



Software Development Life Cycle

Radical Rakhman Wahid



26 Februari 2019
21 Jumadil Akhir 1440

Pengertian



Sebelum melangkah lebih jauh...

SDLC adalah proses pengembangan perangkat lunak dengan metode yang disusun secara sistematis untuk membantu para pengembang mengembangkan perangkat lunak dengan baik.

SDLC penting untuk diterapkan agar para *programmer* tidak menemukan kendala-kendala seiring dengan perkembangan skala sistem-sistem perangkat yang semakin besar. Hal ini karena dengan adanya SDLC arah/langkah-langkah dari pengembangan suatu kegiatan rekayasa perangkat lunak terarah.

Yang ada pada presentasi ini :



1. Model *Waterfall*
2. Model *Process Incremental* (*Iteratif, Rapid Application Development, Agile*)
3. Model *Process Evolutionary* (*Prototipe, Spiral, Concurrent Development*)
4. Model *Proses Khusus* (*Berbasis Komponen, Formal, Aspect Oriented*)

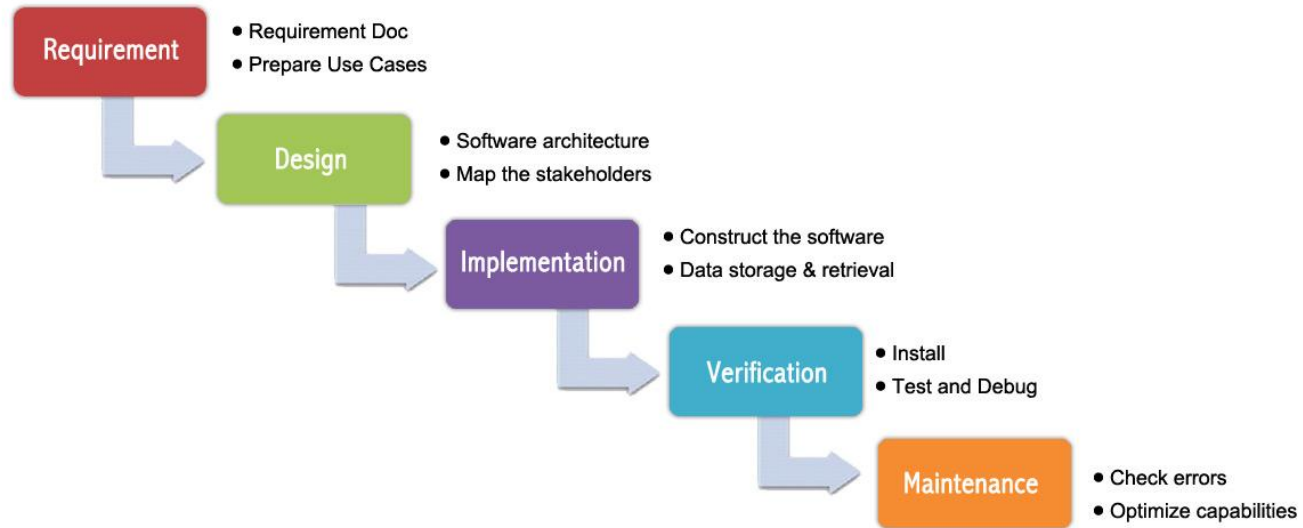
1. *Model Waterfall*



Model *waterfall* adalah model yang paling sederhana. Model ini hanya cocok untuk mengembangkan perangkat lunak dengan spesifikasi yang tidak berubah-ubah.



Ilustrasi model




Kelebihan model *waterfall*



1. Memiliki proses yang urut, mulai dari analisa hingga *support*
2. Setiap proses memiliki spesifikasinya sendiri, sehingga sebuah sistem dapat dikembangkan sesuai dengan apa yang dikehendaki (tepat sasaran).
3. Setiap proses tidak dapat saling tumpang tindih.

Alasan mengapa model *waterfall* jarang dilakukan sesuai dengan alurnya:

- 
1. Perubahan spesifikasi perangkat lunak di tengah alur pengembangan
 2. *Client* kesulitan mendefinisikan semua spesifikasi di awal alur pengembangan
 3. *Client* seringkali menginginkan perubahan spesifikasi di tengah pengembangan

2. Model *Process Incremental*

Iteratif, *Rapid Application Development*, dan Agile

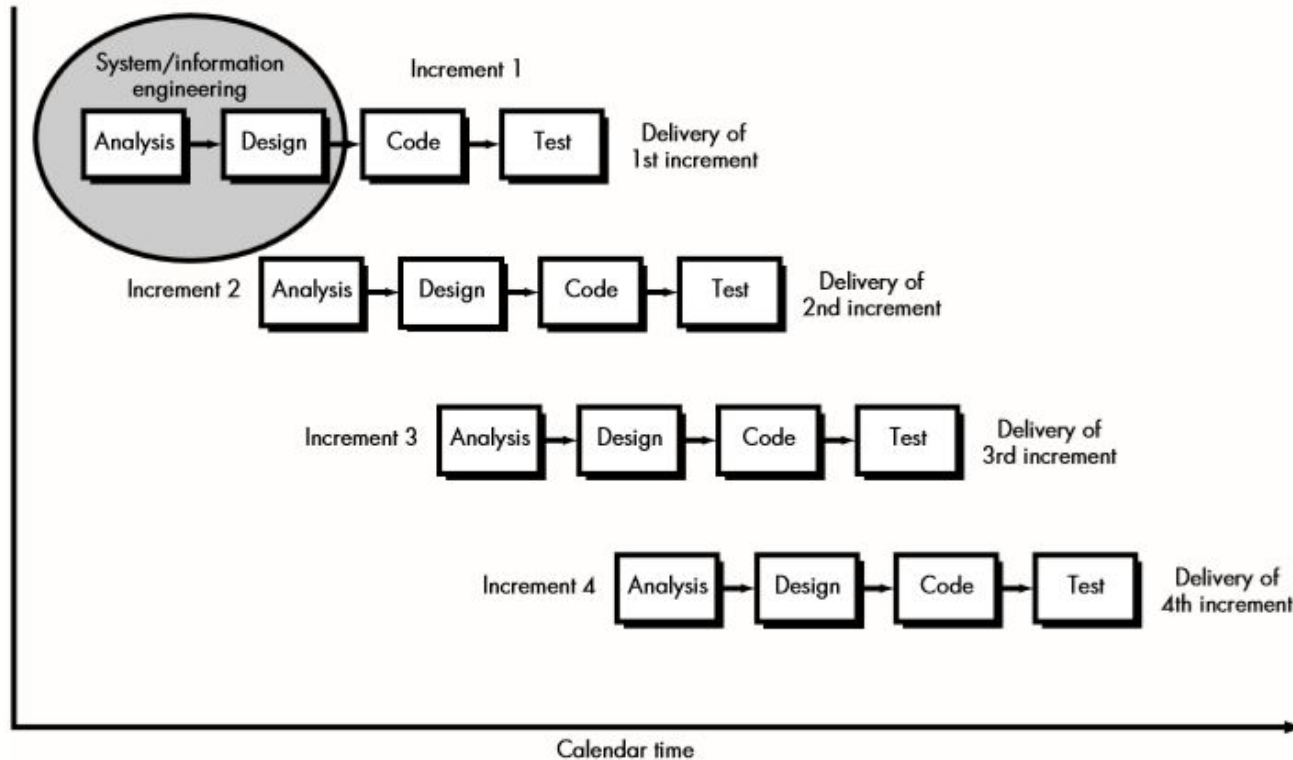


2.1

Model iteratif ada untuk **mengatasi kelemahan** dari **model *waterfall*** yang tidak mengakomodasi iterasi, dan **mengatasi kelemahan** dari **model prototipe** yang memiliki proses terlalu pendek dan setiap iteratif prosesnya tidak selalu menghasilkan produk.



Ilustrasi model



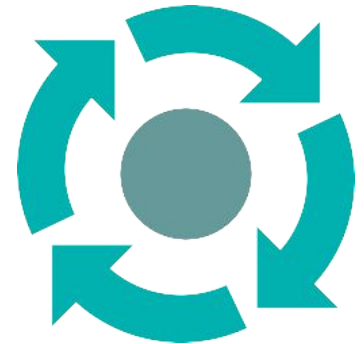
Kekurangan model iteratif



1. Membutuhkan *resource* yang cukup banyak
2. Akan terjadi masalah jika mengubah spesifikasi di tengah-tengah pengembangan
3. Memerlukan Perhatian manajemen
4. Permasalahan sistem arsitektur dan desain mungkin akan timbul, karena tidak semua persyaratan ditentukan di awal pengembangan sistem.
5. Tidak cocok untuk proyek kecil
6. Kompleksitas manajemen
7. Membutuhkan tenaga ahli untuk analisis resiko yang bisa timbul kapan saja

Kelebihan model iteratif

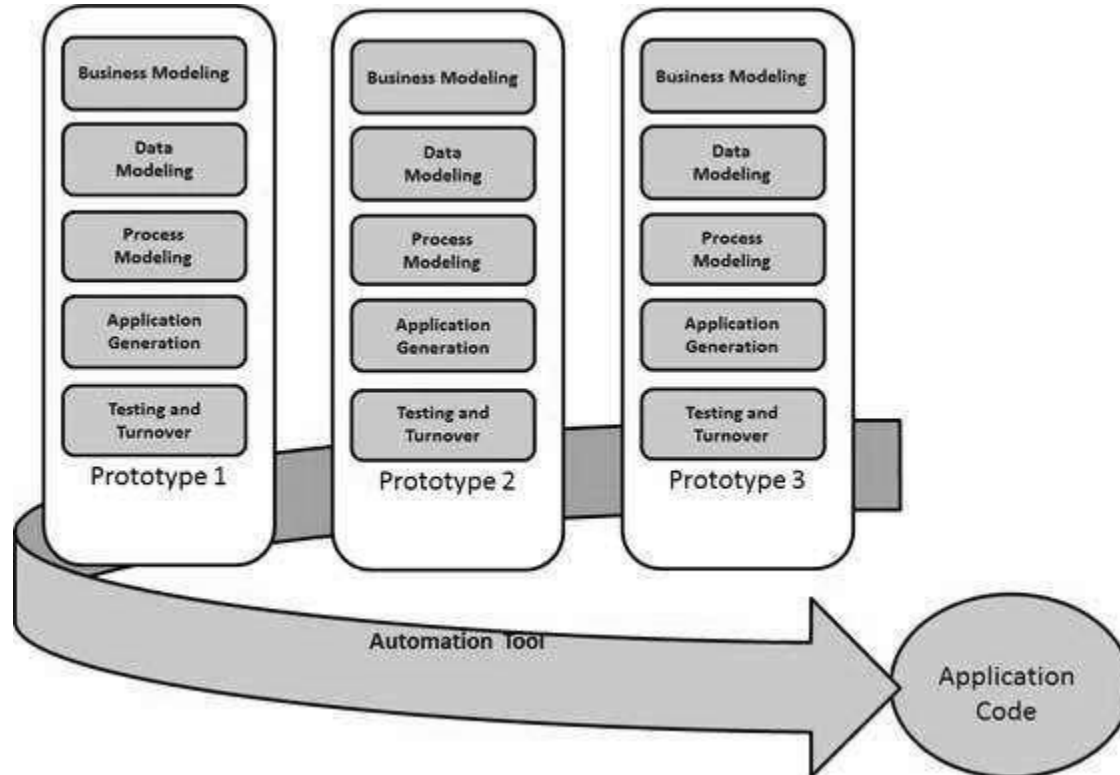
1. Beberapa fungsi dapat di kembangkan dengan cepat di awal pembuatan versi baru.
2. Hasil yang diperoleh secara berkala
3. Kemajuan sebuah sistem dapat diukur
4. Mudah dalam perencanaan
5. Biaya yang dikeluarkan kecil apabila ingin merubah spesifikasi
6. Testing dan debugging selama proses iterasi lebih mudah.
7. Analisis resiko yang lebih baik
8. Mendukung perubahan spesifikasi
9. Waktu operasional yang lebih singkat
10. Cocok untuk proyek besar



Rapid Application Development diadaptasi dari model *waterfall*, di mana tim pengembang dibagi menjadi beberapa tim untuk mengerjakan beberapa komponen masing-masing, sehingga dapat **dilakukan secara paralel**.



Ilustrasi model



Kekurangan model RAD



1. Sumber daya manusia yg besar
2. Harus dikembangkan dengan cepat
3. Sistem harus dimodulkan terlebih dahulu
4. Tidak cocok untuk teknologi baru yang belum banyak dikenal dan dikuasai pengembang

RAD Sangat Cocok Digunakan Oleh

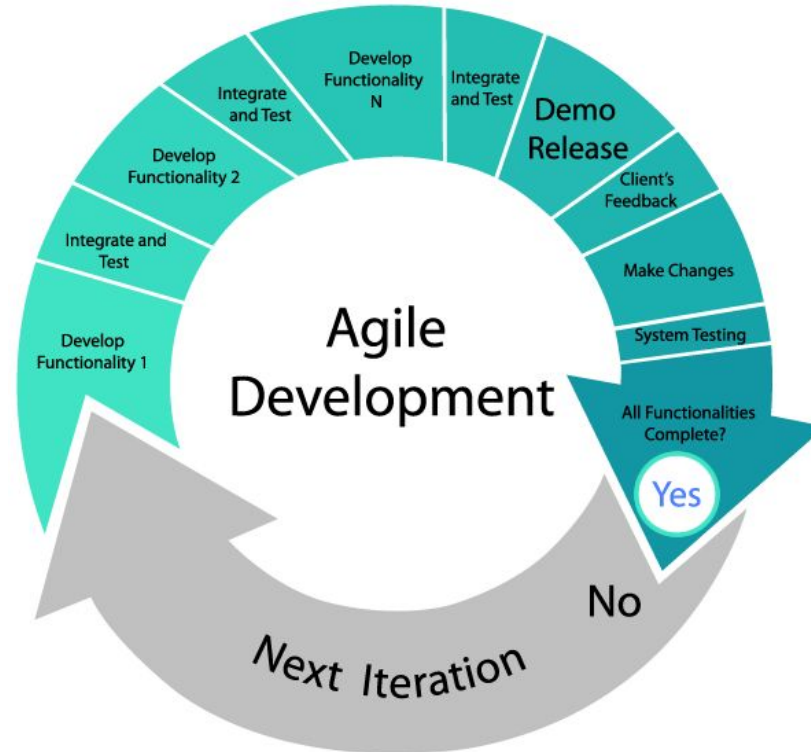


1. Tim pengembang yg sudah berpengalaman
2. Tim pengembang yg memiliki
komponen-komponen sistem yang bisa
digunakan kembali dalam proyek tersebut

Agile adalah kumpulan dari metode-metode pengembangan perangkat lunak yang berbasis pada iteratif dan *Incremental Model*. ***Agile*** memungkinkan mengembangkan perangkat lunak yang memiliki ***spesifikasi*** yang mudah berubah dengan cepat.



Ilustrasi model



Kelebihan model *Agile*



1. Proses iteratif dan *incremental*.
2. Tidak masalah jika spesifikasi dapat berubah sewaktu-waktu.
3. Pelacakan spesifikasi dapat dilakukan dengan melihat *Backlog* produk.
4. Keterlibatan *user* secara aktif.
5. Rilis yang lebih cepat dan berkala, fungsi dirilis setiap akhir iterasi.
6. Testing dilakukan setiap saat.

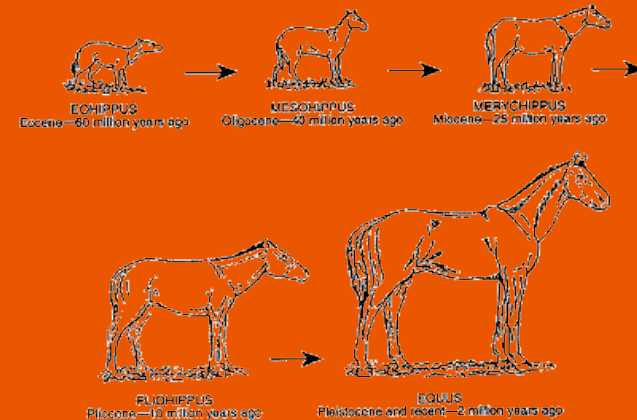
Kekurangan model *Agile*



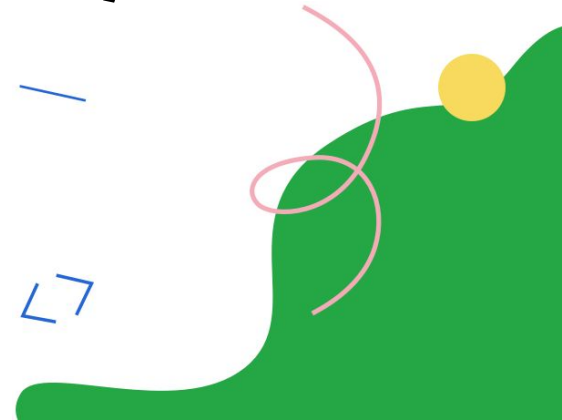
1. Interaksi dengan *client* yang kadang terlalu berlebihan.
2. Agile sulit diimplementasikan dalam proyek yang berskala besar.
3. Waktu perencanaan proyek yang singkat.
4. Membutuhkan manajemen tim yang terlatih.

3. Model *Process Evolutionary*

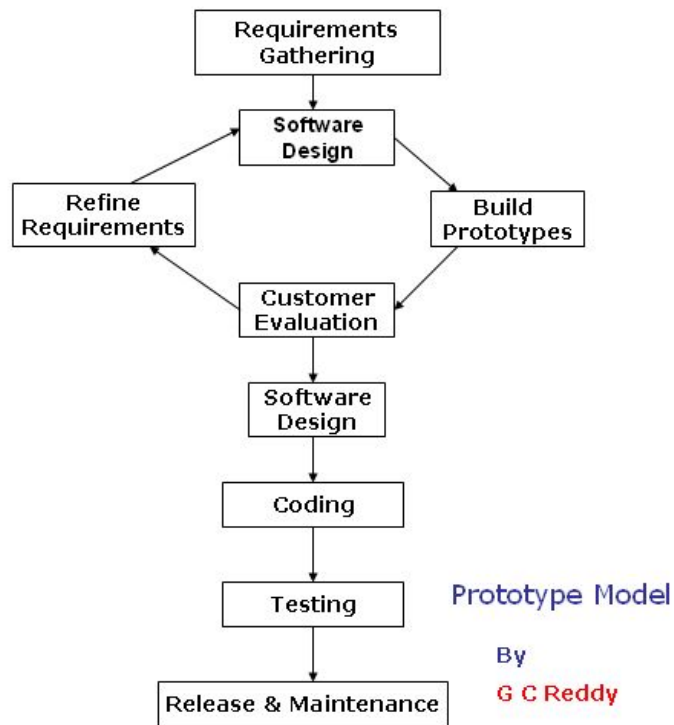
Prototipe, Spiral, dan *Concurrent Development*



Model Prototipe cocok digunakan untuk mengenali spesifikasi kebutuhan pelanggan secara lebih detail namun beresiko terhadap membengkaknya biaya dan waktu proyek.



Ilustrasi model



Kekurangan model prototipe



1. *Client* bisa mengubah-ubah/menambah spesifikasi kebutuhan dan pengembang bisa jadi banyak mengalah.
2. Perangkat lunak yang dihasilkan kurang baik karena spesifikasi dari *client* tidak konsisten.
3. Model ini kurang cocok untuk aplikasi skala besar karena akan memakan waktu dan tenaga besar pula.

Kelebihan model prototipe

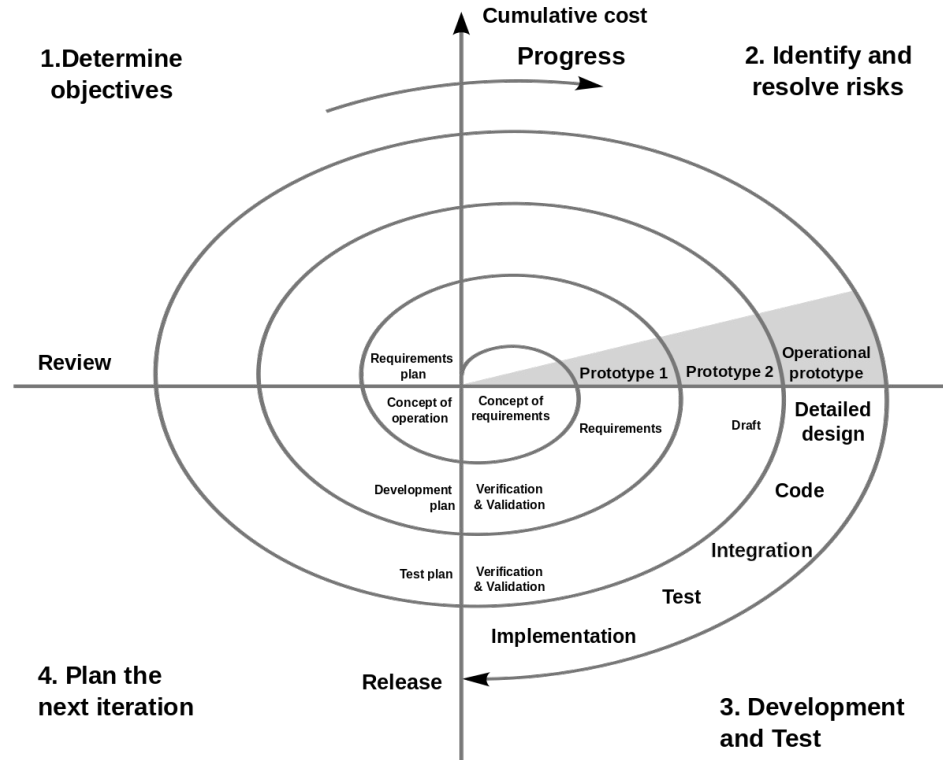


1. *Client* tidak perlu menunggu lama seperti dalam Model *waterfall*.
2. Umpan balik dari pelanggan diterima secara berkala dan perubahan tidak datang sebagai kejutan pada menit-menit terakhir.
3. Interaksi pelanggan meningkatkan kualitas dan juga tingkat keberhasilan(subjektif).

Model spiral memasangkan model iteratif pada model prototipe dengan kontrol dan aspek sistematis yang diambil dari model *waterfall*. Hasil akhir dan evaluasi dari sebuah wilayah kerja akan menjadi inisiasi dari wilayah kerja berikutnya.



Ilustrasi model



Kekurangan model spiral



1. Banyak konsumen (Client) tidak percaya bahwa pendekatan secara evolusioner dapat dikontrol oleh kedua pihak. Model spiral mempunyai resiko yang harus dipertimbangkan ulang oleh konsumen dan developer.
2. Memerlukan tenaga ahli untuk memperkirakan resiko, dan harus mengandalkannya supaya sukses.
3. Belum terbukti apakah metode ini cukup efisien karena usianya yang relatif baru.
4. Memerlukan penaksiran resiko yang masuk akal dan akan menjadi masalah yang serius jika resiko mayor tidak ditemukan dan diatur.
5. Butuh waktu lama untuk menerapkan paradigma ini menuju kepastian yang absolute.

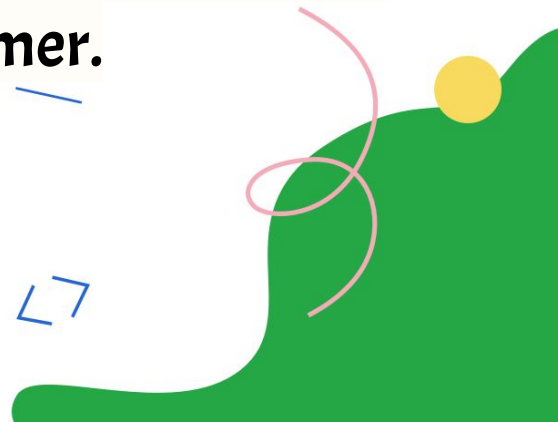
Kelebihan model spiral



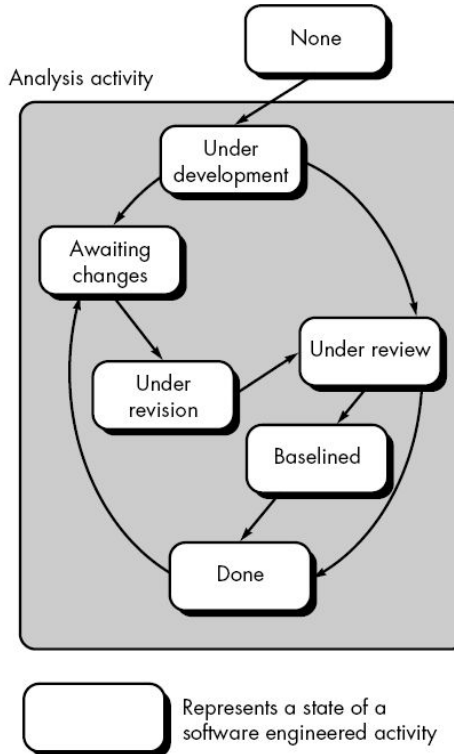
1. Setiap tahap pengerjaan dibuat prototyping sehingga kekurangan dan apa yang diharapkan oleh client dapat diperjelas dan juga dapat menjadi acuan untuk client dalam mencari kekurangan kebutuhan.
2. Lebih cocok untuk pengembangan sistem dan perangkat lunak skala besar.
3. Dapat disesuaikan agar perangkat lunak bisa dipakai selama hidup perangkat lunak komputer.
4. Pengembang dan pemakai dapat lebih mudah memahami dan bereaksi terhadap resiko setiap tingkat evolusi karena perangkat lunak terus bekerja selama proses.
5. Menggunakan prototipe sebagai mekanisme pengurangan resiko dan pada setiap keadaan di dalam evolusi produk.
6. Tetap mengikuti langkah-langkah dalam siklus kehidupan klasik dan memasukkannya ke dalam kerangka kerja iteratif.
7. Membutuhkan pertimbangan langsung terhadap resiko teknis sehingga mengurangi resiko sebelum menjadi permasalahan yang serius.

3.3

Pada model *concurrent development* aktifitas kerja dilakukan secara bersamaan, setiap proses kerja memiliki beberapa pemicu kerja dari aktifitas. Pemicu dapat berasal dari awal proses kerja maupun dari pemicu yang lain karena setiap pemicu akan saling berhubungan. Misalnya proses desain akan berubah atau dihentikan sementara karena ada perubahan permintaan kebutuhan dari customer.



Ilustrasi model



Kekurangan model *concurrent development*



1. Memungkinkan terjadinya perubahan besar-besaran, maka akan membuat biaya dan waktu yang diperlukan membengkak
2. statenya sangat banyak sehingga membutuhkan waktu lebih banyak.

Kelebihan model *concurrent development*



1. berlaku untuk semua jenis pengembangan perangkat lunak dan memberikan gambaran yang akurat tentang keadaan sekarang dari suatu proyek.
2. Hasil yang di dapat akan menghasilkan suatu sistem yang sangat baik karena terdapat perancangan yang terjadi secara besar dan terencana secara matang.

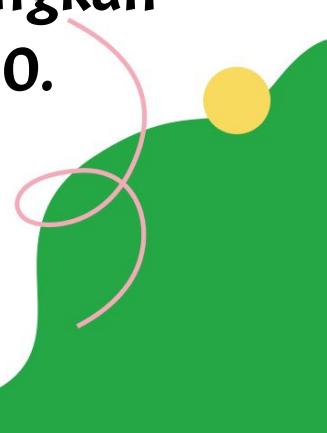
4. Model Proses Khusus

Berbasis Komponen, Formal, *Aspect Oriented*

Specials

4.1

Model pengembangan berbasis komponen adalah perkembangan dari model spiral. Model pengembangan berbasis komponen mengarah ke penggunaan kembali perangkat lunak(*reuse*). Reuse ini akan sangat memudahkan developers dalam mengembangkan perangkat lunak baru karena proses tidak dimulai dari 0.



Ilustrasi model

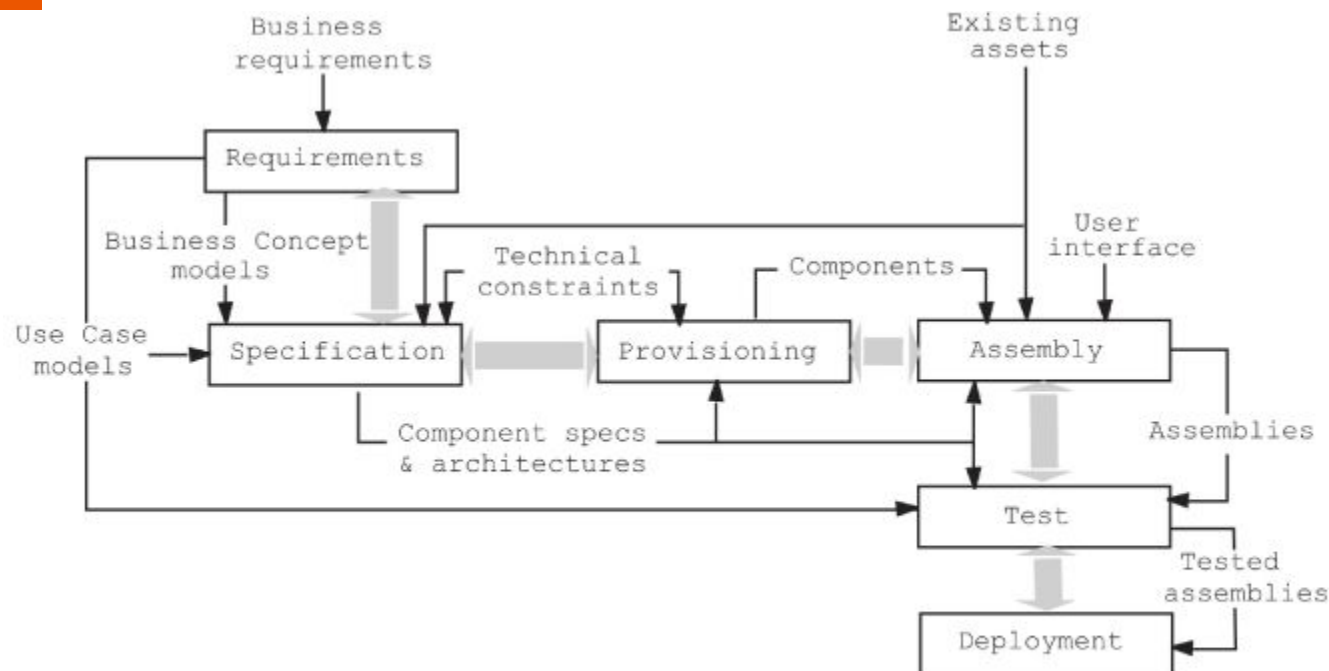


Figure 2. Workflows in the component-based development process[6]

Kekurangan model berbasis komponen



1. Memerlukan arsitektur framework dan sistem aplikasi tertentu sesuai dengan kebutuhan.
2. Membutuhkan komponen yang harus pas dan sesuai yang dibutuhkan
3. Biaya akan menjadi mahal bila komponen yang dibutuhkan berkualitas.

Kelebihan model berbasis komponen



1. Produk berbasis komponen yang tersedia diteliti dan dievaluasi untuk domain aplikasi yang bersangkutan.
2. Masalah integrasi komponen dipertimbangkan.
3. Arsitektur perangkat lunak dirancang untuk mengakomodasi komponen.
4. Komponen diintegrasikan ke dalam arsitektur.
5. Pengujian komprehensif dilakukan untuk memastikan fungsionalitas yang tepat.

Metode formal adalah SDLC yang menerapkan metode atau teknik matematika untuk proses pengembangan sistem perangkat lunak yang kompleks. Pendekatan ini menggunakan bahasa spesifikasi formal untuk mendefinisikan setiap karakteristik sistem.



Ilustrasi model

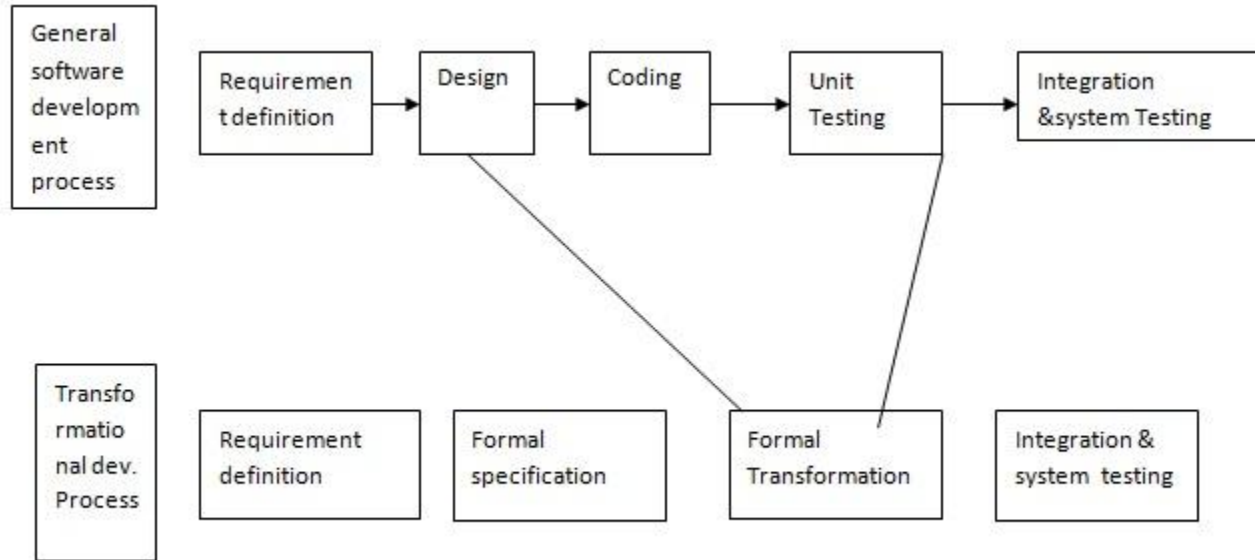


Fig. 2.16, Formal System Development

Kekurangan model formal



1. Membutuhkan waktu yang sangat banyak.
2. Bukti program sangat tidak praktis dan tidak sesuai untuk proyek besar.
3. Harus ditangani oleh orang yang berpengalaman.

Kelebihan model formal

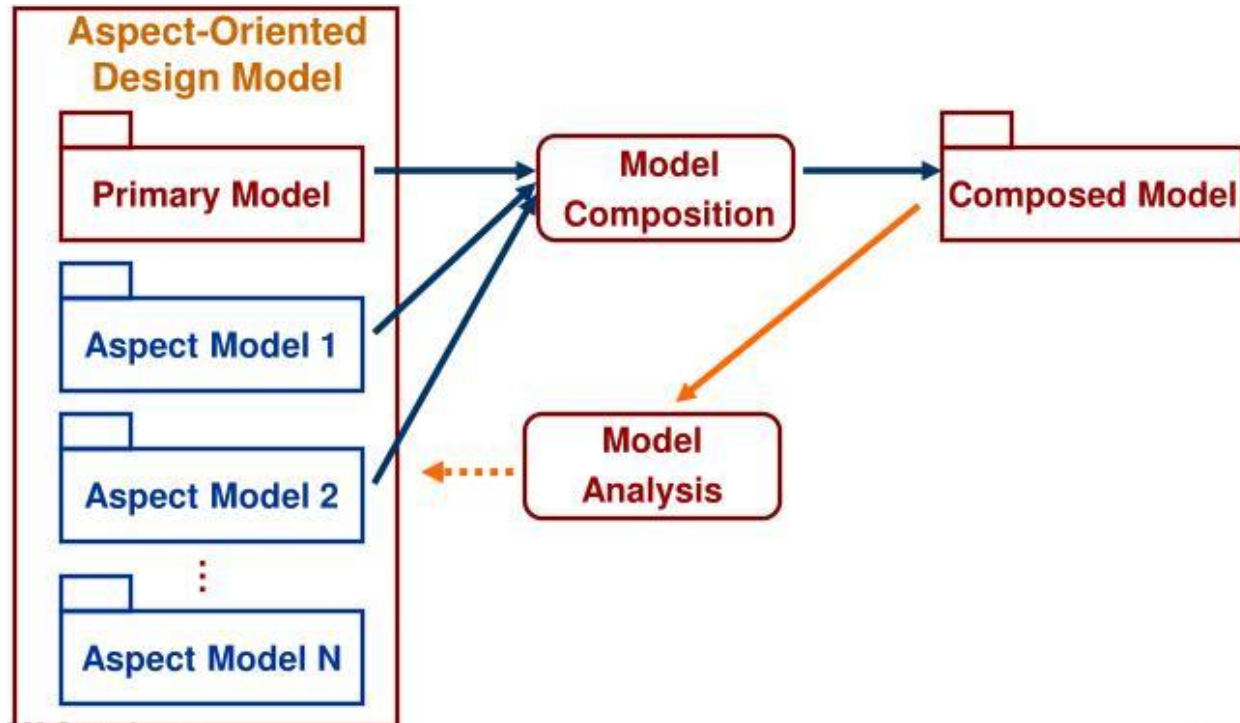


1. Kesalahan dapat dengan mudah ditemukan dan diperbaiki.
2. Bukti matematika hanya menggunakan konsep numerik.

Model *aspect oriented* dapat memberi solusi dalam pengembangan perangkat lunak ketika terdapat masalah dalam modularitas yang tidak dapat diselesaikan oleh paradigma berorientasi objek, dan struktural.



Ilustrasi model



Kekurangan model *aspect oriented*



1. Mengurangi efisiensi dari peningkatan *overhead*
2. Apabila pengkodean lintas sektoral bermasalah, maka akan menyebabkan kerusakan sistem.

Kelebihan model *aspect oriented*



1. Dianggap sebagai bagian dari teknologi pemrograman pasca-objek.
2. Dukungan desain perangkat lunak yang lebih baik melalui isolasi logika bisnis aplikasi dari fungsi pendukung dan sekunder.
3. Memberikan manfaat pelengkap dan dapat digunakan dengan proses lincah lainnya dan standar pengkodean.
4. Fokus utama - Identifikasi, representasi dan spesifikasi masalah, yang mungkin juga bersifat lintas sektoral.
5. Memberikan dukungan modularisasi desain perangkat lunak yang lebih baik, mengurangi desain perangkat lunak, biaya pengembangan dan pemeliharaan.

Referensi :

1. S, Rosa A. & Salahuddin M. 2018. *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek Edisi Revisi*. Bandung : Informatika
2. <https://medium.com/skyshidigital/manajemen-proyek-waterfall-atau-agile-mana-lebih-baik-b92901f88159>
3. <http://www.lessons99.com/prototype-model.html>
4. <https://www.scribd.com/doc/108009101/Makalah-RPL-Kekurangan-dan-kelebihan-dari-macam-macam-model-SDLC>
5. <https://sites.google.com/a/student.unsika.ac.id/metodologi-penelitian-septian-maulana-1141177004039/documents/metode-the-concurrent-development-model>
6. <http://www.tutorsglobe.com/homework-help/software-engineering/formal-method-model-7704.aspx>
7. <https://slideplayer.com/slide/4135371/>
8. <http://www.tutorsglobe.com/homework-help/software-engineering/formal-method-model-7704.aspx>



Terima kasih!

Presentasi ini bisa didapatkan
melalui tautan berikut :
<https://goo.gl/H4yqrh>

