# 3 Model Proses Perangkat Lunak

## Overview

Segala aktifitas yang berkaitan dengan Proses rekayasa perangkat lunak dapat diklasifikasikan ke dalam tiga tahapan umum, yaitu :

A. Fase Definisi (Definition Phase) berfokus pada "apa" (what), dimana pada fase ini pengembang perangkat lunak harus mengidentifikasi informasi apa yang akan diproses, fungsi dan unjuk kerja apa yang diperlukan, tingkah laku sistem seperti apa yang diharapkan, interface apa yang akan dibangun, batasan desain apa yang ada dan kriteria validasi apa yang dibutuhkan untuk mendefinisikan sistem yang sukses.

Kebutuhan (requirement) kunci dari sistem dan perangkat lunak yang didefinisikan. Metode yang diaplikasikan selama fase definisi berbeda, tergantung pada paradigma rekayasa perangkat lunak (atau kombinasi paradigma) yang diaplikasikan.

B. Fase Pengembangan (Development Phase) berfokus pada how (bagaimana), yaitu di mana selama masa pengembangan perangkat lunak, teknisi harus mendefinisikan bagaimana data dikonstruksikan, bagaimana detil prosedur akan diimplementasikan, bagaimana interface dikembangkan, dan sebagainya.

C. Fase Pemeliharaan (Maintenance Phase) berfokus pada perubahan yang dihubungkan dengan koreksi kesalahan, penyesuaian yang dibutuhkan ketika lingkungan perangkat lunak berkembang, serta perubahan sehubungan dengan perkembangan yang disebabkan oleh perubahan kebutuhan dari pengguna.

Menjadi seorang *software engineer* tidak hanya menjadi orang yang pintar akan hal algoritma, *code,* dan *desain interface (GUI)*. Akan tetapi seorang *software engineer* juga harus mempertimbangkan model pengembangan apa yang cocok diterapkan untuk projek pengembangan perangkat lunak yang sedang dijalankan. Dikarenakan, model yang digunakan akan berpengaruh pada kinerja dari tim dan juga dapat menunjang keberhasilan suatu projek pembuatan perangkat lunak.

Pada rekayasa perangkat lunak, banyak model yag telah dikembangkan untuk membantu proses pengembangan perangkat lunak, antara lain *Waterfall Model, Joint Application Development (JAD), Rapid Application Development (RAD), Information Engineering (IE), Rational Unified Process (RUP), Extreme Programming (XP), Agile Modelling, Scrum* dan lain sebagainya.

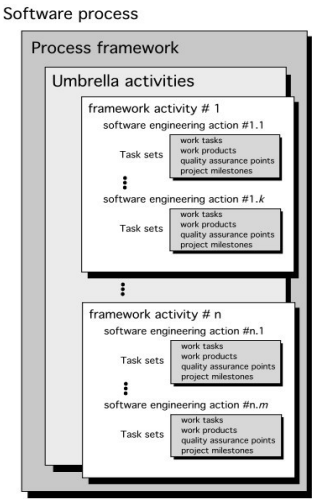
## Pengembangan Perangkat Lunak

Pengembangan perangkat lunak adalah suatu proses dimana kebutuhan pemakai diterjemahkan menjadi produk perangkat lunak. Proses ini mencakup aktivitas penerjemahan kebutuhan pemakai menjadi kebutuhan perangkat lunak, transformasi kebutuhan perangkat lunak menjadi desain, penerapan desain menjadi kode program, uji coba kode program, dan instalasi serta pemeriksaan kebenaran perangkat lunak untuk operasional.

Berdasarkan pengertian tersebut, secara umum dapat dikatakan bahwa proses pengembangan perangkat lunak mengikuti tahap-tahap:

1. Menentukan APA yang harus dikerjakan oleh perangkat lunak dalam satu rentang waktu tertentu.
2. Mendefinisikan BAGAIMANA perangkat lunak dibuat, mencakup arsitektur perangkat lunaknya, antarmuka internal, algoritma, dan sebagainya.
3. Penerapan (penulisan program) dan pengujian unit-unit program.
4. Integrasi dan pengujian modul-modul program.
5. Validasi perangkat lunak secara keseluruhan (pengujian sistem).

## Model Proses Pengembangan Perangkat Lunak




Proses adalah kerangka kerja terstruktur yang diperlukan dalam membangun suatu produk, maka Proses Perangkat Lunak merupakan sekumpulan aktivitas terstruktur yang dibutuhkan untuk mengembangkan sistem perangkat lunak berkualitas tinggi.

Kerangka pengembangan perangkat lunak *(software)* tersusun atas *umbrella activities* yang mendukung *framework activity*, dan di dalam *framework activity* terdapat sekumpulan *task set* yang merupakan unit terkecilnya.

Ada 5 proses *(process framework)* dalam Rekayasa Perangkat Lunak, yaitu:

1. *Communication* - Melakukan komunikasi dengan *stakeholders* untuk menyepakati tujuan, kebutuhan yang harus dipenuhi, serta fitur atau fungsi yang diharapkan.
2. *Planning* - Membuat rencana pekerjaan apa saja yang harus dilakukan lakukan *(software project plan)* dengan mendefinisikan tugas-tugas teknis, sumber daya yang diperlukan, keluaran *(output),* dan juga resiko yang mungkin ditemukan.
3. *Modelling* - Membuat berbagai model perangkat lunak untuk dapat menjelaskan kebutuhan *(requirement)* dengan baik dan juga rancangan yang bisa memenuhinya.
4. *Construction* - Membangun *software* sesuai rancangan, termasuk *coding* dan *testing*.
5. *Deployment* - Menyerahkan *software* kepada *user* untuk digunakan.

Ke lima proses tersebut harus dilakukan berurutan. Aktivitas di setiap proses itu dinamakan *framework activity*.

*Umbrella activity* merujuk pada setiap aktivitas pelengkap dari aktivitas di dalam proses Rekayasa Perangkat Lunak secara keseluruhan, dengan tujuan untuk mengelola kemajuan, kualitas, perubahan dan juga resiko.


Model proses perangkat lunak (disebut juga paradigma rekayasa perangkat lunak) adalah suatu strategi pengembangan yang memadukan lapisan proses, metode, dan alat serta tahap-tahap generik, mencakup kegiatan: proses perangkat lunak, produksi perangkat lunak, dan peran orang yang terlibat pada rekayasa perangkat lunak.

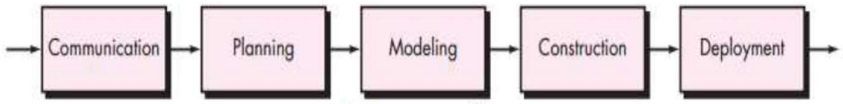
Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat proyek dan aplikasi, metode serta alat yang digunakan, juga pengendalian dan hasil yang diinginkan.

Menurut Roger Pressman, Alur proses *(process flow)* merupakan salah satu aspek penting dalam *software process.* Alur proses mengatur bagaimana seluruh *framework activities* dijalankan sesuai dengan urutan dan waktu yang telah ditentukan.

Proses-proses pada kerangka rekayasa perangkat lunak harus dilakukan sistematis, namun alur proses *(process flow)* tersebut bisa dalam berbagai bentuk. Beberapa contoh *process flow* diantaranya:

- *Linear process flow*

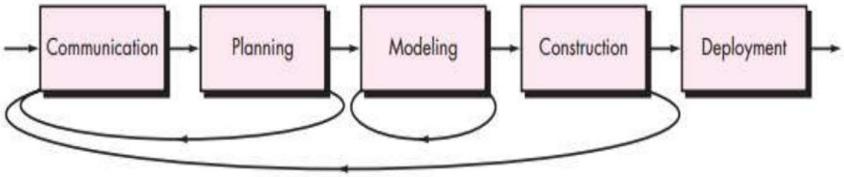
  *Linear process flow* menjalankan setiap *framework activities* secara berurutan, dimulai dari *communication* hingga *deployment.*



(a) Linear process flow

- *Iterative process flow*

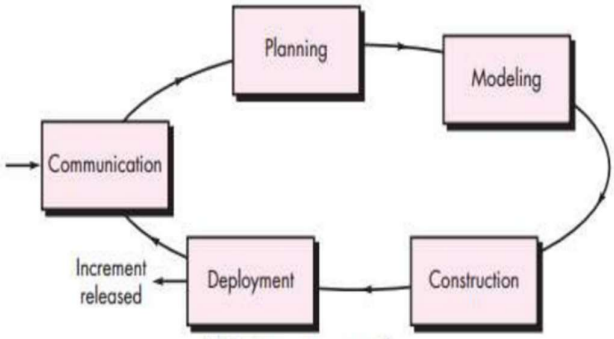
  *Iterative process flow* dapat mengulang satu atau lebih aktivitas sebelum melaksanakan aktivitas selanjutnya.



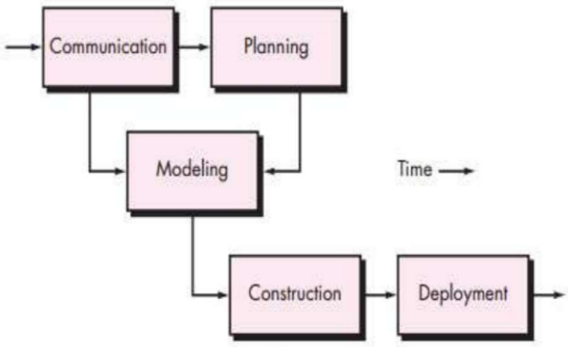
(b) Iterative process flow

- *Evolutionary process flow*

  Pelaksanaan aktivitas yang ada pada *evolutionary process flow* membentuk suatu siklus (dari *deployment* kembali menuju *communication* ). Perulangan dari tiap siklus yang menjalankan 5 aktivitas dasar tersebut akan menghasilkan *software* yang lebih baik lagi.



(c) Evolutionary process flow



(d) Parallel process flow

- *Parallel process flow*

*Parallel process flow* dapat menjalankan satu atau lebih aktivitas secara bersamaan dengan aktivitas lainnya. Sebagai contoh, *process modeling* pada satu bagian dari *software* dapat dilaksanakan secara bersamaan dengan pembuatan *(construction)* pada bagian lain dari *software* tersebut.

*Task set* ditentukan setelah menentukan alur proses yang akan digunakan, dan juga mendefinisikan *framework activity* pada setiap proses tersebut.

Jumlah dan tingkat kesulitan *task set* tergantung dari ukuran *software* yang dikembangkan. *Task set* bersifat unik antar proyek pengembangan *software*. Jika semua *task set* sudah berhasil dilakukan, maka seharusnya tujuan dari proyek perangkat lunak-nya tercapai.

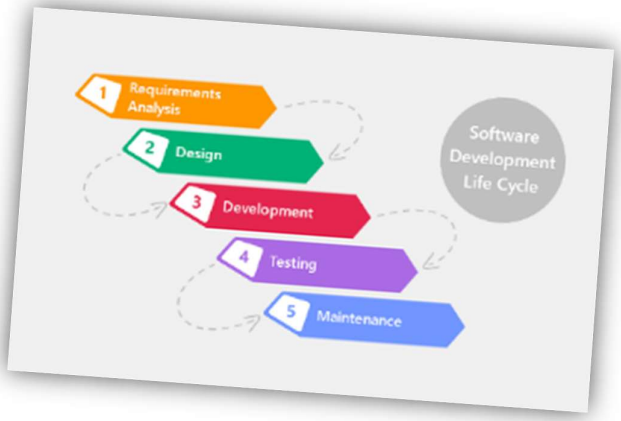
Setiap *framework activity* memiliki sekumpulan *task set* atau tugas-tugas yang harus dilakukan. Contoh *task set* pada aktivitas *requirement gathering:*

1. Membuat daftar *stakeholders.*
2. Mengundang para *stakeholders* untuk *meeting.*
3. Menanyakan fitur dan fungsi apa saja yang dibutuhkan kepada *stakeholders.*
4. Mendiskusikan *requirement* yang terkumpul dan membuat daftar *requirement* final.
5. Memberikan prioritas terhadap *requirement.*
6. Mencatat semua hal yang perlu diperjelas kembali.

## Siklus Pengembangan Perangkat Lunak

Siklus pengembangan perangkat lunak sering disebut dengan siklus hidup perangkat lunak *(System Development Life Cycle)* adalah:

- ☑ Periode waktu yang diawali dengan keputusan untuk mengembangkan produk perangkat lunak dan berakhir setelah perangkat lunak diserahkan. Umumnya siklus pengembangan ini terdiri dari tahap analisis kebutuhan, perancangan, penerapan, pengujian, dan instalasi serta pemeriksaan.
- ☑ Periode waktu yang diawali dengan keputusan untuk mengembangkan produk perangkat lunak dan berakhir saat produk tidak dapat ditingkatkan lebih jauh lagi oleh pengembang.




Beberapa model proses pengembangan perangkat lunak pada umumnya mengacu pada model proses pengembangan sistem yang disebut *System Development Life Cycle (SDLC)*.

Setiap model yang dikembangkan mempunyai karakteristik sendiri, namun secara umum ada persamaan dari model-model tersebut adalah:

- Kebutuhan terhadap definisi masalah yang jelas,
  - ☑ Input utama dari setiap model pengembangan perangkat lunak adalah pendefinisian masalah yang jelas.
  - ☑ Semakin jelas akan semakin baik karena akan memudahkan dalam penyelesaian masalah.
  - ☑ Oleh karena itu pemahaman masalah merupakan bagian penting dari model pengembangan perangkat lunak.
- Tahapan-tahapan pengembangan yang teratur,
  - ☑ Meskipun model-model pengembangan perangkat lunak memiliki pola yang berbeda-beda, biasanya model-model tersebut mengikuti pola umum:

    *" Analysis – design – coding – testing – maintenance "*
- *Stakeholder* berperan sangat penting dalam keseluruhan tahapan pengembangan,
  - ☑ *Stakeholder* dalam rekayasa perangkat lunak dapat berupa pengguna, pemilik, pengembang, pemogram dan orang-orang yang terlibat dalam rekayasa perangkat lunak tersebut.
- Dokumentasi merupakan bagian penting dari pengembangan perangkat lunak,
  - ☑ Masing-masing tahapan dalam model biasanya menghasilkan sejumlah tulisan, diagram, gambar atau bentuk-bentuk lain yang harus didokumentasi dan merupakan bagian tak terpisahkan dari perangkat lunak yang dihasilkan.
- Keluaran dari proses pengembangan perangkat lunak harus bernilai ekonomis.
  - ☑ Nilai dari sebuah perangkat lunak agak susah "dirupiahkan".
  - ☑ Namun efek dari penggunaan perangkat lunak yang telah dikembangkan haruslah memberi nilai tambah bagi organisasi.
  - ☑ Hal ini dapat berupa penurunan biaya operasi, efisiensi penggunaan sumberdaya, peningkatan keuntungan organisasi, peningkatan *"image"* organisasi dan lain-lain.

## Model Proses *'Prescriptive'*

Model proses *'prescriptive'* merupakan model yang sudah mendefinisikan dengan jelas aturan, aktivitas, tugas, *output,* dan berbagai elemen lain dalam proses rekayasa perangkat lunak.

Model proses *'prescriptive'* mengutamakan proses yang terstruktur, berurutan, dan mengikuti suatu panduan dalam setiap aktivitasnya.

Beberapa contoh yang termasuk model proses *'prescriptive'*:

- ✓ *Waterfall Model*
- ✓ *Prototyping Process Model*
- ✓ *Evolutionary / Spiral Process Model*
- ✓ *Unified Process Model*

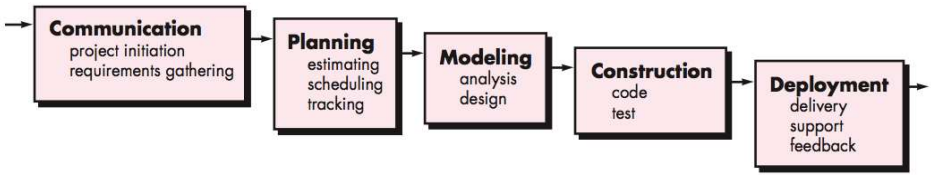
Berikut ini yang termasuk beberapa model proses *'prescriptive'*.

## 1. *Waterfall Model*

Nama model ini sebenarnya adalah *Linear process model* atau disebut juga *"classic life cycle"*. Model ini merupakan pendekatan rekayasa perangkat lunak yang paling tua, pertama kali diterbitkan pada tahun 1970.

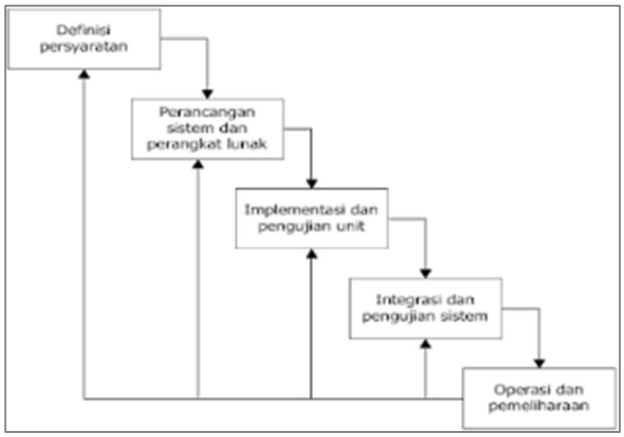
*Waterfall Model* merupakan pendekatan untuk pengembangan perangkat lunak yang alur prosesnya sistematis, berurutan, dan linier. Dimulai dari mengumpulkan kebutuhan dari *customer,* merencanakan tugas, *modelling, construction, deployment,* dan berujung pada penyediaan dukungan secara berkelanjutan ketika *software* telah selesai.

**




Beberapa masalah yang sering muncul pada saat menggunakan *Waterfall Model* adalah:

1. Proyek pengembangan *software* di dunia nyata jarang mengikuti alur proses yang berurutan dan satu arah.
2. Seringkali, pengguna kesulitan untuk mendeskripsikan semua *requirement* secara eksplisit di awal proses.
3. Pengguna (atau pelanggan) harus bersabar karena *software* baru dapat "terlihat" pada tahap akhir proyek.




*Waterfall Model* sudah sangat jarang digunakan dalam pengembangan perangkat lunak modern sekarang ini.

## 2. *Prototyping Model*

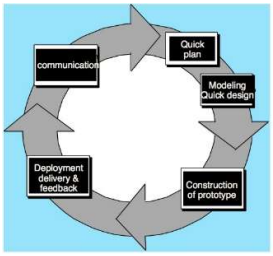

*Prototyping Process Model* adalah pendekatan yang secara langsung mendemontrasikan bagaimana komponen-komponen perangkat lunak akan bekerja dalam lingkungannya sebelum tahapan konstruksi aktual dilakukan.

Seringkali, pelanggan hanya menjabarkan kebutuhan secara umum saja, tanpa penjelasan rinci terkait fungsi atau fitur *software*. Dalam situasi seperti ini, paradigma *prototyping* bisa menjadi pendekatan yang sesuai.

*Ideally, the prototype serves as a mechanism for identifying software requirements.*

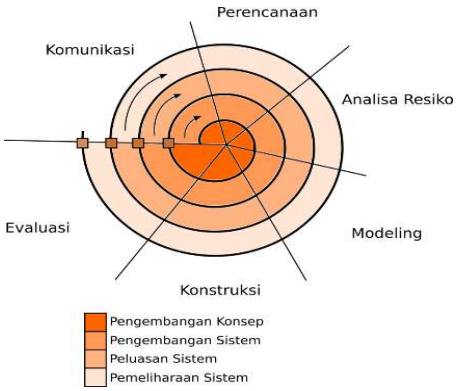
** Permasalahan atau kelemahan utama dari yang *Prototyping process model* adalah:

1. *Stakeholders* sering menganggap *prototype software* sebagai versi yang sudah berjalan (final) dan tidak menyadari bahwa kualitasnya secara keseluruhan belum terjamin.
2. Sebagai *software engineer,* kita sering mengimplementasi *software* yang mengabaikan banyak aspek (misal kualitas dan keamanan) demi menghasilkan *prototipe* yang cepat.



Kuncinya adalah dengan membuat kesepakatan di awal, dan semua *stakeholders* harus setuju bahwa *prototipe software* dibuat untuk mendapatkan *requirement software* versi akhirnya nanti.

### 3. *Evolutionary / Spiral Process Model*




*Spiral Process Model* adalah pendekatan yang realistis untuk mengembangkan sistem dan *software* berskala besar.

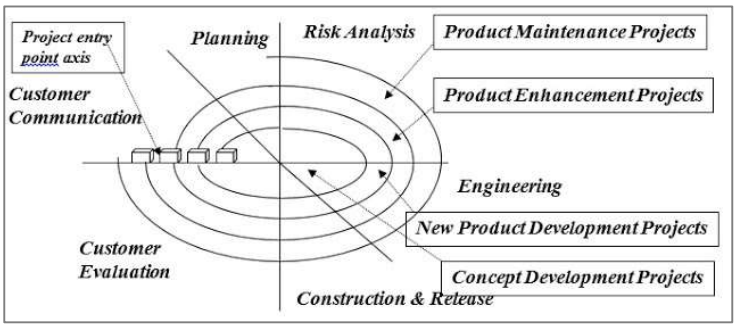
Dalam model ini, *software* dikembangkan dalam suatu seri *incremental release,* yang bersifat *evolutionary.* Setiap versi rilis mengikuti aktivitas kerangka kerja yang ditentukan oleh tim pengembang dimana presentasi diagramnya berbentuk spiral dengan beberapa perulangan.

Setiap putaran dari perulangan tersebut menggambarkan langkah dari proses-proses pengembangan. Setiap perulangan dari spiral disebut fase dari pengembangan perangkat lunak.

Pada iterasi awal, *software* yang dirilis mungkin saja masih berbentuk model (prototipe), kemudian berkembang pada iterasi selanjutnya hingga versi terbaik *software* didapatkan.

*Spiral Process Model* dibagi menjadi 6 aktivitas kerangka kerja, sebagai berikut:

1. Komunikasi dengan pemakai
2. Perencanaan
3. Analsa resiko
4. Rekayasa model
5. Konstruksi dan pelepasan
6. Evaluasi

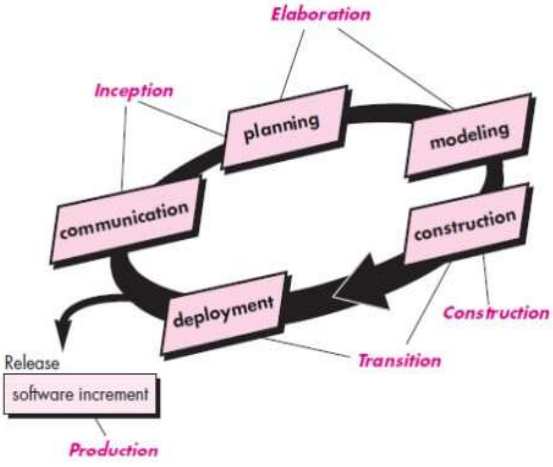



** Permasalahan pada *Spiral Process Model* – sulit meyakinkan pelanggan (pengguna) bahwa pendekatan ini dapat terukur dan terkendali.

### 4. *Unified Process Model*

*Unified Process Model* mengakui pentingnya komunikasi dengan pengguna, dan memilih metode yang disederhanakan untuk menjelaskan sistem dari sudut pandang pengguna.

- ☑ *The inception phase* - melibatkan komunikasi dengan pengguna dalam perencanaan aktivitas.
- ☑ *The elaboration phase* - menggunakan aktivitas-aktivitas komunikasi dan *modelling* seperti *use case,* dsb.
- ☑ *The construction phase* - tahap pengembangan *software* (misalnya: *coding).*
- ☑ *The transition phase* - transisi dari tahap pengembangan *software* ke tahap implementasi (misalnya *release candidate* atau *beta version*).
- ☑ *The production phase* - penyerahan *software* kepada pelanggan (pengguna). *Software* sudah digunakan dan penggunaannya dipantau.



## MEMILIH METODOLOGI PENGEMBANGAN YANG TEPAT

Beberapa pertimbangan pemilihan metodologi pengembangan perangkat lunak yang tepat terdiri dari beberapa kriteria meliputi:

- ✓ kejelasan kebutuhan pengguna *(clarity user requirement),*
- ✓ penguasaan teknologi *(familiarity with technology),*
- ✓ tingkat kerumitan sistem *(system complexity),*
- ✓ tingkat kehandalan sistem *(system realibility),*
- ✓ waktu pelaksanaan *(short time schedules),* dan
- ✓ visibilitas jadwal pelaksanaan *(schedule visibility).*

| Kriteria pengembangan sistem | Waterfall | Parallel | V-Model | Iterative | System Prototyping | Throw away Prototyping | Agile Development |
|---|---|---|---|---|---|---|---|
| Kejelasan kebutuhan pengguna | Buruk | Buruk | Buruk | Baik | Baik Sekali | Baik Sekali | Baik Sekali |
| Penguasaan teknologi | Buruk | Buruk | Buruk | Baik | Buruk | Baik Sekali | Buruk |
| Tingkat kerumitan sistem | Baik | Baik | Baik | Baik | Buruk | Baik Sekali | Buruk |
| Tingkat kehandalan sistem. | Baik | Baik | Baik sekali | Baik | Buruk | Baik sekali | Baik |
| Waktu pelaksanaan. | Buruk | Baik | Buruk | Baik Sekali | Baik Sekali | Baik | Baik Sekali |
| Visibilitas jadwal pelaksanaan. | Buruk | Buruk | Buruk | Baik Sekali | Baik Sekali | Baik | Baik |

# Perbandingan Antar Model

| Model | Pro | Kontra |
|---|---|---|
| ***Waterfall*** | • Mudah untuk dipahami dan direncanakan.<br>• Efektif untuk proyek kecil yang mudah dipahami kebutuhannya.<br>• Proses analisis dan testing mudah dilakukan. | • Sulit menghadapi perubahan requirement.<br>• Testing di lakukan di tahap akhir proyek.<br>• Persetujuan **pengguna** hanya di akhir. |
| ***Prototyping*** | • Meminimalkan dampak perubahan kebutuhan.<br>• Pengguna sering terlibat di awal.<br>• Efektif untuk proyek yang kecil.<br>• Mengurangi resiko penolakan proyek. | • Keterlibatan pengguna bisa mengakibatkan keterlambatan.<br>• Ada kecenderungan untuk memberikan software yang masih berupa prototipe.<br>• Sulit untuk direncanakan dan dikelola. |
| ***Spiral*** | • Ada keterlibatan pengguna secara berkelanjutan.<br>• Dapat mengelola resiko-resiko pengembangan software dengan baik.<br>• Efektif untuk proyek yang besar dan rumit. | • Kesalahan dalam menganalisis resiko bisa menggagalkan proyek.<br>• Sulit untuk mengelola pekerjaan di dalamnya.<br>• Membutuhkan tim pengembang yang sudah ahli. |
| ***Unified Process*** | • Kualitas dari dokumentasi diperhatikan<br>• Ada keterlibatan pengguna secara berkelanjutan<br>• Merespon perubahan requirement dengan baik<br>• Efektif untuk proyek yang bersifat maintenance. | • Use case yang dibuat tidak selalu sesuai.<br>• Cukup sulit mengintegrasikan software increment<br>• Tahap yang tumpang tindah bisa jadi masalah.<br>• Membutuhkan tim pengembang yang sudah ahli. |

| No | Metodologi | Kelebihan | Kelemahan |
|---|---|---|---|
| 1 | **Linear Sequential Model** | 1) Mudah dalam pengelolaan karena hampir seluruh requirements telah diidentifikasikan dan didokumentasikan,<br>2) Tahapan yang berurutan secara linier, identifikasi dan dokumentasi yang lengkap, menyebabkan proses mudah dipahami oleh seluruh tim yang terlibat ataupun project owner. | 1) Tahapan yang berurutan secara linier tidak memungkinkan untuk kembali pada tahapan selanjutnya,<br>2) Tidak fleksibel terhadap perubahan kebutuhan yang terjadi dalam tahap pengembangan sistem,<br>3) Hampir tidak ada toleransi kesalahan, terutama pada tahapan planning dan design. |
| 2 | **Parallel Model** | Waktu pengembangan sistem yang lebih singkat dibandingkan waterfall, karena beberapa tahapan diakselerasikan dengan membagi menjadi beberapa sub project. | 1) Integrasi sistem memiliki kesulitan tersendiri. Kegagalan atau keterlambatan pada salah satu sub project memberikan dampak pada proses mengintegrasikan seluruh sistem,<br>2) Terdapat kemungkinan kesulitan dalam penanganan jika terjadi permasalah pada sub project secara bersamaan. |
| 3 | **Iterative Model** | 1) Umpan balik terus menerus dari pemilik proyek,<br>2) Beberapa revisi pada seluruh aplikasi dan fungsi spesifik,<br>3) Pekerjaan disampaikan di awal proyek. | Setiap perulangan adalah struktur kaku yang menyerupai project kecil waterfall. |
| 4 | **Prototyping Model** | 1) Requirements identification yang akurat karena dilakukan evaluasi secara berkala dan mendapatkan masukan dari project owner terhadap purwa rupa yang dihasilkan,<br>2) User experience yang meningkat, karena secara terus menerus melakukan uji coba dan evaluasi terhadap purwa rupa,<br>3) Kesalahan dan redudansi dapat diminimalkan karena proses identifikasi yang baik terhadap purwa rupa. | 1) Setiap evaluasi dan masukan terhadap purwa rupa, maka akan membutuhkan penyesuaian terhadap purwa rupa tersebut. Dan setiap penyesuaian akan meningkatkan kompleksitas sistem yang dikembangkan,<br>2) Memberikan beban tambahan kepada programmer,<br>3) Terdapat kebutuhan biaya tambahan terkait dengan pembuatan purwa rupadan dapat dilakukan penyesuaian versi purwa rupa sesuai kebutuhan, hingga purwa rupa dapat disetujui oleh project owner. |

| | | | |
|---|---|---|---|
| **5** | **RAD (Rapid Application Development) Model** | 1) Efisiensi waktu pengiriman,<br>2) Perubahan kebutuhan dapat ditampung,<br>3) Waktu siklus dapat pendek dengan penggunaan alat-alat RAD yang kuat,<br>4) Produktivitas dengan lebih sedikit orang dalam waktu singkat,<br>5) Penggunaan alat-alat dan kerangka kerja. | 1) Kompleksitas manajemen,<br>2) Cocok untuk sistem yang berbasis komponen dan terukur,<br>3) Membutuhkan keterlibatan pengguna di seluruh siklus hidup,<br>4) Membutuhkan personal yang sangat terampil,<br>5) Ketergantungan tinggi pada kemampuan modeling,<br>6) Tidak berlaku untuk proyek-proyek yang lebih murah sebagai biaya pemodelan dan otomatis generasi kode sangat tinggi untuk proyek-proyek yang dianggarkan lebih murah untuk pantas. |
| **6** | **Spiral Model** | 1) Jumlah analisis risiko yang tinggi,<br>2) Baik untuk proyek-proyek besar dan mission-critical,<br>3) Software diproduksi di awal siklus hidup perangkat lunak. | 1) Dapat menjadi model mahal untuk digunakan,<br>2) Analisis risiko membutuhkan keahlian yang sangat spesifik,<br>3) Keberhasilan proyek sangat tergantung pada tahap analisis risiko,<br>4) Tidak bekerja dengan baik untuk proyek-proyek yang lebih kecil. |
| **7** | **V-Shaped Model** | 1) Sederhana dan mudah digunakan,<br>2) Setiap fase memiliki delivery tertentu,<br>3) Kesempatan keberhasilan yang lebih tinggi atas model waterfall karena perkembangan awal dari rencana pengujian selama siklus hidup,<br>4) Bekerja dengan baik untuk proyek-proyek kecil di mana persyaratan yang mudah dipahami. | 1) Sangat kaku seperti model waterfall,<br>2) Sedikit fleksibilitas dan ruang lingkup menyesuaikan sulit dan mahal,<br>3) Software dikembangkan selama tahap implementasi, sehingga tidak ada prototipe awal dari perangkat lunak yang dihasilkan,<br>4) Model ini tidak memberikan jalan yang jelas untuk masalah yang ditemukan selama pengujian tahap. |
| **8** | **Agile Development** | 1) Metode ringan sesuai proyek ukuran kecil-menengah,<br>2) Menghasilkan kohesi tim yang baik, | 1). Tidak cocok untuk menangani dependensi yang kompleks, |

| | | | |
|---|---|---|---|
| | | 3) Menekankan produk akhir,<br>4) Berulang,<br>5) Pendekatan berbasis tes untuk persyaratan dan jaminan kualitas. | 2). Lebih risiko keberlanjutan, perawatan dan diperpanjang,<br>3). Sebuah rencana keseluruhan, pemimpin lincah dan manajemen proyek tangkas praktek adalah suatu keharusan tanpa yang tidak akan bekerja. |