# Merencanakan Sebuah Sistem

Bagaimana perkembangan software melahirkan Rapid Application Development

**Zainal / Rafli / Sylvano / Vincent / Kal-el**


Rekayasa Perangkat Lunak
Sekolah Menengah Kejurusan Letris 2
10 September 2023

# Contents

# 1 Pendahuluan

Abad ke-21 adalah abad yang tiada duanya.

Di mana revolusi industri melahirkan tingkat produktivitas dan pertumbuhan yang belum pernah terjadi pada sejarah peradaban manusia, menaklukkan alam pada kehendak manusia, yang dapat membuat nenek moyang primata kita menjerit dan meringis dalam kekaguman dan ketakutan akan kekuatan layaknya seorang dewa.

Perkembangan ilmu pengetahuan dan teknologi nuklir merupakan puncak dari pengetahuan ilmiah abad ke-20, umat manusia tidak lagi hanya mampu memanfaatkan kekuatan batu bara dan baja untuk menggerakkan mesin-mesin perkasa, tetapi fondasi dasar dari alam semesta materi kita sendiri, atom, berada di genggam tangan kita dengan memungkinkannya kekuatan nuklir untuk penggunaan manusia... untuk baik atau tidak.

Tidak ada seorang pun yang dapat menghindari gelombang kemajuan industy modern yang tak tertahankan. Pada saat itu, anda tentu tidak akan salah jika berpikir bahwa umat manusia telah mencapai klimaks akhir sebagai peradaban.

Karena itu, bahwa perkembangan industri modern berada dalam keadaan yang spesial dibanding pada masa-masa yang lalu. Dari latar belakang itulah, munculnya kepentingan software pada pengembangan sistem informasi. Pengunaan data yang baik sangat penting untuk pengembangan sistem informasi yang berkualitas, dari *requirement analysis, client feedback, debugging, dll* merupakan fondasi dasar dari perkembangan sistem informasi modern.

Belum lama ini, Watt S. Humphrey, yang dikenal sebagai bapak kualitas dalam software, mengatakan: **"Setiap bisnis adalah bisnis software."** Baru-baru ini, CEO Microsoft, Satya Nadella, mengulangi kutipan tersebut: **"Setiap perusahaan adalah perusahaan software".**[10] Tentu saja, kita bisa menunjuk ke banyak perusahaan teknologi tertentu yang mengembangkan software. Jika ada sebuah aplikasi, pasti ada yang mengembangkannya. Namun organisasi bisnis yang tidak "bergerak di bidang software" mengandalkan perangkat lunak dan teknologi untuk menjalankan bisnisnya (dengan kata lain, semuanya). Organisasi-organisasi ini perlu mengadaptasi setidaknya beberapa solusi *off-the-shelf*, kemungkinan besar akan menyesuaikan software untuk menyelaraskan dan mengoptimalkan dengan operasi bisnis mereka yang unik.

Dari kebutuhan akan lebih banyaknya software pada perkembangan semua bidang industri, muncullah *Systems Development Life Cycle* (SDLC), yang berguna untuk membantu pembuatan software dengan jaminan kualitas dan ketetapan deadline dengan struktur yang jelas yang memastikan perkembangan software berjalan sesuai dengan keinginan klien, waktu, dan funding.

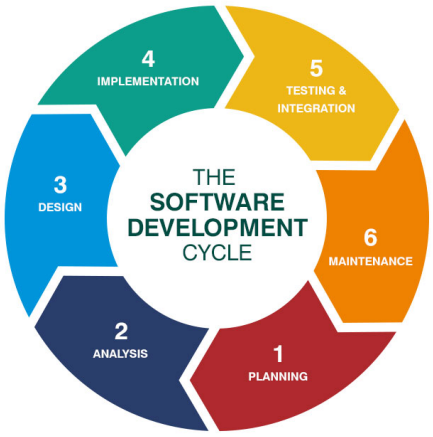
# 2 SDLC

## 2.1 Apa itu SDLC?

SDLC atau *Systems Development Life Cycle*, atau dalam bahasa Indonesia disebut Siklus Hidup Pengembangan Sistem. SDLC adalah siklus yang digunakan dalam pembuatan atau pengembangan sistem informasi yang bertujuan untuk menyelesaikan masalah secara efektif. Dalam pengertian lain, SDLC adalah tahapan kerja yang bertujuan untuk menghasilkan sistem berkualitas tinggi yang sesuai dengan keinginan pelanggan atau tujuan dibuatnya sistem tersebut. SDLC menjadi kerangka yang berisi langkah-langkah yang harus dilakukan untuk memproses pengembangan suatu perangkat lunak. Sistem ini berisi rencana lengkap untuk mengembangkan, dan memelihara hasil produk software yang jadi.[1]

## 2.2 Fungsi SDLC

SDLC digunakan untuk membangun suatu sistem informasi agar dapat berjalan sesuai dengan apa yang diharapkan. SDLC (Systems Development Life Cycle) atau Systems Life Cycle, dalam rekayasa sistem dan rekayasa perangkat lunak, adalah proses pembuatan dan pengubahan sistem serta model dan metodologi yang digunakan untuk mengembangkan sistem-sistem tersebut. Konsep ini umumnya merujuk pada sistem komputer atau informasi. SDLC juga merupakan pola yang diambil untuk mengembangkan sistem perangkat lunak, yang terdiri dari




tahap-tahap:

1. *Planning* (rencana)
2. *Analysis* (analysis)
3. *Design* (design)
4. *Implementation* (penerapan)
5. *Testing* (uji coba)
6. *Maintenance* (pengelolaan)

Dalam rekayasa perangkat lunak, konsep SDLC mendasari berbagai jenis metodologi pengembangan perangkat lunak. Metodologi-metodologi ini membentuk suatu kerangka kerja untuk perencanaan dan pengendalian pembuatan sistem informasi, yaitu proses pengembangan perangkat lunak. Terdapat 3 jenis metode siklus hidup sistem yang paling banyak digunakan, yakni:

- *Traditional System Life Cycle*
- *Life Cycle Using Prototyping*
- *Object-oriented System Life Cycle*

# 3 Sejarah Singkat

*Kelahiran suatu pikiran sering menyamai kelahiran seorang anak. Ia didahului dengan penderitaan-penderitaan pembawaan kelahirannya.*[8]

Untuk memastikan kita benar-benar memahami suatu hal, kita harus memahami konteks historis bagaimana hal itu terlahirkan, konsekuensi dari perkembangannya, dan kontradiksi internal yang mendahului perkembangannya.

## 3.1 Kelahirannya

Menariknya, sejarah SDLC tidak sedalam sejarah perkembangan *software development*. Kerangka framework, "SDLC" yang mempertimbangkan struktur tahapan yang terlibat dalam pengembangan aplikasi dari studi kelayakan awal hingga penerapannya di lapangan dan pemeliharaan, mulai dikenal dengan model waterfall. Deskripsi formal pertama dari model waterfall dikutip dalam sebuah artikel tahun 1970 oleh Winston W. Royce.[5]

Model Waterfall menyediakan model yang terorganisir dan terkendali untuk mengelola proyek; model ini berkembang secara linear melalui fase-fase yang diskrit, logis, dan dapat dijelaskan, sehingga mudah dipahami dan diimplementasikan. Model ini menyediakan pencapaian yang mudah diidentifikasi dalam proses pengembangan.

SDLC "berasal dari tahun 1960-an, untuk mengembangkan sistem bisnis fungsional berskala besar di era konglomerat bisnis berskala besar. Aktivitas sistem informasi berkisar pada pemrosesan data yang berat dan rutinitas penghitungan angka.".[6, see p.86]

*Structured Systems Analysis and Design Method* (SSADM) dibuat untuk Kantor Perdagangan Pemerintah Inggris pada tahun 1980-an. Sejak saat itu, "pendekatan siklus hidup tradisional untuk pengembangan sistem semakin digantikan dengan pendekatan dan kerangka kerja alternatif, yang berusaha mengatasi beberapa kekurangan yang melekat pada SDLC tradisional"[6, see p.86]

## 3.2 Konsekuensi Era yang Berbeda

Agar kita dapat memahami alasan utama dari kekurangan yang melekat pada metodologi SDLC tradisional, kita harus menempatkan pengembangan SDLC ke dalam konteks sejarah yang tepat di mana metodologi tersebut muncul.

Perkembangan SDLC kurang lebih bersamaan dengan perkembangan *Software Engineering* sebagai sebuah disiplin ilmu. Dan karena itu, karena *Software Engineering* awal harus tumbuh dari bidang lain untuk membentuk dirinya sendiri sebagai disiplin ilmu yang tepat, maka secara alamiah harus tumbuh dari bidang lain yang terkait seperti Matematika, Teknik, Fisika, dll.

Upaya awal untuk tumbuh dengan sendirinya ini adalah salah satu alasan mengapa metodologi SDLC awal mengikuti jalur linier yang kaku dan ketat untuk model pengembangannya. Se-

bagai warisan dari nenek moyangnya, pada awalnya, sifat asli perangkat lunak belum sepenuhnya disadari. Fleksibilitas, kecepatan, kelincahan, dan kekuatan komputasi yang dapat dihasilkannya, bukanlah karakteristik perangkat lunak yang dimanfaatkan sepenuhnya oleh para *Software Engineers* dahulu.

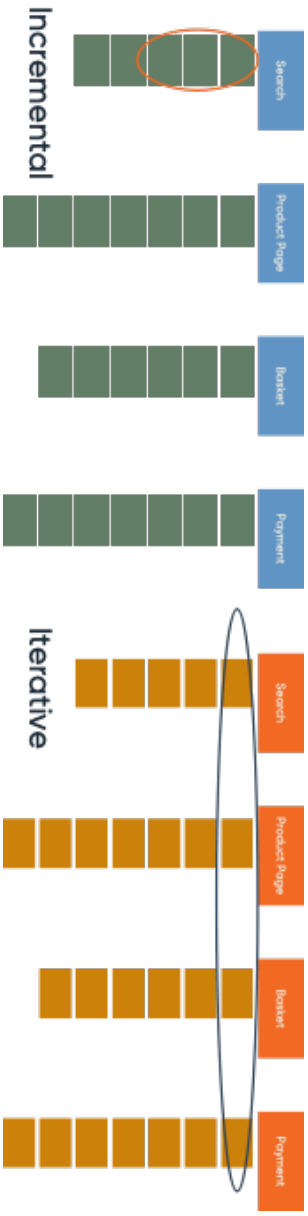
Pengembangan software awal sebagian besar terdiri dari proyek-proyek pemerintah yang sangat terspesialisasi, dengan proyek-proyek seperti Proyek SAGE, Apollo Guidance Computer, dll.

## 3.3 Era Baru

Ketika fokusnya bergerak ke arah *Customer Centricity*, model Waterfall mulai mendapat kritik karena modelnya yang bertahap secara linier, yang tidak memungkinkan adanya fleksibilitas terhadap perubahan pelanggan di tengah-tengah pengembangan software dan dalam banyak kasus memiliki waktu yang lama, yang mengakibatkan lamanya waktu penyelesaian.

Pengembangan iteratif diciptakan sekitar tahun 1975 sebagai respons terhadap inefisiensi dan masalah yang ditemukan dalam model Waterfall. Iteratif dan Inkremental sering kali saling melengkapi dan dalam beberapa proyek, keduanya digunakan bersamaan. Fondasi dasar metode ini mendukung pengembangan sistem melalui siklus berulang (Iteratif) dan dalam subset yang lebih kecil dalam satu contoh (Inkremental), sehingga memungkinkan tim untuk memanfaatkan pembelajaran dari fase sebelumnya dan berimprovisasi dalam iterasi saat ini.

Walau model iteratif dan inkremental saling melengkapi dan berada dalam project yang sama, kedua model itu tetap mempunyai posisi yang berbeda pada suatu project, Berikut adalah visualisasi perbedaan antara model iteratif dan inkremental :



Evolusi SDLC terus berlanjut, dengan fokus pada kelincahan dalam upaya untuk mengurangi penyelesaian, membangun produk yang layak sambil menjaga pelanggan tetap menjadi pusat dari segalanya. Hal ini membutuhkan paradigma perubahan yang mengedepankan individu dan interaksi daripada proses dan alat, software yang berfungsi daripada dokumen yang komprehensif, kolaborasi pelanggan melalui semua fase, dan merespons perubahan daripada mengikuti rencana. Kebutuhan saat itu adalah memiliki metode yang lebih fleksibel yang dapat memenuhi kebutuhan tersebut. Metode Agile muncul sebagai pengembangan langsung dari metode perangkat lunak dari tahun 1980-an, yaitu

- *Joint Application Design* (1986)
- *Rapid Systems Development* (1987)
- *Rapid Application Development* (1991).

# 4 Rapid Application Development

Sebuah SDLC yang terlahir dari kekuasaan tirranis waterfall. *Rapid Application Development* atau RAD, bertujuan untuk memberi alternatif pada waterfall yang lebih flexible, user oriented, dan pengembangan iterative, dimana model RAD memungkinkan pengembangan *software* yang lebih cepat dibanding model waterfall.

## 4.1 Rencana Baru

Rapid Application Development adalah metodologi pengembangan perangkat lunak yang diperkenalkan pada tahun 1990-an dan disajikan dalam bentuk buku *Rapid Application Development* oleh ahli teknologi informasi James Martin. Sebuah reaksi terhadap metodologi yang sudah mapan saat itu yang menekankan pada pengumpulan persyaratan yang cermat dan berkepanjangan sebelum pengembangan perangkat lunak yang sebenarnya dimulai.[3]

Walau James Martin bukan "pencipta" dari SDLC RAD, ada beberapa contoh penggunaan prinsip-prinsip RAD seperti Barry Boehman dengan pembuatan model Spiral. James Martin lah yang menformalisasikan RAD sebagai model SDLC tersendiri.

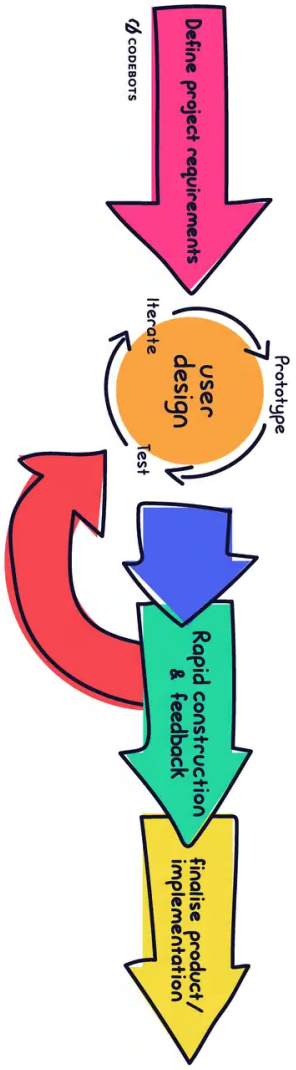
## 4.2 Apa Itu RAD?

Rapid Application Development (RAD) atau *Rapid Prototyping* adalah model proses pembangunan perangkat lunak yang tergolong dalam teknik incremental (bertingkat). RAD menekankan pada siklus pembangunan pendek, singkat, dan cepat. Waktu yang singkat adalah batasan yang penting untuk model ini. Rapid application development menggunakan metode iteratif (berulang) dalam mengembangkan sistem dimana *working model* sistem dikonstruksikan di awal tahap pengembangan dengan tujuan menetapkan kebutuhan (requirement) user dan selanjutnya disingkirkan. Working model digunakan kadang-kadang saja sebagai

basis desain dan implementasi sistem final.[7]

## 4.3 Sruktur

Fondasi RAD sebagai system yang sangat flexible, terlihat sangat jelas disaat kita melihat *Life Cycle* model tersebut[4] :




1. *Define project requirements*:
   **Fase awal dimana semua pihak terkait mendefinisikan requirements yang dibutuhkan secara besar.**

2. *Prototype*:
   **proses interaktif kontinu yang memungkinkan pengguna untuk memahami, memodifikasi, dan pada akhirnya menyetujui model kerja sistem yang memenuhi kebutuhan mereka.**

3. *Rapid Construction and Feedback Gathering*:
   **Rapid Construction adalah tempat pengkodean aplikasi, pengujian sistem, dan integrasi unit, mengubah prototipe dan sistem beta menjadi model kerja.**

4. *Finalise product / implementation*:
   **Fase terakhir dari Rapid Application Development, di mana developer mengatasi utang teknis yang timbul dalam pembuatan prototipe awal, mengoptimalkan implementasi untuk meningkatkan stabilitas dan maintenance saat mereka menyelesaikan produk untuk diluncurkan.**

## 4.4 RAD vs Tradisi

Table berikut akan menjelaskan perbedaan sistem RAD dengan sistem SDLC tradisional[2] :

| Parameter | RAD | Tradisional SDLC |
|---|---|---|
| Proses Pengembangan Applikasi | Inkremental dan Iteratif | Linear dan Prediktif |
| Struktur Team | Tim yang kecil dan flexible | Tim yang besar dan kaku |
| Produktivitas dan Flexibilitas | Tinggi : struktur iteratif | Rendah : struktur linear |
| Dokumentasi | Minimalist | Full Spesifikasi |
| Waktu dan Estimasi Harga | Variable | Fixed |
| Uji Coba | Extensive | Minimal |
| Interaksi Pengguna Akhir | Setiap fase development | Hanya awal |

Dari tabel tersebut, kita bisa menyimpulkan bahwa perbedaan paling dominan dari kedua sistem tersebut antara flexibitas dengan keteguhan, iterasi dengan lineariti, dan komunikasi konstant dengan komunikasi kecil.

## 4.5 Kelebihan/Kekurangan RAD

Berikut adalah beberapa kelebihan sistem RAD:[9, See p.75]

1. Proses pengiriman menjadi lebih mudah, hal ini dikarenakan proses pembuatan lebih banyak menggunakan potongan - potongan script.
2. Mudah untuk diamati karena menggunakan model prototype, sehingga user lebih mengerti akan sistem yang dikembangkan.
3. Lebih fleksibel karena pengembang dapat melakukan proses desain ulang pada saat yang bersamaan.
4. Mampu meminimalkan kesalahan-kesalahan dengan menggunakan alat-alat bantuan (CASE tools).
5. Mempercepat waktu pengembangan sistem secara keseluruhan karena cenderung mengabaikan kualitas.

Berikut adalah beberapa kekurangan sistem RAD:

1. Membutuhkan komitmen tingkat tinggi dari semua pihak.
2. Memerlukan sistem modular dan sulit untuk proyek berskala besar.
3. Ketelitian menjadi berkurang karena tidak menggunakan metode yang formal dalam melakukan pengkodean.
4. Kesulitan melakukan pengukuran mengenai kemajuan proses.
5. Menuntut fokus antarmuka karena fokusnya pada protoyping.

## 4.6 Untuk Siapa sih?

RAD adalah metode yang baik untuk lingkungan yang bergerak cepat dengan tim berpengalaman yang memiliki anggaran untuk alat pengembangan aplikasi yang cepat, seperti *Low Code Platform* dan generator kode. RAD sangat berguna untuk usaha kecil yang menghasilkan produk inovatif di pasar yang kompetitif dan membutuhkan keterlibatan bisnis yang tinggi. Pendekatan *on-the-fly* mengakomodasi perubahan kebutuhan yang tidak terduga. Ketika proyek memiliki tenggat waktu yang ketat, metode pengembangan aplikasi yang cepat membuat tim bertanggung jawab untuk memberikan produk yang berfungsi secepat mungkin.

Pertimbangkan daftar berikut ini untuk menilai apakah sistem RAD tepat untuk Anda.[4]

1. Apakah Anda memiliki tim yang berpengalaman yang dapat berkomitmen pada proses pengembangan yang intensif dan berkelanjutan yang melibatkan komunikasi tingkat tinggi?

2. Apakah klien Anda terbuka terhadap sistem ini, dan tingkat keterlibatan langsung yang diperlukan? Jika tidak, gangguan komunikasi dapat menyebabkan proyek gagal.

3. Apakah klien Anda bersedia mematuhi jadwal proyek dan jadwal penyelesaian model? Semua pihak berkepentingan harus terlibat untuk menerapkan metodologi ini secara efektif.

4. Dapatkah sistem dimodularisasi dalam jangka waktu 2-3 bulan?

5. Apakah Anda memiliki alat komunikasi dan pengembangan serta perangkat lunak yang tepat untuk menerapkan RAD secara efektif? Jika tidak, apakah Anda memiliki anggaran untuk membeli alat yang dibutuhkan, termasuk anggota tim ahli tambahan jika diperlukan?

# 5 Kesimupulan

Berdasarkan pembahasan artikel ini, dimulai dari asal usul SDLC, konteks perkembangannya, dan bagaimana akhirnya SDLC RAD perkembang sebagai sistem tersendiri. Kita bisa menyimpulkan bahwa SDLC adalah suatu hal yang sangat flexible, kita bisa melihatnya dari seberapa banyaknya sistem-sistem SDLC yang sudah terlahir dari 20 tahun terakhir saja, yang pada akhirnya, semua bertujuan untuk menyelesaikan masalah yang sama. Yaitu bagaimana kita bisa menyelesaikan sebuah masalah, menggunakan software, dengan struktur yang jelas, tepat, dan realistis, sehingga development solusi tidak terjebak pada *Development Hell* dimana, bagaikan Sisyphus yang mendorong batu perkasa diatas bukit hanya untuk batunya jatuh dan menibannya dalam kesiksaan, perkembangan software juga bisa terjebak dalam pengulangan tak berhenti.

# References

[1] Anonymous. *Memahami System Development Life Cycle*. 2020. URL: https://accounting.binus.ac.id/2020/05/19/memahami-system-development-life-cycle/.

[2] Anonymous. *RAD vs Traditional SDLC*. November 23, 2020. URL: https://www.wavemaker.com/rapid-application-development-vs-traditional-sdlc/.

[3] Anonymous. *Rapid Application Development*. URL: https://www.bestpricecomputers.co.uk/glossary/rapid-application-development.htm.

[4] Christine Chien. *What is Rapid Application Development (RAD)?* 4February 2020. URL: https://codebots.com/app-development/what-is-rapid-application-development-rad.

[5] Shantanu Choudhary. *Evolution of System Development Life Cycle (SDLC)*. 2018. URL: https://www.linkedin.com/pulse/evolution-system-development-life-cycle-sdlc-shantanu-choudhary.

[6] Geoffrey Elliot. *Global business information technology : an integrated systems approach*. Ed. by Geoffrey Elliot. Harlow, England ; New York : Pearson Addison Wesley, 2004.

[7] Taufik Ismail. *Perbedaan SDLC, RAD, dan JAD*. 2011. URL: https://anakkos21.blogspot.com/2011/11/perbedaan-sdlc-rad-dan-jad.html.

[8] Tan Malaka. *Naar de 'Republiek Indonesia'*. 1925. URL: https://www.marxists.org/indonesia/archive/malaka/1925-Menuju.htm.

[9] Agustinus Noertjahyana. "STUDI ANALISIS RAPID APLICATION DEVELOPMENT SEBAGAI SALAH SATU ALTERNATIF METODE PENGEMBANGAN PERANGKAT LUNAK". In: *Jurnal Informatika, OJS PETRA* Vol 3.No. 2 (2004-06-21).

[10] Muhammad Raza. *The Software Development Lifecycle (SDLC): An Introduction*. 2021. URL: https://www.weforum.org/agenda/2021/05/world-data-produced-stored-global-gb-tb-zb/.