan

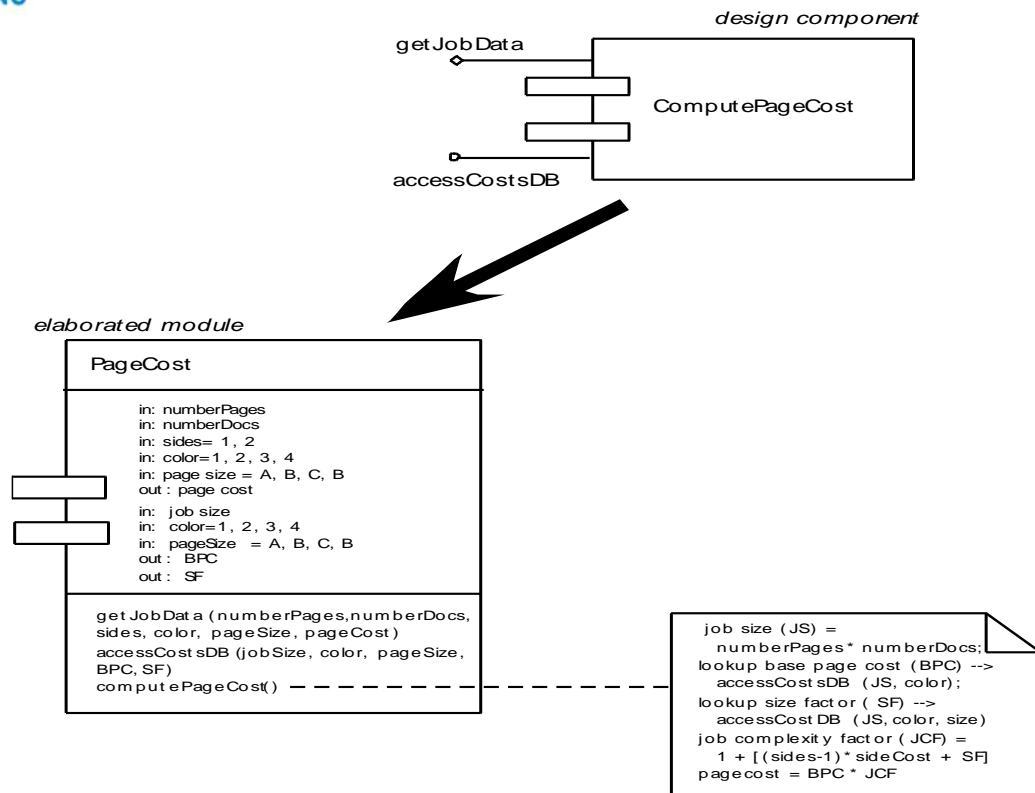
2. Component-Level Design

OMG *Unified Modeling Language Specification* mendefinisikan komponen sebagai bagian yang modular, dapat *dideploy* dan dapat digantikan dari sebuah

sistem yang mengenkapsulasi implementasi dan exposes dari sekumpulan interface. Dalam pandangan *object oriented*, sebuah komponen berisi sekumpulan kelas yang saling berkolaborasi.



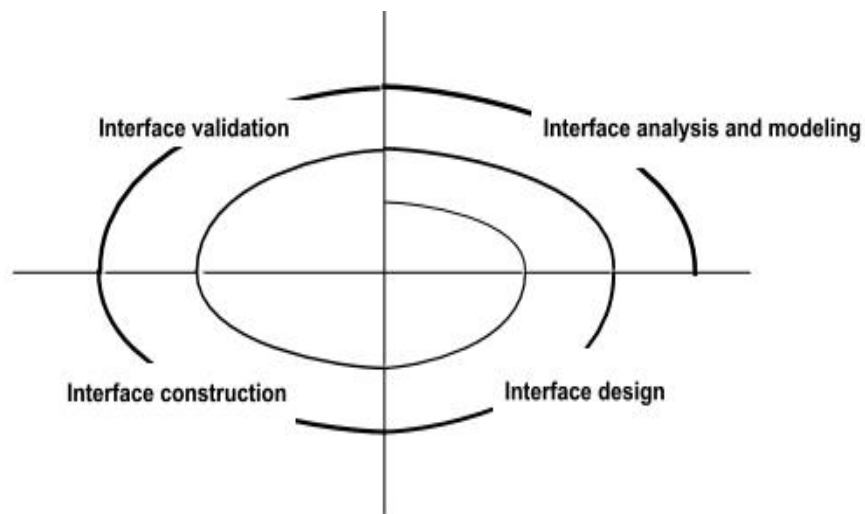
Dalam sudut pandang konvensional, komponen berisi logika pemrosesan, struktur data internal yang dibutuhkan untuk mengimplementasi logika pemrosesan, dan *interface* yang memungkinkan komponen untuk diminta dan data dilewatkan.



3. User Interface Design

Berikut ini adalah beberapa model perancangan user interface:

- Model *user*, sebuah profile dari semua *end user* dari sistem
- Model *design*, sebuah realisasi perancangan terhadap model *user*
- Model mental, gambaran mendari dari user terhadap apa itu *interface*
- Model implementasi, bagaimana *interface* “terlihat dan terasa” dengan dukungan informasi yang menggambarkan sintaks dan semantic *interface*.



4. *Pattern Based Design*

Suatu *design pattern* yang efektif memiliki karakteristik sebagai berikut:

- a. Memecahkan masalah
- b. Membuktikan konsep yang ada
- c. Solusi tidak jelas
- d. Menjelaskan hubungan yang ada
- e. Memiliki komponen manusia yang signifikan

Generative pattern menggambarkan aspek yang penting dan berulang dari sebuah sistem, dan kemudian menyediakan cara untuk membangun aspek tersebut di dalam sistem.

5. *WebApp Design*

Kualitas perancangan dari sebuah aplikasi dapat dilihat dari beberapa faktor berikut:

- **Security:**
 - Tahan terhadap serangan eksternal

- Menolak akses yang tidak terotorisasi
- Memastikan privasi dari user/customer

- ***Availability***

- Ukuran dari persentase waktu dimana aplikasi tersedia bagi user untuk digunakan

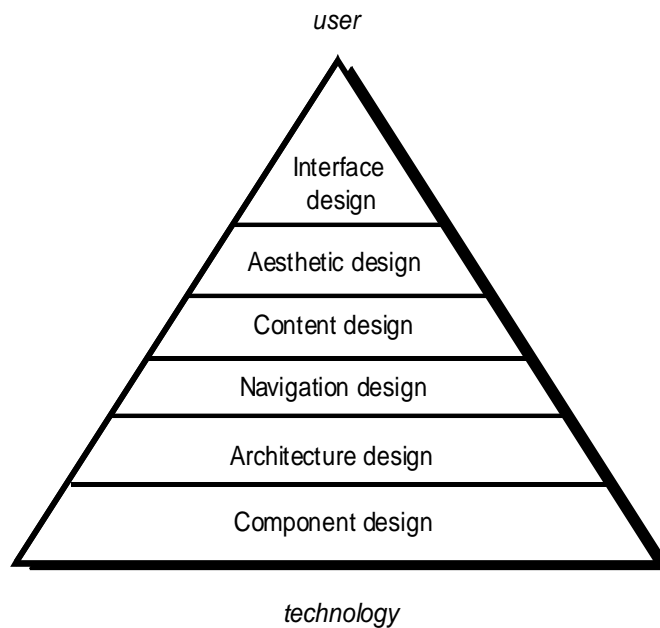
- ***Scalability***

- Dapatkah aplikasi web dan sistem interfacenya handle beragam signifikasi dari user atau volume transaksi

- ***Time to market***

- Waktu yang dibutuhkan untuk distribusi dalam pasar.

Berikut ini adalah piramida perancangan untuk aplikasi *web*:



6. Mobile App Design

Seperti semua alat *computing*, *platform mobile* dibedakan berdasarkan *software* yang diberikan, kombinasi dari sistem operasi (Android, iOS, dll) dan

bagian kecil dari ribuan aplikasi *mobile* yang menyediakan berbagai fungsionalitas.

Studi Kasus

Pada banyak implementasi software engineering, pada umumnya metode yang dibahas sebelumnya digunakan untuk kondisi pembangunan perangkat lunak yang dibangun dari awal, misalnya Anda ingin membangun:

- Aplikasi penjualan di perusahaan yang spesifik ke perusahaan tertentu
- Aplikasi front-end, misalnya front-end untuk aplikasi perbankan
- Aplikasi marketing yang memiliki fitur yang sangat khusus pada perusahaan tertentu

Beberapa pembuatan atau kustomisasi produk yang dikeluarkan oleh vendor tertentu, misalnya Oracle, SAP, atau IBM terdapat mekanisme dan metodologi tersendiri yang dikeluarkan oleh mereka. Metodologi tersebut dapat digabungkan dengan beberapa konsep desain engineering yang saat ini kita pelajari.

DAFTAR PUSTAKA

- Introduction to software quality assurance,,
http://www.youtube.com/watch?v=5_cTi5xB1Yg
- Software reliability ,
http://www.youtube.com/watch?v=ww51aF_qODA
- Lean Six Sigma and IEEE standards for better software engineering,,
<http://www.youtube.com/watch?v=oCkPD5YvWqw>

LECTURE NOTES

Software Engineering

Minggu 5

Software Quality Assurance

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO3 – Mendemonstrasikan penjaminan mutu suatu *software*

Outline Materi (Sub-Topic):

1. *Software Quality*
2. *The Software Quality Dilemma*
3. *Achieving Software Quality*
4. *Review Techniques*
5. *Defect Amplification*
6. *Review Metrics*
7. *Informal Reviews*
8. *Formal Technical Reviews*
9. *Software Reliability*
10. Studi kasus

ISI MATERI

1. *Software Quality*

Apakah yang dimaksud dengan kualitas? Kualitas berarti seberapa jauh suatu produk dapat memenuhi kebutuhan yang ditentukan di awal pengerjaan. Produk yang tidak berkualitas akan menimbulkan masalah, di antaranya:

- Diperlukannya *rework* untuk produk tersebut
- Pertambahan biaya
- Pengaruh kepada reputasi produk dan perusahaan
- Berdampak pada kepuasan pelanggan

Bahkan ada dampak yang lebih berat lagi jika terjadi masalah yang berhubungan dengan hukum, atau pun berkaitan dengan nyawa manusia. Misalnya, suatu produk perangkat lunak yang terdapat pada otomotif, atau kereta api. Jika terjadi kesalahan algoritma yang menyebabkan terjadinya kecelakaan, tentunya ini sangat berbahaya.

Membangun perangkat lunak yang berkualitas tentunya menjadi hal yang sangat penting untuk dilakukan. Terdapat dua kategori kegiatan yaitu: *software quality assurance (SQA)* dan *software quality control (SQC)*. SQA berhubungan dengan memastikan bahwa kegiatan yang berhubungan dengan kualitas perangkat lunak dikerjakan sesuai dengan rencana, sedangkan SQC sendiri merupakan bentuk kegiatan untuk melakukan testing terhadap perangkat lunak yang dibangun.

Sebelum melakukan kegiatan pelaksanaan penjaminan mutu perangkat lunak, diperlukan perencanaan terhadap aktivitas yang perlu dilakukan, di antara nya:

- Kegiatan apa yang perlu dilakukan terhadap kegiatan yang berkenaan terhadap kualitas

- Metode apa yang dipakai untuk melakukan verifikasi, validasi dan testing
- Apa saja metrics yang dipakai di dalam mengukur suatu kualitas perangkat lunak, misalnya:
 - Performance
 - Availability
 - Security
 - Usability
 - Reliability
- Bagaimana membandingkan antara hasil yang didapatkan dengan metrics yang direncanakan di awal
- Alat bantu apa yang cocok digunakan

2. *The Software Quality Dilemma*

Di dalam menentukan suatu derajat atau *metrics* kualitas dari perangkat lunak, juga harus dipertimbangkan dengan penjadwalan dan biaya yang berhubungan dengan pencapaian kualitas tersebut.

Jika kita mengembangkan sebuah *software* yang memiliki kualitas buruk, maka hal tersebut akan menjadi sebuah kegagalan dimana tidak akan ada yang mau membeli *software* tersebut. Di sisi lain, jika kita menghabiskan waktu terlalu lama dengan usaha dan biaya yang sangat besar untuk mengembangkan sebuah *software* yang sempurna, *software* yang dihasilkan akan terlalu mahal sehingga kita tidak dapat memasarkannya. Ini adalah salah satu dilemma terbesar dalam pengembangan *software*. Sehingga dalam *software engineering*, yang menjadi target adalah *software* yang “cukup baik” sehingga tidak akan

ditolak oleh pasar dengan menggunakan sumber daya yang efisien dalam pengembangannya (biaya, waktu, sumber daya manusia, dll). *Software* yang “cukup baik” memberikan fungsi dan *feature* dengan kualitas tinggi yang diinginkan oleh *user*, namun juga memberikan fungsi-fungsi spesifik yang kurang jelas yang masih memiliki *bugs*.

3. *Achieving Software Quality*

Kualitas perangkat lunak didefinisikan sebagai konformansi terhadap kebutuhan fungsional dan kinerja yang dinyatakan secara eksplisit, standar perkembangan yang didokumentasikan secara eksplisit, dan karakteristik implisit yang diharapkan bagi semua perangkat lunak dikembangkan secara profesional. definisi tersebut berfungsi untuk menekankan tiga hal penting, yaitu:

1. Kebutuhan perangkat lunak merupakan fondasi yang melaluinya kualitas diukur.
2. Standar yang telah ditentukan menetapkan serangkaian kriteria pengembangan yang menuntun cara perangkat lunak direkayasa.
3. Ada serangkaian kebutuhan implisit yang sering dicantumkan (misalnya kebutuhan akan kemampuan pemeliharaan yang baik).
4. Kelompok SQA berfungsi sebagai perwakilan inhouse pelanggan, yaitu orang yang akan melakukan SQA harus memperhatikan perangkat lunak dari sudut pandang pelanggan.

Kelompok SQA harus dapat menjawab pertanyaan-pertanyaan dibawah ini untuk memastikan bahwa kualitas perangkat lunak benar-benar terjaga.

- Apakah perangkat lunak cukup memenuhi faktor kualitas
- Sudahkah pengembangan perangkat lunak dilakukan sesuai dengan standar yang telah ditetapkan sebelumnya?
- Sudahkah disiplin teknik dengan tepat memainkan perannya sebagai bagian dari aktivitas SQA?

Aktivitas SQA Jaminan kualitas perangkat lunak terdiri dari berbagai tugas yang berhubungan dengan dua konstituen yang berbeda:

- perekayasa perangkat lunak yang mengerjakan kerja teknis
- kelompok SQA yang bertanggung jawab terhadap perencanaan jaminan kualitas, kesalahan, penyimpanan rekaman, analisis, dan pelaporan.

Tugas kelompok SQA adalah membantu tim rekayasa perangkat lunak dalam pencapaian produk akhir yang berkualitas tinggi.

Aktivitas yang dilakukan (atau difasilitasi) oleh kelompok SQA yang independen: Menyiapkan rencana SQA untuk suatu proyek. Rencana tersebut mengidentifikasi hal-hal berikut:

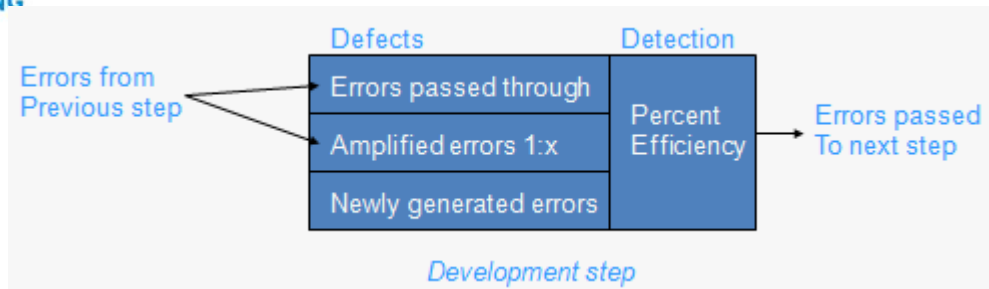
- Evaluasi yang dilakukan
- Audit dan kajian yang dilakukan
- Standar yang dapat diaplikasikan pada proyek
- Prosedur untuk pelaporan & penelusuran kesalahan
- Dokumen yang dihasilkan oleh kelompok SQA
- Jumlah umpan balik yang diberikan pada tim proyek perangkat lunak

4. *Review Techniques*

Kajian perangkat lunak merupakan salah satu aktivitas SQA yang terpenting. Kajian perangkat lunak adalah suatu filter bagi proses rekayasa perangkat lunak, yaitu kajian yg diterapkan pd berbagai titik selama pengembangan PL & berfungsi untuk mencari kesalahan yg kemudian akan dihilangkan. Kajian perangkat lunak berfungsi untuk “memurnikan” produk kerja perangkat lunak yang terjadi sebagai hasil dari analisis, desain, dan pengkodean.

5. *Defect Amplification*

Sebuah *defect amplification* model dapat digunakan untuk mengilustrasikan pembuatan dan pendeteksian error dalam proses perancangan dan programming seperti pada gambar di bawah ini.



6. Review Metrics

Berikut ini adalah *metric* yang dapat digunakan untuk melakukan *review* terhadap *software* berdasarkan *effort* dan total jumlah *error* yang ditemukan:

- $E_{review} = E_p + E_a + E_r$
- $Err_{tot} = Err_{minor} + Err_{major}$

Defect *density* menggambarkan jumlah *error* yang ditemukan per unit kerja dari produk yang direview:

- $Defect\ density = Err_{tot} / WPS$

Keterangan:

E_p : usaha persiapan (orang/jam) yang dibutuhkan untuk melakukan *review* dari sebuah produk kerja sebelum rapat *review* yang sebenarnya dilakukan

E_a : usaha penilaian (orang/jam) yang dikeluarkan selama proses *review* yang sebenarnya

E_r : usaha (orang/jam) yang didedikasikan untuk melakukan koreksi pada *error* yang ditemukan selama proses *review*

WPS : *work product size* merupakan ukuran dari produk kerja yang direview (seperti UML model atau jumlah halaman dari dokumen-dokumen, jumlah baris dari *source code*)

Err_{minor} : jumlah *error* yang ditemukan yang dikategorikan sebagai *error minor*.

Err_{major} : jumlah *error* yang ditemukan yang dikategorikan sebagai *error major*.

7. *Informal Reviews*

Informal review meliputi kegiatan berikut:

- *Desk check* sederhana dari produk kerja *software engineering* yang dilakukan dengan rekan kerja
- *Casual meeting*, meliputi lebih dari 2 orang yang bertujuan untuk melakukan *review* terhadap produk kerja
- Melakukan *review* dari pair programming.

8. *Formal Technical Reviews*

FTR (Formal Technical Review) adalah aktivitas jaminan kualitas perangkat lunak yang dilakukan oleh perekayasa perangkat lunak. Kajian teknik formal atau *walkthrough* adalah pertemuan kajian yang disesuaikan dengan kebutuhan yang terbukti sangat efektif untuk menemukan kesalahan.

Keuntungan utama kajian teknis formal adalah penemuan kesalahan sejak awal sehingga tidak berlanjut ke langkah selanjutnya dalam proses perangkat lunak.

Tujuan FTR adalah

1. Menemukan kesalahan dlm fungsi, logika, / implementasinya dlm berbagai representasi PL;
2. Membuktikan bahwa perangkat lunak di bawah kajian memenuhi syarat;

3. Memastikan bahwa PL disajikan sesuai dgn standar yg sudah ditentukan sebelumnya;
4. Mencapai perangkat lunak yg dikembangkan dengan cara yang seragam;
5. Membuat proyek lebih dapat dikelola.

FTR berfungsi sebagai dasar pelatihan yang memungkinkan perekayasa junior mengamati berbagai pendekatan yang berbeda terhadap analisis perangkat lunak, desain, dan implementasi. FTR juga berfungsi untuk mengembangkan backup dan kontinuitas karena sejumlah orang mengenal baik bagian-bagian perangkat lunak yang tidak mereka ketahui sebelumnya. Masing-masing FTR dilakukan sebagai suatu pertemuan dan akan berhasil hanya bila direncanakan, dikontrol dan dihadirkan dengan tepat. Dalam paragraf berikut, panduan yang mirip dengan walktrough disajikan sebagai kajian teknis formal representatif.

Pertemuan Kajian Tanpa memperhatikan format FTR yang dipilih, setiap pertemuan kajian harus mematuhi batasan-batasan berikut ini:

- Antara 3 & 5 orang (khususnya) harus dilibatkan dalam kajian;
- Persiapan awal harus dilakukan, tetapi waktu yang dibutuhkan harus tidak lebih dari 2 jam dari kerja bagi setiap person
- Durasi pertemuan kajian harus kurang dari 2 jam Pertemuan kajian dihadiri oleh pimpinan kajian, pengkaji, dan prosedur.

Salah satu dari pengkaji berperan sebagai pencatat, yaitu seseorang yang mencatat semua masalah penting yang muncul selama pengkajian. FTR dimulai dengan pengenalan agenda dan pendahuluan dari prosedur. Bila ada masalah kesalahan ditemukan akan dicatat. Pada akhir kajian, semua peserta FTR yang hadir harus memutuskan apakah akan

1. Menerima produk kerja tanpa modifikasi lebih lanjut,
2. Menolak produk kerja sehubungan dengan kesalahan yang ada (sekali dbetulkan, kajiann lain harus dilakukan), atau

3. Menerima produk kerja secara sementara (kesalahan minor telah terjadi & harus dikoreksi, tetapi kajian tambahan akan diperlukan). Keputusan kemudian dibuat.

Semua peserta FTR melengkapinya dengan tanda tangan yang menunjukkan partisipasi mereka dalam kajian serta persetujuan mereka terhadap pertemuan tim kajian. Pelaporan Kajian dan Penyimpanan Rekaman Selama FTR, seorang pengkaji (pencatat) secara aktif mencatat semua masalah yang sudah dimunculkan, yang kemudian dirangkum pada akhir pertemuan sehingga dihasilkan daftar masalah kajian. Sebagai tambahan, laporan rangkuman kajian yang sederhana telah diselesaikan di mana rangkuman kajian merupakan jawaban dari tiga pertanyaan berikut:

1. Apa yang dikaji?
2. Siapa yang melakukan?
3. penemuan apa yang dihasilkan dan apa kesimpulannya?

Daftar masalah kajian mempunyai dua tujuan:

1. Mengidentifikasi area masalah pada produk,
2. Daftar item kegiatan yang menjadi petunjuk bagi prosedur saat koreksi dilakukan. Daftar masalah biasanya dilampirkan pada laporan.

Pedoman Kajian Pedoman untuk melakukan kajian teknis formal harus dilakukan sebelumnya, didistribusikan kepada semua pengkaji, disetujui, dan kemudian dilaksanakan. Kajian yang tidak terkontrol sering dapat menjadi lebih buruk daripada bila tidak ada kajian sama sekali. Berikut ini serangkaian pedoman minimum untuk kajian teknis formal:

1. Kajian produk, bukan produser.
2. Menetapkan agenda dan menjaganya.
3. Membatasi perdebatan dan bantahan.

4. Menetapkan area masalah, tetapi tidak tergoda untuk menyelesaikannya setiap masalah yang dicatat.
5. Mengambil catatan tertulis.
6. Membatasi jumlah peserta dan mewajibkan persiapan awal.
7. Mengembangkan daftar bagi masing-masing produk kerja yang akan dikaji.
8. Mengalokasikan sumber-sumber daya dan jadwal waktu untuk FTR.
9. Melakukan pelatihan bagi semua pengkaji.
10. Mengkaji kajian awal Anda.

9. *Software Reliability*

Reliabilitas perangkat lunak, tidak seperti faktor kualitas yang lain, dapat diukur, diarahkan, dan diestimasi dengan menggunakan data pengembangan historis. Reliabilitas perangkat lunak didefinisikan dalam bentuk statistik sebagai “kemungkinan operasi program komputer bebas kegagalan di dalam suatu lingkungan tertentu dan waktu tertentu”. Kapan saja reliabilitas perangkat lunak dibicarakan, selalu muncul pertanyaan yang sangat penting : Apa yang dimaksudkan dengan bentuk “kegagalan?” dalam konteks dan banyak diskusi mengenai kualitas dan reliabilitas perangkat lunak, kegagalannya adalah ketidaksesuaian dengan kebutuhan perangkat lunak. Kegagalan hanya akan mengganggu atau bahkan merupakan bencana. Satu kegagalan dapat diperbaiki dalam beberapa detik sementara kesalahan yang lain mungkin membutuhkan waktu pembetulan berminggu-minggu atau bahkan berbulan-bulan. Pembetulan satu kegagalan kenyataannya dapat menghasilkan kesalahan lain yang baru yang mungkin akan membawa lagi kesalahan yang lain lagi. Pengukuran Reliabilitas dan Availabilitas Kerja awal dalam reliabilitas perangkat lunak berusaha mengekstrapolasi matematika teori reliabilitas perangkat keras. Sebagian besar model reliabilitas yang berhubungan dengan perangkat keras didasarkan pada kegagalan sehubungan dengan keusangan (*wear*), bukan kesalahan karena cacat desain. Dalam perangkat keras, kegagalan sehubungan dengan keusangan fisik (misalnya

pengaruh suhu, korosi, kejutan) lebih banyak terjadi daripada kegagalan karena isu. Akan tetapi, yang terjadi pada perangkat lunak adalah hal yang sebaliknya. Kenyataannya, semua kegagalan perangkat lunak dapat ditelusuri ke dalam desain atau masalah implementasi; keusangan tidak tercakup. Masih ada perdebatan yang terjadi di seputar hubungan antara konsep kunci dalam reliabilitas perangkat keras dan kemampuan aplikasinya terhadap perangkat lunak. Meskipun ada hubungan yang tidak dapat dibantah, namun sangat penting untuk memprtimbangkan beberapa konsep sederhana yang berlaku untuk kedua sistem elemen tersebut. Bila kita andaikan suatu sistem yang berbasis komputer, pengukuran reliabilitas secara sederhana adalah berupa *mean time between failure* (MTBF), dimana: $MTBF = MTTF + MTTR$ (Akronim MTTF adalah *mean time to failure* dan MTR berarti *mean time to repair*.) Banyak peneliti berpendapat bahwa MTBF merupakan pengukuran yang jauh lebih berguna daripada pengukuran cacat/KLOC. Secara sederhana dapat dikatakan bahwa seorang pemakai akhir lebih memperhatikan kegagalan, bukan jumlah cacat. Karena masing-masing cacat yang ada pada sebuah program tidak memiliki tingkat kegagalan yang sama, maka penghitungan cacat total hanya memberikan sedikit indikasi tentang reliabilitas sistem. Contohnya adalah sebuah program yang telah beroperasi selama 14 bulan. Banyak cacat mungkin tidak terdeteksi dalam jumlah waktu yang lama sampai pada akhirnya cacat itu ditemukan. MTBF dari cacat yang tidak jelas seperti itu dapat berlangsung sampai 50, bahkan 100 tahun. Cacat yang lain, yang juga belum ditemukan, dapat memiliki tingkat kegagalan 18 atau 24 bulan. Meskipun setiap kategori pertama cacat (yang memiliki MTBF panjang) dihilangkan, pengaruhnya pada reliabilitas perangkat lunak tidak dapat diabaikan. Availabilitas perangkat lunak adalah kemungkinan sebuah program beroperasi sesuai dengan kebutuhan pada suatu titik yang diberikan pada suatu waktu dan didefinisikan sebagai: $Availabilitas = \frac{MTTF}{(MTTF + MTTR)} \times 100\%$ Pengukuran reliabilitas MTBF sama sensitifnya dengan MTTF dan MTTR. Pengukuran availabilitas jauh lebih sensitif daripada MTTR, yang merupakan pengukuran tidak langsung terhadap kemampuan pemeliharaan perangkat lunak

Studi kasus

Di dalam proyek pengembangan perangkat lunak, misalnya Anda akan membangun suatu aplikasi HRD berbasis web. Hal-hal yang perlu diperhatikan di dalam perencanaan kualitas adalah:

- Penentuan target yang ingin dicapai. Biasanya hal ini berhubungan dengan non functional requirement atau matrix, misalnya:
 - Availability, seberapa besar waktu suatu system untuk tersedia, misalnya 95% availability per bulan
 - Performance, seberapa cepat system ini dapat memproses transaksi per hari, biasanya ditentukan oleh TPS (Transaction Per Second)
 - Security, jenis security apa saja yang diperlukan
- Metode Pengujian yang akan dijalankan, misalnya:
 - Sistem Integration Test (SIT)
 - Performance Test
 - Security Test
 - User Acceptance Test (UAT)
- Environment apa yang akan digunakan untuk testing, apakah akan disediakan mesin khusus untuk testing atau pun menggunakan mesin development yang ada
- Alat bantu yang digunakan untuk testing, apakah perlu sewa atau beli, atau kan menyewa vendor lain untuk melakukan test
- Bagaimana penerimaan hasil test dibandingkan dengan perencanaan

DAFTAR PUSTAKA

- Software engineering : a practitioners approach :
Chapter 19/Pages 412 & chapter 20/Pages 431
- Introduction to software quality assurance,,
http://www.youtube.com/watch?v=5_cTi5xBLYg
- Software reliability:
http://www.youtube.com/watch?v=vv51aF_qODA
- Lean Six Sigma and IEEE standards for better software engineering,,
<http://www.youtube.com/watch?v=oCkPD5YvWqw>
-

LECTURE NOTES

Software Engineering

Minggu 6

Application Testing and Security Engineering

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO3 – Mendemonstrasikan penjaminan mutu suatu *software*

Outline Materi (Sub-Topic):

1. *Testing Conventional Applications*
2. *Testing Object Oriented Applications*
3. *Testing Web Applications*
4. *Testing Mobile Applications*
5. *Security Engineering*
6. Studi Kasus

ISI MATERI

1. *Testing Conventional Applications*

Proses testing ini dapat dilakukan secara manual dengan membuat perencanaan *dan test case* sebelum dilakukan testingnya. Kapan metode ini digunakan, tentunya disesuaikan dengan kebutuhan dan karakteristik dari system.

Misalnya, Anda membuat aplikasi HRD berbasis desktop. Pengujian konvensional ini dapat dilakukan dengan membuat test case yang berisi:

- Tahap testing yang dibutuhkan
- Aktivitas test untuk setiap fitur
- Hasil yang diharapkan
- Status, apakah sudah lulus tes atau belum
- Action plan, yaitu jika terjadi error, maka aktivitas apa yang perlu dilakukan.

Contoh tabel di dalam pembuatan suatu *test case*

No	Modul	Tahapan Test	Hasil yang diharapkan	Status	Action Plan
1	Cari data pegawai	1. Login ke system 2. Masuk ke menu Search 3. Masukkan NIM pegawai	Sistem dapat menampilkan data pegawai yang diinginkan		

Pada proses pengujian konvensional, berikut adalah beberapa hal yang perlu diperhatikan:

- *Operability*, dapat dioperasikan dengan baik dan bersih
- *Observability*, hasil dari setiap test case siap untuk diobservasi
- *Controllability*, tingkatan dimana pengujian dapat diotomasi dan dioptimasi
- *Decomposability*, pengujian dapat ditargetkan
- *Simplicity*, mengurangi kompleksitas arsitektur dan logika untuk menyederhanakan pengujian.
- *Stability*, hanya sedikit perubahan yang diminta pada saat proses pengujian berlangsung
- *Understandability*, rancangan sistem yang dapat dipahami

Apa yang disebut sebagai pengujian yang “baik”?

- Sebuah pengujian baik memiliki kemungkinan yang tinggi dalam menemukan error
- Pengujian yang baik tidak redundan
- Pengujian yang baik harus dari “keturunan terbaik”
- Pengujian yang baik tidak boleh terlalu sederhana atau terlalu kompleks.

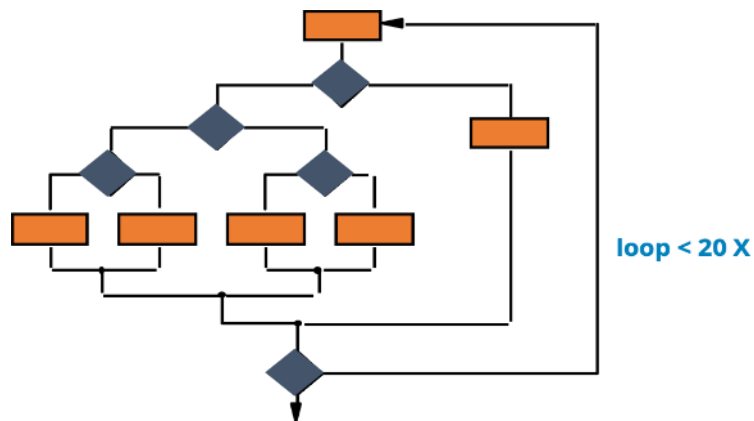
Merancang *test case*:

- Tujuan: untuk menemukan error
- *Criteria*: dengan cara yang lengkap
- Batasan: dengan usaha dan waktu yang minimal

- *Testing konvensional* dibagi menjadi dua jenis testing:

a. Exhaustive Testing

Testing yang mengeksekusi semua kemungkinan path dari logika atau algoritma yang ada di dalam sistem. Pada sebuah contoh, terdapat kemungkinan 10^{14} path yang dapat diuji, sehingga akan membutuhkan waktu 3.170 tahun untuk melakukan pengujian dengan asumsi satu path dieksekusi selama satu mili second.



Sedangkan berdasarkan metode pengujian, terdapat dua kategori testing:

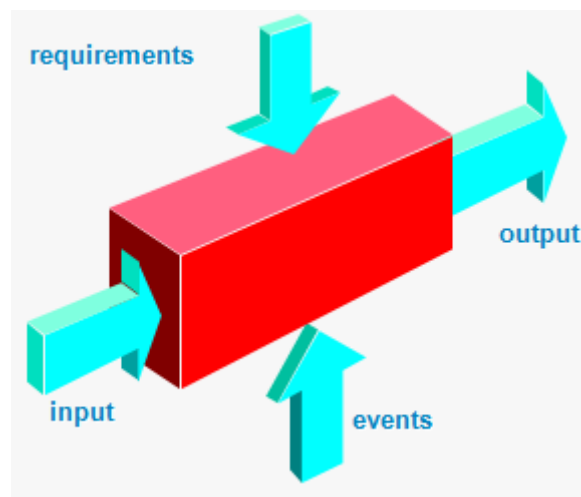
a. Metode *White box*

Tujuan utama dari white box testing adalah untuk memastikan bahwa semua *statement* dan kondisi telah dieksekusi minimal satu kali. Sehingga untuk melakukan *white box testing*, diperlukan *source code* dan dokumentasi yang terkait dengan aplikasi. *Tester* yang biasa melakukan *white box* testing adalah developer/programmer yang mengembangkan sistem.

Pada *white box testing*, biasa dilakukan *basis path testing* dengan menghitung *cyclomatic complexity*.

b. Metode *Black Box*

Metode *black box* merupakan pengujian tanpa melihat isi dari aplikasi. Pengujian dilakukan dengan memberikan input, melakukan sekumpulan *events* dan memberikan requirements sesuai dengan instruksi yang diberikan, kemudian mengecek output yang dihasilkan.



2. Testing Object Oriented Applications

Agar dapat menguji aplikasi yang berorientasi obyek dengan layak, tiga hal berikut harus dilakukan:

- Definisi dari pengujian harus diperluas untuk meliputi teknik pencarian error yang dapat diaplikasikan dalam model perancangan dan analisis *object oriented*.
- Strategi dari pengujian integrasi dan uni harus diubah secara signifikan
- Rancangan dari *use case* harus memfasilitasi karakteristik yang unik dari *software object oriented*.

➤ Terdapat tiga strategi pengujian *object oriented* yang dapat dilakukan:

a. Unit testing

Konsep dari unit berubah. Unit terkecil yang dapat diuji adalah kelas yang terenkapsulasi. Satu operasi *single* tidak dapat lagi dioperasikan dalam isolasi, tetapi sebagai bagian dari kelas.

b. Integration testing

- Pengujian berdasarkan thread mengintegrasikan sekumpulan dari kelas yang dibutuhkan untuk memberikan respon untuk satu *input/event* dari sistem.
- Pengujian berdasarkan “penggunaan” memulai konstruksi dari sistem dengan menguji kelas-kelas yang menggunakan paling sedikit kelas *server*. Setelah kelas yang independen tersebut diuji, lapisan kelas berikutnya disebut sebagai kelas dependen.
- Pengujian *cluster* mendefinisikan cluster dari kelas kolaborasi yang dilakukan dengan merancang *test case* yang berusaha untuk mengungkapkan *error* dalam kolaborasi.

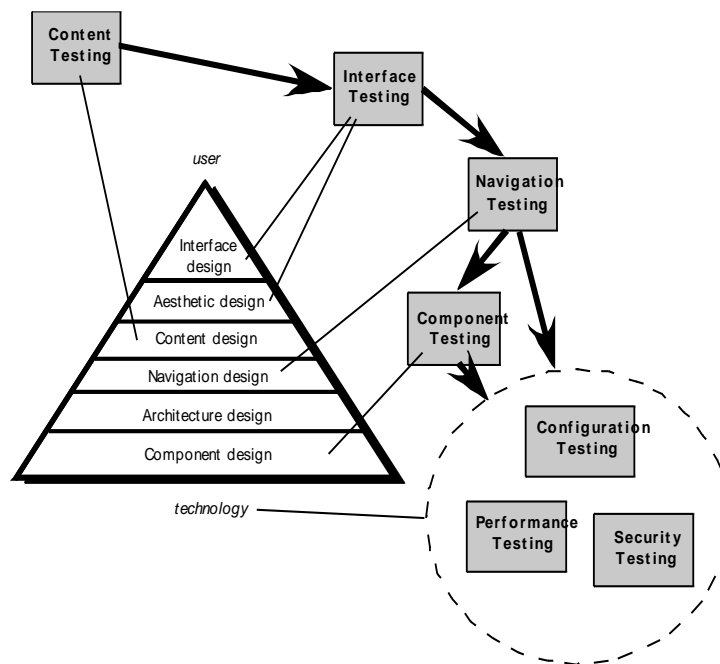
c. Validation testing

Detail dari koneksi kelas yang menghilang.

3. Testing Web Applications

Pada pengujian aplikasi *web*, terdapat beberapa dimensi kualitas yang harus diperhatikan:

- a. *Content*, dievaluasi pada kedua level *semantic* dan *syntactic*.
- b. *Function*, diuji kebenaran, stabilitasnya, dan kesesuaian general hingga standar implementasi yang layak.
- c. *Structure*, dinilai untuk memastikan bahwa fungsi dan konten dari aplikasi web disampaikan dengan benar, dapat diperluas, dapat didukung sebagai konten baru atau penambahan fungsionalitas.
- d. *Usability*, diuji untuk memastikan bahwa setiap kategori dari user didukung oleh *interface*, serta dapat mempelajari dan mengaplikasikan semua navigasi yang dibutuhkan.
- e. *Navigability*, diuji untuk memastikan semua *semantic* dan sintaks navigasi dilaksanakan untuk menemukan *error* navigasi.
- f. *Performance*, pengujian *performance* dilakukan dalam berbagai kondisi operasi, konfigurasi, dan loading untuk memastikan bahwa sistem *responsive* akan interaksi user.
- g. *Compatibility*, diuji dengan menjalankan aplikasi *web* dalam berbagai konfigurasi host yang berbeda untuk *client* dan *server*.
- h. *Interoperability*, diuji untuk memastikan bahwa aplikasi *web* memiliki interface yang layak sesuai dengan *interface* aplikasi atau/dan *database* lainnya.
- i. *Security*, diuji untuk menilai potensi kerentanan dan berusaha untuk mengeksploitasi setiap kerentanan yang ditemukan.



4. *Testing Mobile Applications*

Kriteria lingkungan dan *tool* pengujian aplikasi *mobile*:

- Identifikasi object
- *Security*
- *Devices*
- *Functionality*
- *Emulators and plug ins*
- *connectivity*

5. *Security Engineering*

Bagian penting dalam membangun sistem yang aman adalah mengantisipasi kondisi atau ancaman yang mungkin digunakan untuk merusak sistem atau mengubah sistem sehingga tidak dapat diakses oleh user yang berwenang. Proses ini disebut analisis ancaman.

6. Studi kasus

Beberapa perusahaan memiliki suatu departmen khusus yang dinamakan sebagai QA/QC departmen, yaitu *Quality Assurance* dan *Quality Control*. Tugasnya untuk memastikan semua sistem yang dibangun sudah dilakukan test sehingga layak untuk dilakukan *deployment* ke *production system*, yaitu lingkungan yang terdapat aplikasi yang digunakan untuk kegiatan nyata.

Tahapan awal yang dilakukan adalah membuat perencanaan dari testing, yang mencakup:

- Tujuan testing
- Aplikasi atau system apa yang akan di-test
- Metode testing yang dilakukan
- Jadwal testing
- Penerimaan dari testing tersebut

DAFTAR PUSTAKA

- Software engineering : a practitioners approach : Chapter 23/Pages 496, Chapter 24/Pages 523, Chapter 25/Pages 540, Chapter 26/Pages 567, and Chapter 27/Pages 584

LECTURE NOTES

Software Engineering

Minggu 7

Software Configuration Management

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO 4 – Menganalisa proyek *management software*

Outline Materi (Sub-Topic) :

1. *The Software Configuration*
2. *SCM Repository*
3. *The SCM Process*
4. *SCM for Web Engineering*
5. Studi Kasus

ISI MATERI

1. *The Software Configuration*

Software configuration merupakan rangkaian komponen yang saling berhubungan dengan software yang kita bangun. Terdapat istilah configuration items yaitu software dan elemen konfigurasinya, contoh:

- Program atau source code itu sendiri
- Dokumentasi
- Data

Jadi *Software* tidak hanya merupakan program dari *computer* dalam bentuk *source code*, tapi meliputi semua dokumen terkait dengan pengembangan *software* tersebut dan *database* yang digunakan.

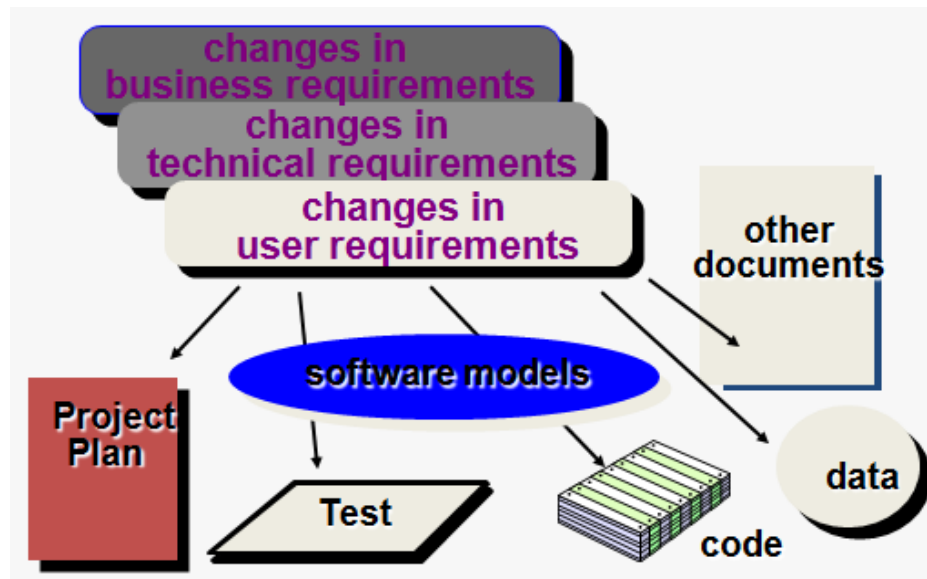
Contoh:

Anda membangun aplikasi penjualan berbasis web pada perusahaan Anda, maka yang termasuk komponen dari *software configuration* adalah:

- Source code dari software yang memiliki versi-versi tertentu, misalnya versi 1.0, versi 2.0 dan seterusnya
- Dokumentasi yang dihasilkan dan dibutuhkan, misalnya ada beberapa dokumen dengan versi-versi tertentu
- Data yang digunakan, dapat berupa *database* atau pun data mentah, misalnya data dalam bentuk *spreadsheet* atau bentuk lainnya.

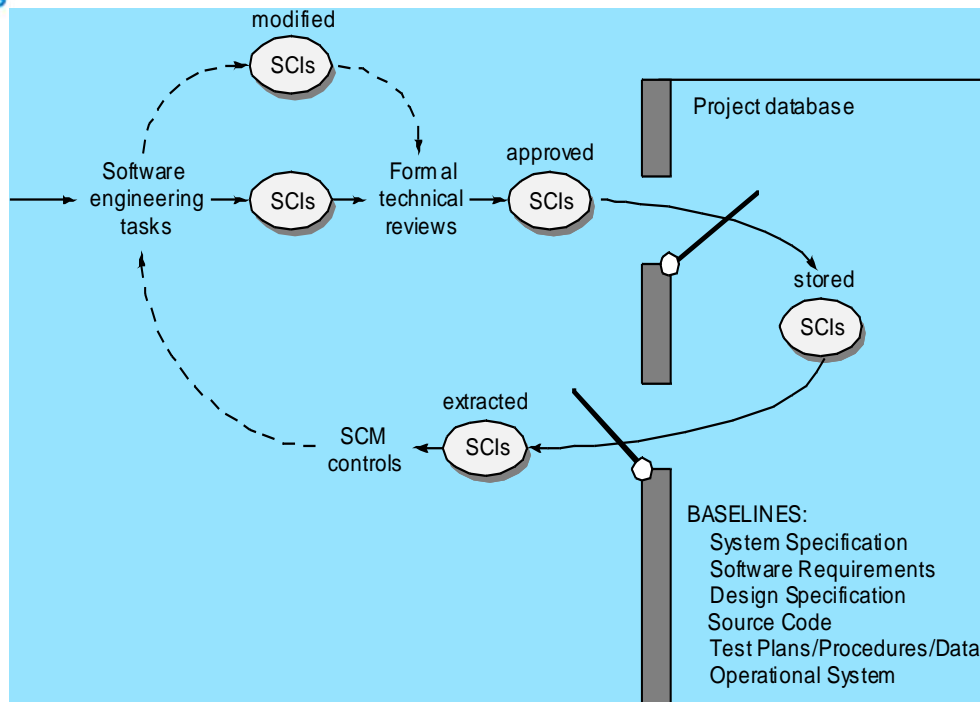
Software konfigurasi bertujuan untuk mengakomodasi perubahan yang mungkin terjadi, perubahan dapat terjadi dalam:

- *Requirement* bisnis, proses bisnis yang berubah sehingga menuntut penyesuaian dari program yang dibuat.
- *Requirement* teknis, perubahan dari kebutuhan teknis seperti perubahan platform menyebabkan penyesuaian program.
- *Requirement user*, perubahan kebutuhan dari user secara specific.



Pada *software configuration management* dilakukan dengan mengubah baseline yang ada. Baseline adalah spesifikasi atau produk yang secara formal telah direview dan disetujui, sehingga menjadi dasar dari pengembangan selanjutnya. Baseline hanya dapat diubah setelah mendapatkan persetujuan secara formal melalui prosedur control perubahan.

Baseline juga disebut sebagai milestone dalam proses pengembangan *software* yang ditandai dengan mengirimkan satu atau lebih item *software* konfigurasi.



2. *SCM Repository*

Software Configuration Management repository adalah sekumpulan mekanisme dan struktur data yang memungkinkan tim software untuk mengatur perubahan dalam cara yang *effective*. *SCM repository* menjalankan fungsi-fungsi berikut:

- *Data integrity*
- *Information sharing*
- *Tool integration*
- *Data integration*
- *Methodology enforcement*
- *Document standardization*

Repository feature merupakan *feature-feature* yang dapat digunakan untuk melakukan fungsi dari pengaturan konfigurasi software. Berikut adalah beberapa *feature* dari *SCM repository*:

a. *Versioning*

Pada *feature versioning*, dilakukan penyimpanan semua versi *software* untuk memungkinkan pengaturan secara efektif untuk produk yang akan direlease dan untuk mengijinkan *developer* untuk kembali ke versi sebelumnya.

b. *Dependency tracking and change management*

Repository mengatur hubungan yang sangat beragam antar data elemet yang disimpan di dalamnya.

c. *Requirement tracing*

Menyediakan kemampuan untuk melakukan pelacakan semua perancangan dan konstruksi komponen yang dihasilkan dari proses spesifikasi *requirement*.

d. *Configuration management*

Menjaga *track* dari sekumpulan konfigurasi yang merepresentasikan milestone dari proyek tertentu.

e. *Audit trails*

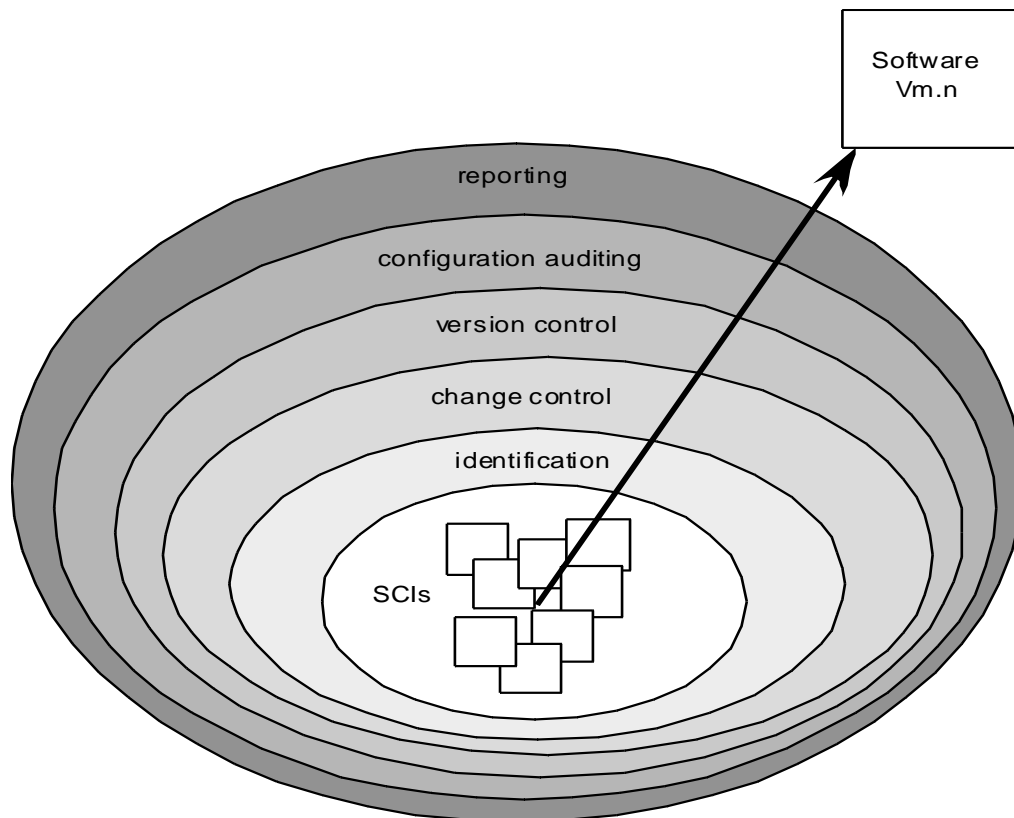
Menetapkan informasi tambahan mengenai kapan, bagaimana dan oleh siapa perubahan dibuat.

Beberapa elemen dari *Software Configuration Management*:

- ***Component element***, sekumpulan dari *tool* yang dipasarkan dengan manajemen sistem data (seperti *database*) yang memungkinkan akses ke data tersebut dan melakukan pengaturan terhadap setiap item konfigurasi *software*.
- ***Prosess element***, merupakan kumpulan dari prosedur dan tugas yang mendefinisikan pendekatan yang efektif untuk melakukan manajemen perubahan melibatkan semua yang terlibat dalam manajemen.

- **Construction elements**, kumpulan dari *tool-tool* yang mengautomasi pembuatan *software* dengan memastikan validasi komponen yang layak.
- **Human element**, untuk mengimplementasikan SCM yang efektif, tim *software* harus menggunakan sekumpulan *feature tool* dan proses.

3. The SCM Process



Urutan proses dalam SCM:

- Dimulai dari pengenalan kebutuhan akan perubahan
- Permintaan akan perubahan dari user
- Evaluasi dari *developer*
- Laporan perubahan dibuat
- Memutuskan otoritas *control* dari perubahan

- f. Permintaan dimasukkan kedalam daftar antrian untuk dieksekusi (jika diterima)
- g. Menugaskan individu pada setiap obyek konfigurasi
- h. “*check out*” obyek konfigurasi
- i. Melakukan perubahan
- j. Melakukan review/audit terhadap perubahan
- k. “*check in*” konfigurasi item yang telah diubah
- l. Menentukan baseline untuk pengujian
- m. Menjalankan pengujian kualitas dan aktifitas pengujian
- n. Mempromosikan perubahan untuk penyertaan di revisi berikutnya
- o. Membangun kembali versi *software* yang benar
- p. Melakukan review/audit terhadap perubahan dari semua item konfigurasi
- q. Mengikutkan perubahan di versi terbaru
- r. Melakukan distribusi versi baru

4. *SCM for Web Engineering*

Dalam melakukan pengaturan konfigurasi untuk aplikasi *mobile* atau *web*, beberapa faktor yang harus diperhatikan:

a. *Content*

Sebuah aplikasi web berisi berbagai content seperti *text*, grafik, *applet*, *script*, *audio/video*, *form*, *element* dari *page*, *table*, dan data

streaming. Tantangan dalam mengorganisasikan kumpulan konten menjadi kumpulan konfigurasi yang rasional dan kemudian menentukan mekanisme *control* konfigurasi yang layak.

b. People

Faktor manusia perlu diperhatikan karena pengembangan aplikasi web memiliki persentase yang *signifikan* untuk dilakukan dengan cara ad hoc, dimana setiap orang dapat membuat content dalam web tersebut.

c. Scalability

Dengan pertumbuhan ukuran dan kompleksitas, perubahan kecil dapat menyebabkan efek yang dapat menyebabkan masalah. Oleh karena itu, mekanisme konfigurasi control harus dapat diarahkan secara proposional dan *scalable*.

d. Politik

Siapa yang memiliki *web application*? Siapa yang bertanggung jawab dalam akurasi informasi yang diberikan di *web*? Siapa yang memastikan proses *quality control*? Siapa yang bertanggung jawab membuat perubahan?

5. Studi kasus

Praktik software configuration yang efektif pada perusahaan adalah penggunaan aplikasi manajemen pengembangan aplikasi, yang di dalamnya sudah terdapat version untuk melakukan manajemen konfigurasi ini. Terdapat beberapa fitur yang sangat berguna, misalnya:

- *Version control*
- *Access control* dari konfigurasi yang ada

- Manajemen untuk *source code*
- Fitur kolaborasi antar team di dalam pengelolaan *source code* tersebut

Penggunaan alat bantu atau tools untuk software configuration management ini sangat membantu para stakeholder untuk melakukan manajemen source code dan fungsi lainnya, dibandingkan dengan melakukan proses konfigurasi secara manual.

DAFTAR PUSTAKA

- Software configuration Management,
<http://www.sei.cmu.edu/reports/87cm004.pdf>
- Software engineering : a practitioners approach

LECTURE NOTES

Software Engineering

Minggu 8

Software Project Management And Software Metrics

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO 4 – Menganalisa proyek *management software*

Outline Materi (Sub-Topic) :

1. *Team Coordination & Communication*
2. *Problem Decomposition*
3. *A Framework for Product Metrics*
4. *Metrics for the Requirements Model*
5. *Metrics for the Design Model*
6. *Design Metrics for WebApps*
7. *Code Metrics*
8. *Metrics for Testing*
9. *Maintenance Metrics*
10. *Metrics in the Process and Project Domain*
11. *Software Measurement*
12. *Metrics for Software Quality*
13. Studi kasus

ISI MATERI

1. *Team Coordination & Communication*

Koordinasi dan komunikasi sangat diperlukan di dalam proyek pembangunan software. Seorang manajer proyek ataupun software engineer tentunya harus melakukan koordinasi dan komunikasi baik secara internal atau pun eksternal.

Berdasarkan sifat dan pendekatannya koordinasi dan komunikasi tim dibagi menjadi:

- *Formal, impersonal approaches*

Komunikasi ini bersifat formal dan dapat digunakan untuk dokumen-dokumen yang umumnya bersifat baseline dan berdampak pada target-target proyek.

Meliputi *document software engineering* dan produk kerja (termasuk *source code*), memo teknis, milestone proyek, jadwal dan *project control tools*, permintaan perubahan, laporan penelusuran error, dan *data repository*.

- *Formal, interpersonal procedures*

Komunikasi formal ini digunakan untuk hal-hal yang sifatnya koordinasi antar tim, tetapi nanti hasilnya dapat dibuatkan formal sebagai landasan untuk target-target proyek seterusnya.

Fokus pada kegiatan penjaminan kualitas yang diterapkan dalam produk kerja *software engineering*. Hal ini meliputi status *review meeting*, perancangan dan inspeksi kode.

- *Informal, interpersonal procedures*

Komunikasi ini bersifat informal sehingga cocok juga untuk hal-hal yang bukan kegiatan formal, misalnya meeting kelompok untuk diseminasi informasi dan pemecahan masalah. *Team building* juga menggunakan komunikasi ini.

- *Electronic communication*

Meliputi komunikasi dalam bentuk e-mail, *electronic bulletin board*, atau *video conference*.

- *Interpersonal networking*

Meliputi diskusi informal dengan anggota tim dan anggota di luar tim pengembang yang memiliki pengalaman yang dapat membantu anggota tim.

2. Problem Decomposition

Dekomposisi masalah dikenal juga dengan istilah partisi atau elaborasi masalah. Satu cakupan berarti:

- Didekomposisikan menjadi fungsi-fungsi
- Didekomposisikan menjadi obyek data yang dapat dilihat oleh user, atau
- Didekomposisikan menjadi sekumpulan kelas masalah

Proses dekomposisi dilanjutkan hingga semua fungsi atau masalah dapat didefinisikan. Ketika sebuah kerangka proses telah ditentukan, maka karakteristik dari proyek harus dipertimbangkan.

3. A Framework for Product Metrics

Sebuah ukuran menentukan indikasi kuantitatif dari tingkatan, jumlah, dimensi, kapasitas, atau ukuran dari suatu attribute proses atau produk. Berdasarkan glossary dari IEEE, *metric* didefinisikan sebagai ukuran

kuantitatif mengenai tingkatan dari sebuah sistem, komponen, atau proses yang dimiliki dalam atribut. Sebuah indikator adalah sebuah *metric* atau kombinasi *metric* yang menyediakan wawasan mengenai proses *software*, proyek *software* atau produk itu sendiri.

Karakteristik dari *attribute metric*:

- Sederhana dan dapat dihitung

Sebuah *metric* seharusnya mudah dipelajari mengenai bagaimana *metric* tersebut diturunkan. Perhitungan *metric* seharusnya dapat dilakukan tanpa menyita terlalu banyak waktu dan usaha.

- Secara empiris dan intuitif *persuasive*.

Metrik harus dapat memenuhi intuisi *engineer* mengenai atribut produk yang dipertimbangkan

- Konsisten dan obyektif

Metrik harus selalu memberikan hasil yang jelas dan tidak ambigu, serta bersifat obyektif.

- Konsisten dalam penggunaan unit dan dimensi

Perhitungan matematika dari *metric* harus menggunakan ukuran yang tidak memberikan hasil yang aneh.

- Tidak bergantung pada bahasa pemrograman

Metric harus berdasarkan pada model analisis, model perancangan atau struktur program, bukan berdasarkan bahasa pemrograman.

- Mekanisme yang efektif untuk mendapatkan *feedback* kualitas

Metric harus menyediakan informasi yang dapat membantu dalam menghasilkan produk akhir yang berkualitas tinggi.

4. Metrics for the Requirements Model

a. *Metric* berdasarkan fungsi

Menggunakan *function point* sebagai *factor* normalisasi atau sebagai ukuran dari besar kecilnya suatu spesifikasi. *Function point* pertama kali diusulkan oleh Albrecht yang digunakan secara efektif sebagai alat untuk mengukur fungsionalitas sistem yang diberikan.

b. *Metric* berdasarkan spesifikasi

Digunakan sebagai indikasi terhadap kualitas dengan mengukur jumlah dari requirement berdasarkan tipenya.

5. Metrics for the Design Model

Terdapat 9 karakteristik dari perancangan berorientasi obyek:

- a. Ukuran, ukuran didefinisikan melalui 4 sudut pandang: populasi, jumlah, panjang, dan fungsionalitas
- b. Kompleksitas, bagaimana kelas-kelas dalam perancangan berorientasi obyek dihubungkan satu dengan yang lain.
- c. *Coupling*, hubungan fisik antara elemen-elemen perancangan berorientasi obyek.
- d. *Sufficiency*, ukuran dimana abstraksi memiliki *feature-feature* yang dibutuhkan, atau tingkatan dimana komponen design memiliki *feature-feature* dalam abstraksinya, dalam sudut pandang aplikasi saat ini.
- e. Kelengkapan, implikasi secara tidak langsung mengenai tingkat abstraksi atau komponen perancangan yang dapat digunakan kembali
- f. Kohesi, ukuran dimana semua operasi dalam sebuah kelas bekerja bersama untuk mencapai satu tujuan yang telah didefinisikan dengan baik.

- g. Primitiveness, dapat diterapkan pada kelas maupun operasi yang merupakan tingkatan atomic sebuah operasi atau kelas.
- h. Persamaan, ukuran dimana dua atau lebih kelas dianggap mirip dari segi struktur, fungsi, behavior, atau tujuan.
- i. *Volatility*, ukuran kemungkinan perubahan akan muncul.

Salah satu *metric* berorientasi obyek yang dapat digunakan diusulkan oleh Chidamber and Kemerer yang dibagi menjadi beberapa *metric*:

- *weighted methods per class*
- *depth of the inheritance tree*
- *number of children*
- *coupling between object classes*
- *response for a class*
- *lack of cohesion in methods*

Sedangkan Lorenz and Kidd mengusulkan beberapa *metric* lain untuk pengukur berorientasi obyek:

- *class size*
- *number of operation overridden by a subclass*
- *number of operation added by a subclass*
- *specialization index*

6. Design Metrics for WebApps

Beberapa pertanyaan berikut dapat digunakan sebagai acuan dalam merancang *metric* dalam pengukuran aplikasi mobile atau web:

- a. Apakah *user interface* yang ada mempromosikan kegunaan dari aplikasi?

- b. Apakah unsur estetika dari aplikasi layak untuk domain aplikasi dan menyenangkan user?
- c. Apakah konten sudah dirancang dengan cara yang membagi sebagian besar informasi dengan usaha yang minim?
- d. Apakah navigasi yang ada efisien dan langsung?
- e. Apakah arsitektur dari aplikasi telah dirancang untuk mengakomodasi tujuan dari aplikasi.
- f. Apakah komponen dirancang dalam cara yang mengurangi kompleksitas procedural dan meningkatkan ketepatan, kepercayaan dan performance dari aplikasi?

7. Code Metrics

Halstead's software science: kumpulan komprehensif dari metrik semuanya memprediksi jumlah (angka atau intensitas) dari operator dan operand dalam sebuah program atau komponen.

8. Metrics for Testing

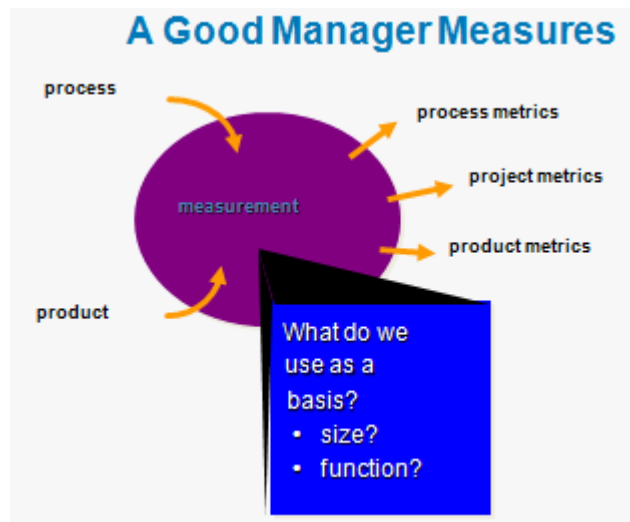
Usaha yang dilakukan untuk pengujian dapat diperkirakan dengan menggunakan metrik yang diturunkan dari ukuran halstead. Berikut ini adalah metrik yang dapat digunakan untuk mengukur pengaruh dari proses pengujian:

- *Lack of cohesion in methods (LCOM)*
- *Percent public and protected (PAP).*
- *Public access to data members (PAD).*
- *Number of root classes (NOR).*
- *Fan-in (FIN).*
- *Number of children (NOC) and depth of the inheritance tree (DIT).*

9. Maintenance Metrics

Software maturity index (SMI) merupakan salah satu *metric* yang diusulkan dalam pengukuran proses *maintenance* yang menyediakan indikasi stabilitas dari sebuah produk *software* (berdasarkan perubahan yang muncul untuk setiap produk yang dikeluarkan).

10. Metrics in the Process and Project Domain



Proses dalam pengembangan suatu softwar diukur secara tidak langsung. Oleh karena itu, dibutuhkan *metric* yang dalam mengukur sebuah proses berdasarkan hasil yang diberikan. Hasil/keluaran dapat berupa:

- *Error* yang tidak tercover sebelum softare dikeluarkan
- *Defect* yang dikirim dan dilaporkan oleh end user
- Produktifitas, jumlah produk kerja yang dikirimkan
- Usaha manusia yang dikeluarkan
- Waktu kalender yang dihabiskan
- Pendekatan jadwal

11. Software Measurement

Pengukuran sebuah software secara keseluruhan dapat diukur melalui ukuran dan fungsi dari software itu sendiri. Metrik berorientasi ukuran:

- *errors per KLOC (thousand lines of code)*
- *defects per KLOC*
- *\$ per LOC*
- *pages of documentation per KLOC*
- *errors per person-month*
- *errors per review hour*
- *LOC per person-month*
- *\$ per page of documentation*

Metrik berorientasi fungsi:

- *errors per FP (thousand lines of code)*
- *defects per FP*
- *\$ per FP*
- *pages of documentation per FP*
- *FP per person-month*

Perbandingan antara *Line of Code* dengan *Function Point*:

Programming Language	LOC per Function point			
	avg.	median	low	high
Ada	154	-	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	63	53	77	-
JavaScript	58	63	42	75
Perl	60	-	-	-
PL/I	78	67	22	263
Powerbuilder	32	31	11	105
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
Visual Basic	47	42	16	158

12. Metrics for Software Quality

Mengukur kualitas sebuah softwar dapat dilihat dari criteria berikut:

- **Correctness**

Ukuran dimana program dalam melakukan operasi-operasi yang seharusnya sesuai dengan spesifikasi.

- **Maintainability**

Ukuran dimana program dapat melakukan penyesuaian terhadap perubahan.

- **Integrity**

Ukuran dimana program dapat bertahan dari serangan luar

- **Usability**

Ukuran dimana program mudah untuk digunakan oleh user.

13. Studi Kasus

Perusahaan dapat menggunakan beberapa metodologi yang ada di dalam pelaksanaan manajemen pengelolaan perangkat lunak, misalnya:

- PMBOK (Project Management Body of Knowledge)
- PRINCE2 (Project IN Controlled Environment)
- Scrum
- Extreme Programming (XP)
- Kanban

Penggunaan metode tersebut tentunya disesuaikan dengan kondisi proyek, misalnya untuk pembangunan proyek dengan kondisi requirement yang sudah dapat ditentukan di awal, maka dapat menggunakan PMBOK dan SDLC. Untuk pembangunan proyek di mana requirement belum dapat ditentukan di awal, maka cocok untuk menggunakan metode agile dengan Scrum.

Software metrics nanti juga berhubungan dengan *quality metrics*, dan ini juga dapat digambarkan sebagai KPI (*Key Process Improvement*) yang merupakan target-target yang perlu dicapai pada proyek.

DAFTAR PUSTAKA

- Pressman, R.S. (2015). *Software Engineering : A Practioner's Approach. 8th ed.* McGraw-Hill Companies.Inc, Americas, New York.
ISBN : 978 1 259 253157.
- Project Management
http://www.pricystems.com/resources/mf_risks_remedies_facts.asp
- Project Portfolio Management
<http://www.daptiv.com/index.htm>
- SW Metrics
<http://www.spc.ca/resources/metrics/>
- Function Point Measurement
<http://www.functionpoints.com>
- SW Metrics service Estimation
http://www.charismatek.com/_public4/html/services/pdf/service_estimate.pdf

LECTURE NOTES

Software Engineering

Minggu 9

Estimation for Software Project And Project Scheduling

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO 4 – Menganalisa proyek *management software*

Outline Materi (Sub-Topic) :

1. *Software Project Planning*
2. *Software Scope*
3. *Resources*
4. *Project Estimation*
5. *Empirical Estimation Models*
6. *Estimation for OO Projects*
7. *Estimation for Agile Projects*
8. *The Make-Buy Decision*
9. *Project Scheduling*
10. *Earned Value Analysis (EVA)*

ISI MATERI

1. *Software Project Planning*

Sebelum Anda melakukan pengerjaan suatu proyek, tahapan perencanaan harus dilakukan untuk memastikan bahwa kita akan membuat perangkat lunak sesuai dengan spesifikasi, biaya, jadwal, dan kualitas yang ditentukan Bersama dengan pelanggan.

Di dalam PMBOK, sebelum perencanaan terdapat proses inisiasi yang berhubungan dengan keputusan formal untuk dimulainya suatu proyek. Gambar di bawah ini memaparkan tahapan-tahapn di dalam proyek, dimulai dari *initiating*, *planning*, *executing*, *monitoring and controlling* sampai proses *closing*.



Tujuan keseluruhan dari perencanaan proyek adalah untuk menetapkan strategi pragmatic untuk mengendalikan, melacak, dan memonitoring proyek teknikal yang kompleks sehingga proyek dapat diselesaikan tepat waktu dengan berkualitas.

Kegiatan-kegiatan dalam perencanaan proyek:

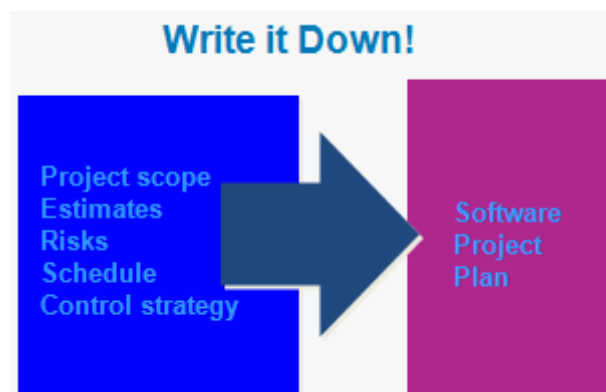
- Menentukan ruang lingkup proyek

- Menentukan kelayakan
- Menganalisa resiko
- Mendefinisikan sumber-sumber daya yang dibutuhkan (meliputi sumber daya manusia, sumber *software* yang dapat digunakan kembali, sumber daya dari lingkungan)
- Memperkirakan biaya dan usaha yang dibutuhkan
- Mengembangkan dua atau lebih prediksi menggunakan ukuran, function point, tugas proses atau *use case*
- Mengembangkan jadwal proyek

Dalam melakukan prediksi terhadap sumber daya, biaya, dan jadwal dibutuhkan:

- Pengalaman
- Akses terhadap informasi *history (metric)*
- Keberanian untuk memberikan prediksi kuantitatif ketika semua informasi kualitatif tersedia.

Prediksi membawa resiko dan resiko memiliki ukuran ketidakpastian.



2. *Software Scope*

Ruang lingkup atau *scope* suatu *software* adalah segala sesuatu yang disepakati untuk dikerjakan pada proyek pembangunan *software*.

Untuk dapat memahami ruang lingkup dari sebuah *software*, maka dibutuhkan pemahaman mengenai:

- Kebutuhan customer
- Konteks bisnis
- Batasan proyek
- Motivasi dari *customer*
- Kemungkinan perubahan

Namun demikian, meskipun semua hal diatas telah dipahami, prediksi tidak dapat dijamin.

3. *Resources*

Sumber daya yang dibutuhkan dari pengembangan sebuah *software* dapat dibagi menjadi 3 kategori utama:

- **Manusia**

Manusia meliputi skill baik teknis maupun softskill, jumlah orang yang dimiliki serta lokasi. Misalnya, pada proyek pembangunan *software* dibutuhkan peran sebagai berikut:

- *Programmer*
- *System analyst*
- *Tester*
- *Business analyst*

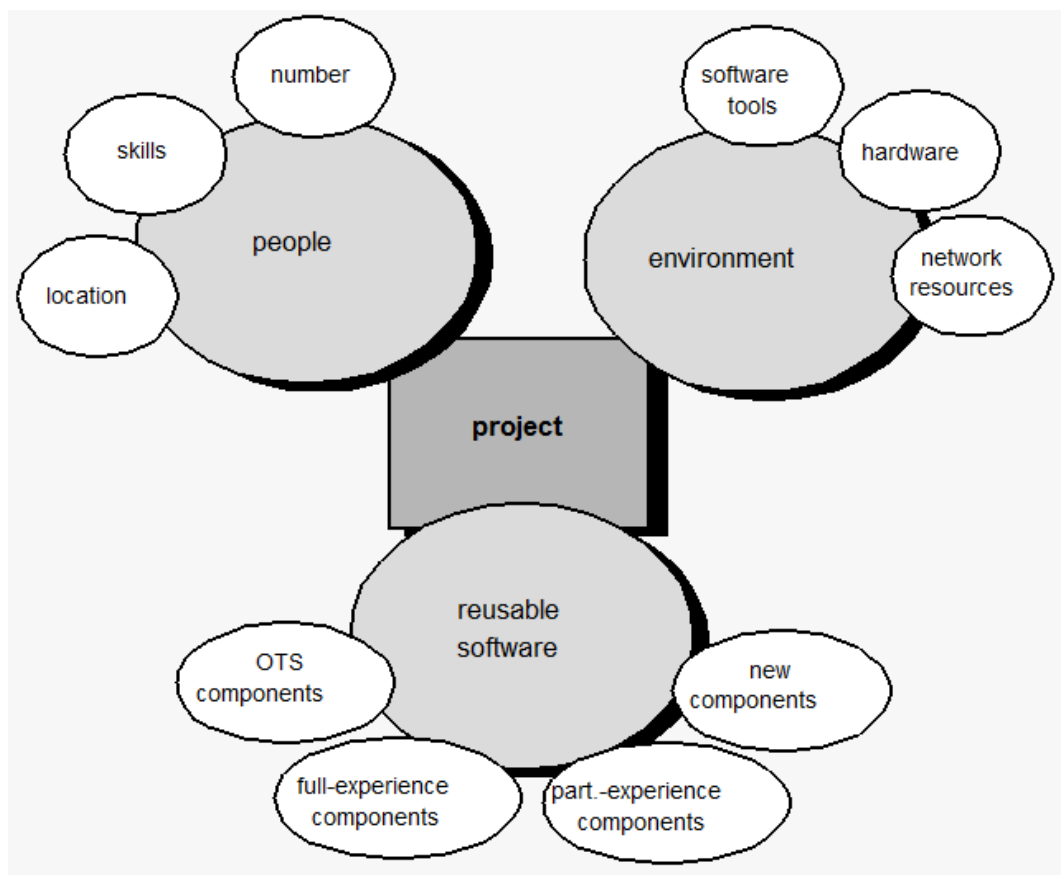
- *Database administrator*
- *Security specialist*

- **Lingkungan**

Lingkungan *software* meliputi *software tools*, *hardware* dan sumber-sumber jaringan.

- **Software yang dapat digunakan kembali (*reuse*)**

Software yang dapat digunakan kembali meliputi: komponen baru, komponen *part-experience*, *component full-experience*, dan *component OTS*.



4. Project Estimation

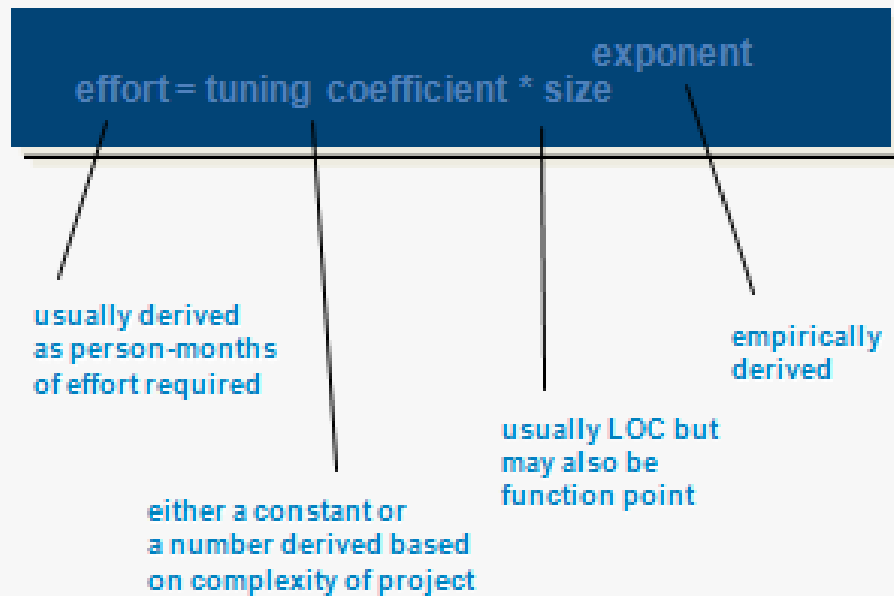
Dalam melakukan prediksi proyek, ruang lingkup harus dipahami seperti yang dibawah pada sub topic di atas. Selain itu, dekomposisi atau elaborasi juga dibutuhkan. Pada banyak kasus, sejarah mengenai metric pada proyek-proyek sebelumnya sangat membantu dalam melakukan prediksi. Dalam melakukan prediksi, setidaknya dibutuhkan dua teknik yang berbeda (contoh di bawah), sehingga hasil dapat dibandingkan dan mendapatkan prediksi yang lebih baik. Dan hal penting yang harus diingat bahwa, prediksi mengandung unsure ketidak pastian, sehingga rencana lebih lanjut perlu dilakukan.

Teknik dalam melakukan prediksi ada 4:

- Masa lampau, dengan melihat pengalaman proyek dari masa lalu untuk proyek-proyek yang sejenis.
- Teknik prediksi yang konvensional
 - o Membagi tugas dan memprediksi usaha
 - o Berdasarkan ukuran (contoh: function point)
- Model empiris
- Tool automasi

5. Empirical Estimation Models

General form:



Gambar diatas adalah contoh model prediksi secara empiris. Selain itu, terdapat COCOMO II yang merupakan sebuah hirarki dari model prediksi yang dalam membantu melakukan prediksi pada area berikut:

- Model komposisi aplikasi

Digunakan pada tahap awal proses pengembangan *software* (*software engineering*), ketika dilakukan prototyping dari *user interface*, pertimbangan terhadap interaksi software dan sistem, penilaian performance, evaluasi kematangan teknologi.

- Model tahap perancangan awal

Digunakan setelah *requirement* ditetapkan dan arsitektur dasar software sudah ditentukan.

- Model tahapan setelah arsitektur

Digunakan selama tahap pembuatan program.

6. *Estimation for OO Projects*

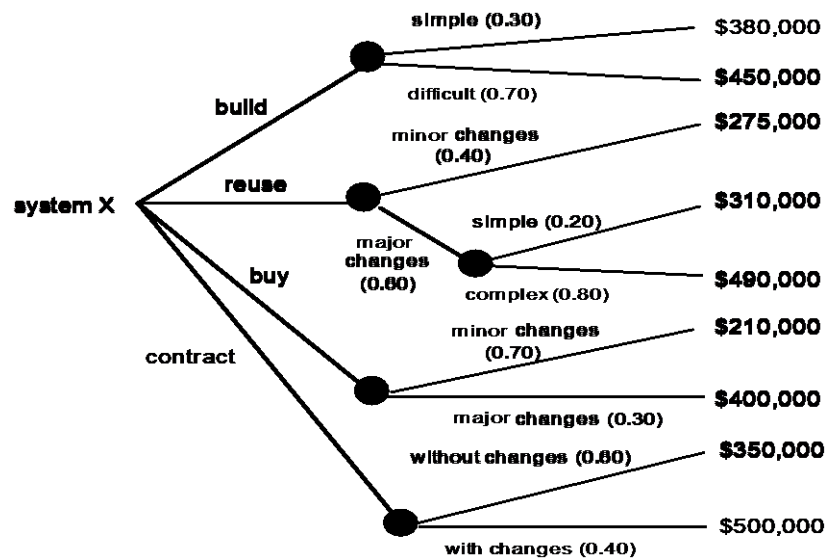
Berikut ini adalah hal-hal yang perlu diperhatikan dalam melakukan prediksi terhadap proyek berorientasi obyek:

- a. Mengembangkan prediksi menggunakan dekomposisi usaha, analisis function point dan metode lainnya yang dapat diterapkan untuk aplikasi konvensional
- b. Menggunakan modeling requirement berorientasi obyek, mengembangkan use case dan menentukan sebuah hitungan
- c. Dari analisis model yang ada, tentukan jumlah dari kelas kunci
- d. Kategorikan tipe interface dari aplikasi dan kembangkan sebuah multiplier untuk kelas pendukung
- e. Kalikan jumlah dari kelas kunci (pada step c) dengan multiplier (pengali) untuk mendapatkan perkiraan jumlah kelas pendukung
- f. Kalikan total kelas (kelas kunci dan kelas pendukung) dengan rata-rata jumlah kerja per kelas.
- g. Cek ulang estimasi berdasarkan kelas dengan mengalikan rata-rata jumlah kerja per use case.

7. *Estimation for Agile Projects*

Dalam melakukan prediksi untuk proyek agile, difokuskan pada *user scenario* (*mini use case*) yang dapat dipertimbangkan secara terpisah untuk kebutuhan perkiraan. *Scenario user* tersebut kemudian didekomposisi menjadi sekumpulan tugas *software engineering* yang dibutuhkan dalam tahap pengembangannya. Setiap tugas diperkirakan secara terpisah (dapat berdasarkan *history*, model empiris atau pengalaman).

8. The Make-Buy Decision



Berdasarkan contoh ilustrasi di atas, diskusikan mengenai keputusan yang sebaiknya diambil dengan alasan yang jelas.

9. Project Scheduling

Berikut ini adalah prinsip-prinsip dalam melakukan penjadwalan:

- Compartmentalization* —mendefinisikan tugas-tugas yang berbeda
- Interdependency* —mengidentifikasi hubungan antar tugas
- Effort validation* —untuk memastikan bahwa sumber daya yang dibutuhkan tersedia
- Defined responsibilities* —orang-orang harus mendapatkan penugasan.
- Defined outcomes* —setiap tugas harus memiliki keluaran (*output*)
- Defined milestones* —melakukan *review* terhadap kualitas

Dalam melakukan penjadwalan untuk proyek aplikasi web atau mobile, terdapat 7 *increments*:

- Informasi dasar dari produk dan perusahaan

- Informasi detail dari produk dan download
- Kuota produk dan urutan proses produk
- Layout space dan rancangan sistem keamanan
- Informasi dan meminta pelayanan monitoring
- Kendali online dari peralatan monitoring
- Kendali akses informasi

10. Earned Value Analysis (EVA)

Earned value adalah ukuran dari sebuah progress/kemajuan yang digunakan untuk menilai kelengkapan suatu software dalam persen. Earned value memberikan analisis kuantitatif. Di dalam organisasi, perhitungan EVA sudah tercakup di dalam tools manajemen proyek.

11. Studi Kasus

Pada organisasi, proyek pengembangan perangkat lunak biasanya menggunakan metode yang sudah biasa dipakai, misalnya PMBOK, SDLC, Agile dan lain-lain. Metode tersebut juga memiliki panduan yang lebih rinci dan sesuai dengan yang dijelaskan pada dokumen ini.

Untuk proses estimasi, Kebanyakan organisasi di dalam menggunakan satuan man-days, misalnya untuk melakukan pekerjaan instalasi operating system dibutuhkan 2 man-days. Untuk estimasi biaya biasa digunakan man-days rate, misalnya 6 juta per man-days. Setiap organisasi dapat memiliki man-days yang berbeda-beda.

DAFTAR PUSTAKA

- Software engineering : a practitioners approach : Chapter 35/Pages 777, Chapter 36/Pages 795
- Sw maintenance and Reengineering,
<http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/13.25.SWEng4/html/text.html>
- Risk Management & operational Risk,
<http://www.grafp.com/products/risk-manage.html>

LECTURE NOTES

Software Engineering

Minggu 10

Software Risk Management And Reengineering

LEARNING OUTCOMES

Setelah menyelesaikan pembelajaran ini, mahasiswa akan mampu:

- ☒ LO 4 – Menganalisa proyek *management software*

Outline Materi (Sub-Topic) :

1. *Reactive versus Proactive Risk Strategies*
2. *Software Risk*
3. *Software Maintenance*
4. *Business Process Reengineering*
5. *Software Reengineering*
6. *Reverse Engineering*
7. *Forward Engineering*
8. *The Economics of Reengineering*
9. Studi Kasus

ISI MATERI

1. *Project Risk Management*

Risiko memiliki arti sesuatu yang bersifat tidak pasti (uncertainty) dan memiliki kemungkinan terjadi pada masa yang akan datang. Risiko sendiri yang sering dipahami adalah sesuatu yang bersifat negative dan berdampak pada proyek, dapat secara Teknik atau pun non teknis.

Risiko yang bersifat teknis misalnya:

- Kemungkinan terjadi nya kehilangan data karena tidak ada backup system
- Kemungkinan adanya penyusup karena penerapan system security yang kurang memadai
- Kemungkinan terjadinya issue pada basis data

Adapun risiko yang sifatnya non teknis lebih pada aspek komponen dari manajemen proyek, seperti:

- Ada risiko mengenai keterlambatan proyek
- Risiko pengeluaran biaya yang melebihi anggaran
- Risiko mengenai adanya scope yang tidak dapat diimplementasikan
- Risiko tidak diterimanya *software* oleh pelanggan

Risiko-risiko tersebut perlu diidentifikasi di awal untuk mengurangi dampak dan kemungkinan terjadi, sehingga diperlukan pengetahuan dan ketrampilan mengenai pengelolaan manajemen risiko oleh semua tim proyek.

2. *Reactive versus Proactive Risk Strategies*

Rective risk management:

- Tim proyek bereaksi terhadap resiko ketika resiko itu muncul
- Mitigasi – rencana untuk sumber daya tambahan sebagai bentuk antisipasi
- Memperbaiki kegagalan, sumber daya ditemukan dan diimplementasikan ketika resiko terjadi
- Manajemen krisis, kegagalan tidak merespon untuk diterapkan untuk sumber daya dan proyek dalam bahaya

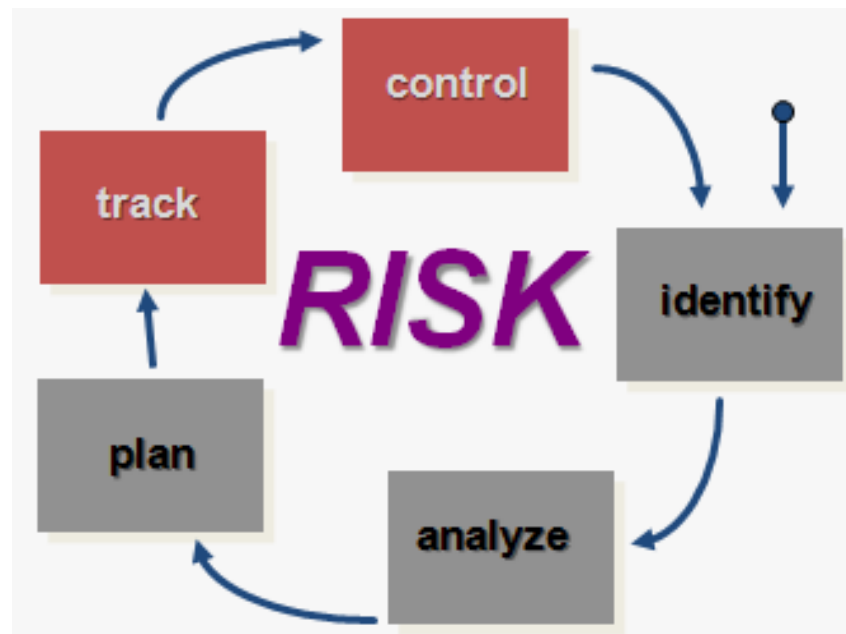
Proactive risk management:

- Analisa resiko secara formal dijalankan
- Melakukan perbaikan terhadap akar masalah dari resiko, hal ini dapat menggunakan konsep TQM dan analisis SQA, serta memeriksa sumber dari resiko.

3. *Software Risk*

Tujuh prinsip resiko *software*:

- a. Menjaga prespektif global
- b. Mengambil pandangan jauh ke depan
- c. Menghimbau komunikasi terbuka
- d. Integrasi
- e. Menekankan proses yang berlanjut
- f. Mengembangkan visi produk yang sama
- g. Mendorong kerja tim



Mengidentifikasi resiko sebuah software dapat dilakukan dengan mempertimbangkan hal berikut:

- Ukuran produk, resiko diasosiasikan dengan ukuran secara keseluruhan dari software yang akan dibuat/dimodifikasi.
- Dampak bisnis, resiko diasosiasikan dengan batasan yang dimiliki oleh pihak *management* atau pasar.
- Karakteristik *customer*, resiko diasosiasikan dengan kecanggihan dari customer dan keahlian dari developer dalam berkomunikasi dengan customer dengan cara yang tepat.
- Defisi proses, resiko diasosiasikan dengan ukruang dimana proses software telah didefinisikan dan diikuti oleh perusahaan.
- Lingkungan pengembangan, resiko diasosiasikan dengan ketersediaan dan kualitas dari tool yang digunakan untuk membangun produk.
- Teknologi yang akan dibuat, resiko diasosiasikan dengan kompleksitas sistem yang akan dibuat dan keterbaruan teknologi yang disatukan dengan sistem tersebut.

- Pengalaman dan jumlah staff, resiko diasosiasikan dengan pengalaman proyek dan teknikal dari para software engineer yang akan melakukan pengembangan.

4. *Software Maintenance*

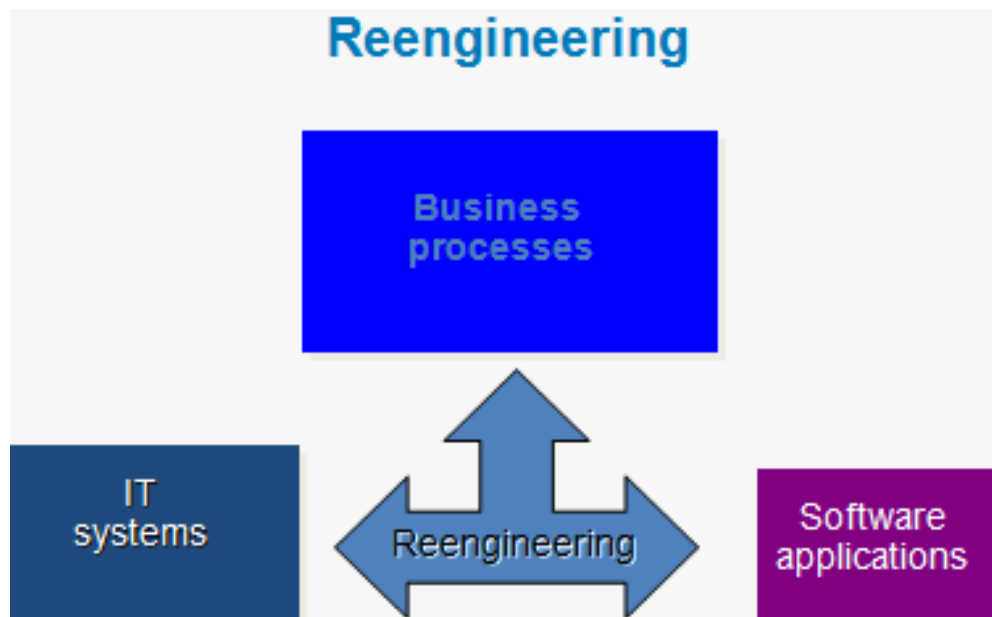
Ketika *software* telah direlease, maka kegiatan setelahnya disebut sebagai software maintenance proses.

- Memperbaiki laporan *bug* yang diterima
- Permintaan terhadap perubahan *software* sehingga dapat memenuhi kebutuhan khusus.
- Permintaan pengembangan *softwar* dari perusahaan untuk meningkatkan keuntungan.

Untuk memenuhi tuntutan dalam proses maintenance, *software* harus dapat di *maintain* (*maintainable*):

- *Software* yang dapat dimaintain menunjukkan modularitas yang efektif
- Menggunakan *design pattern* yang memudahkan pemahaman
- Dikembangkan dengan menggunakan konvesi dan standar kode yang bagus, sehingga menghasilkan *source code* yang dapat mendokumentasikan dirinya sendiri serta dapat mudah dipahami.
- Telah melewati berbagai teknik penjaminan kualitas yang membuka masalah maintaince yang potensial sebelum software direlease.
- Dibuat oleh software engineer yang memahami bahwa mereka mungkin tidak berada ditempat ketika terjadi masalah.

5. *Business Process Reengineering*



Pada gambar dibawah ini, dapat dilihat proses-proses yang ada dalam BPR (*Business Process Reengineering*):

- **Pendefinisian bisnis**

Tujuan bisnis diidentifikasi dalam konteks 4 driver kunci: mengurangi biaya, mengurangi waktu, meningkatkan kualitas, dan pengembangan serta pemberdayaan personel.

- **Identifikasi proses**

Identifikasi proses-proses yang penting dalam mencapai tujuan yang didefinisikan dalam bisnis.

- **Evaluasi proses**

Proses yang digunakan untuk melakukan pengukuran dan analisa.

- **Perancangan dan spesifikasi proses**

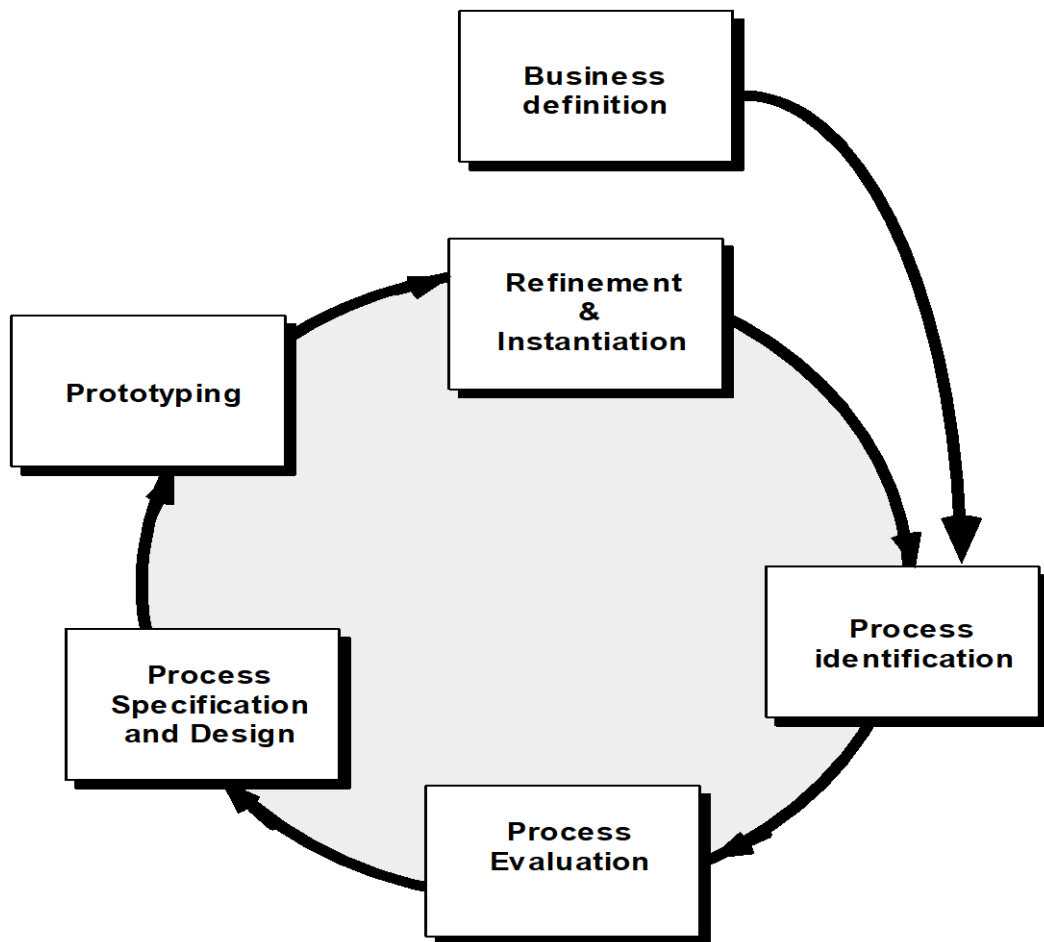
Berdasarkan informasi yang didapat selama 3 fase pertama BPR, use case disiapkan untuk setiap proses yang dirancang kembali

- **Prototyping**

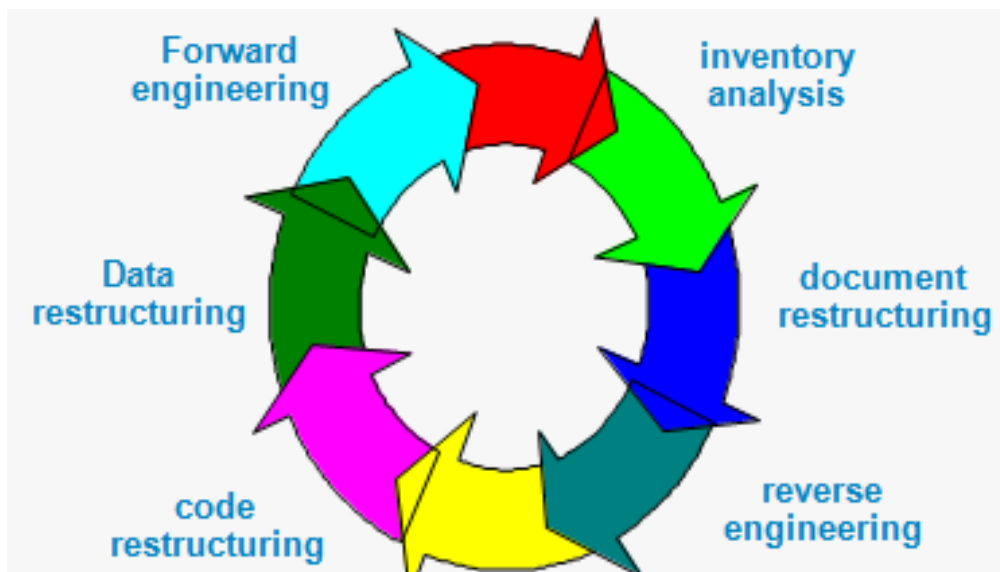
Perancangan kembali proses bisnis harus dibuat prototypenya sebelum benar-benar diintegrasikan ke dalam bisnis.

- **Perbaikan dan instansiasi**

Berdasarkan masukan dari proses prototype, proses bisnis diperbaiki dan kemudia diinstansiasikan ke dalam sistem bisnis.



6. *Software Reengineering*



Inventory Engineering:

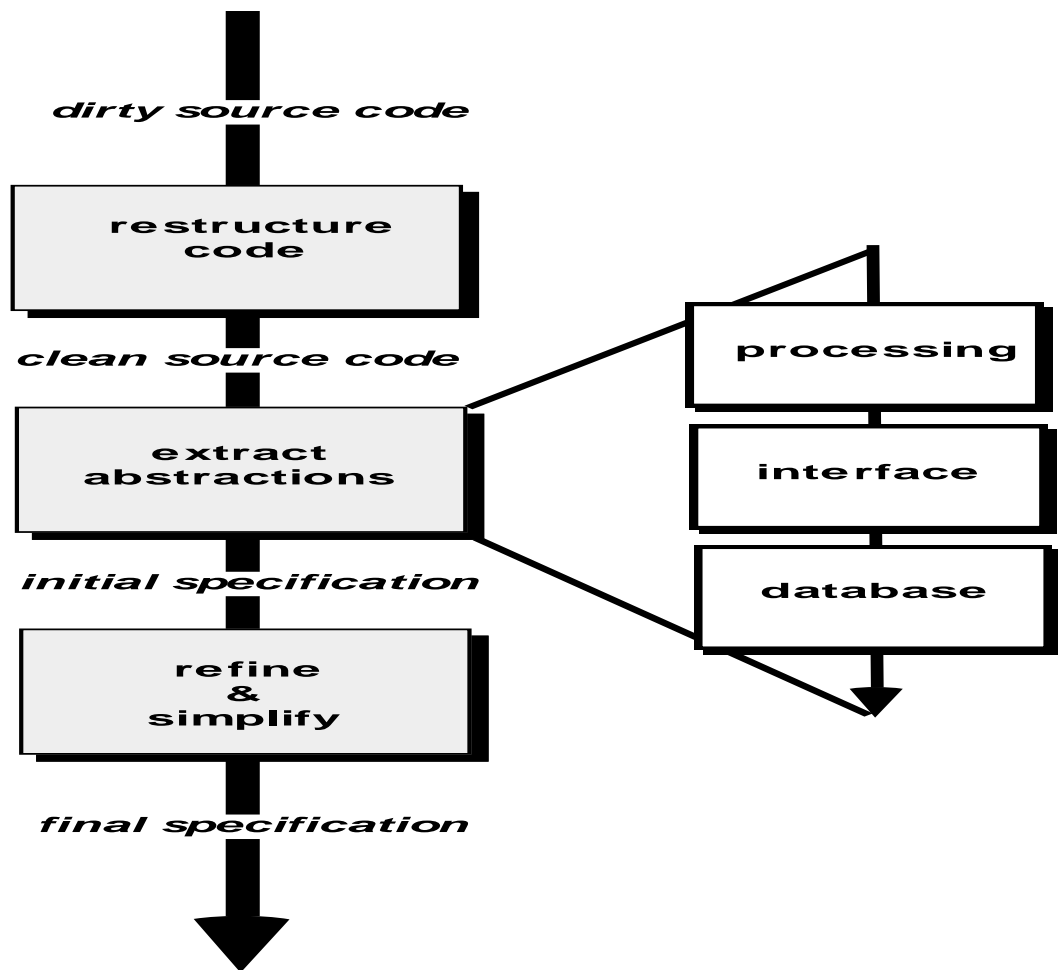
Software engineering dimulai dengan tahapan *inventory engineering* dengan membuat sebuah tabel yang berisi semua aplikasi yang dimiliki. Kemudian menganalisa dan memprioritaskan kandidat aplikasi yang akan di-*reengineering*.

Document Restructuring:

Penstrukturan kembali dokumen dapat dilakukan:

- Menciptakan dokumentasi yang tidak memakan waktu.
- Dokumentasi harus selalu diperbarui, namun hal ini memiliki keterbatasan sumber daya.
- Sistem adalah kritikal bisnis dan harus selalu diredokumentasikan secara penuh.

7. Reverse Engineering



Code Restructuring:

- *Source code* dianalisa menggunakan tool restructuring
- Bagian kode yang dirancang dengan kurang baik akan dirancang ulang.
- Pelanggaran terhadap struktur pemrograman dicatat dan distruktur ulang
- *Code* yang telah distruktur ulang direview dan diuji untuk memastikan bahwa tidak ada anomaly yang disebabkan
- Dokumentasi untuk *internal code* diperbarui

8. *Forward Engineering*

- a. Biaya untuk maintain satu baris dari *source code* mungkin 20-40 kali lipat dari biaya awal yang dikeluarkan pada proses pengembangan untuk baris tersebut.
- b. Merancang ulang arsitektur *software* menggunakan konsep perancangan *modern* dapat memfasilitasi proses maintenance di masa depan
- c. Karena *prototype* dari aplikasi sudah ada, produktifitas pengembangan harus lebih tinggi dari rata-rata
- d. *User* sudah memiliki pengalaman terhadap *software*, oleh karena itu, requirement baru dan arah perubahan dapat dihadapi dengan lebih mudah
- e. Tool *CASE* untuk *reengineering* akan mengautomasi beberapa bagian pekerjaan.
- f. Konfigurasi *software* yang lengkap akan dihasilkan dari penyelesaian proses

9. Studi kasus

Di dalam pendefinisian manajemen risiko, banyak terjadi kesalah pahaman antara risiko dan problem. Problem adalah sesuatu masalah yang saat ini sedang terjadi, sedangkan risiko adalah kemungkinan terjadinya masalah pada waktu yang akan datang. Problem sedang atau sudah terjadi, sedangkan risiko belum terjadi. Problem diselesaikan dengan issue atau problem management, sedangkan risiko dikelola melalui manajemen risiko.

Contoh problem: user tidak dapat masuk ke dalam system karena system down

Contoh risiko: karena tidak ada backup server, terjadi kemungkinan system tidak dapat diakses Ketika terjadi crash pada system yang ada sekarang

DAFTAR PUSTAKA

- Software engineering : a practitioners approach : Chapter 35/Pages 777, Chapter 36/Pages 795
- Sw maintenance and Reengineering,
<http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/13.25.SWEng4/html/text.html>
- Risk Management & operational Risk,
<http://www.grafp.com/products/risk-manage.html>