

Siklus Hidup Pengembangan Sistem (SDLC)

Siklus Hidup Pengembangan Sistem (SDLC) adalah proses yang digunakan untuk merancang, mengembangkan, dan menguji perangkat lunak berkualitas tinggi. SDLC bertujuan untuk menghasilkan sistem yang memenuhi atau melampaui harapan pelanggan, selesai tepat waktu dan dalam anggaran.



Apa itu SDLC?

1

Definisi

SDLC adalah pendekatan terstruktur untuk pengembangan sistem informasi.

2

Tujuan

Menghasilkan sistem berkualitas tinggi yang memenuhi kebutuhan pelanggan.

3

Proses

Serangkaian fase yang mendefinisikan tugas dalam setiap tahap pengembangan.



Fase-fase SDLC

1

Perencanaan

Mendefinisikan ruang lingkup, tujuan, dan sumber daya proyek.

2

Analisis

Mengumpulkan persyaratan detail dari pemangku kepentingan bisnis dan domain.

3

Desain

Merancang arsitektur sistem dan komponen perangkat lunak.

4

Implementasi

Mengembangkan kode berdasarkan spesifikasi desain.



Fase-fase SDLC (lanjutan)

1

Pengujian

Memverifikasi bahwa sistem memenuhi persyaratan dan bebas dari cacat.

2

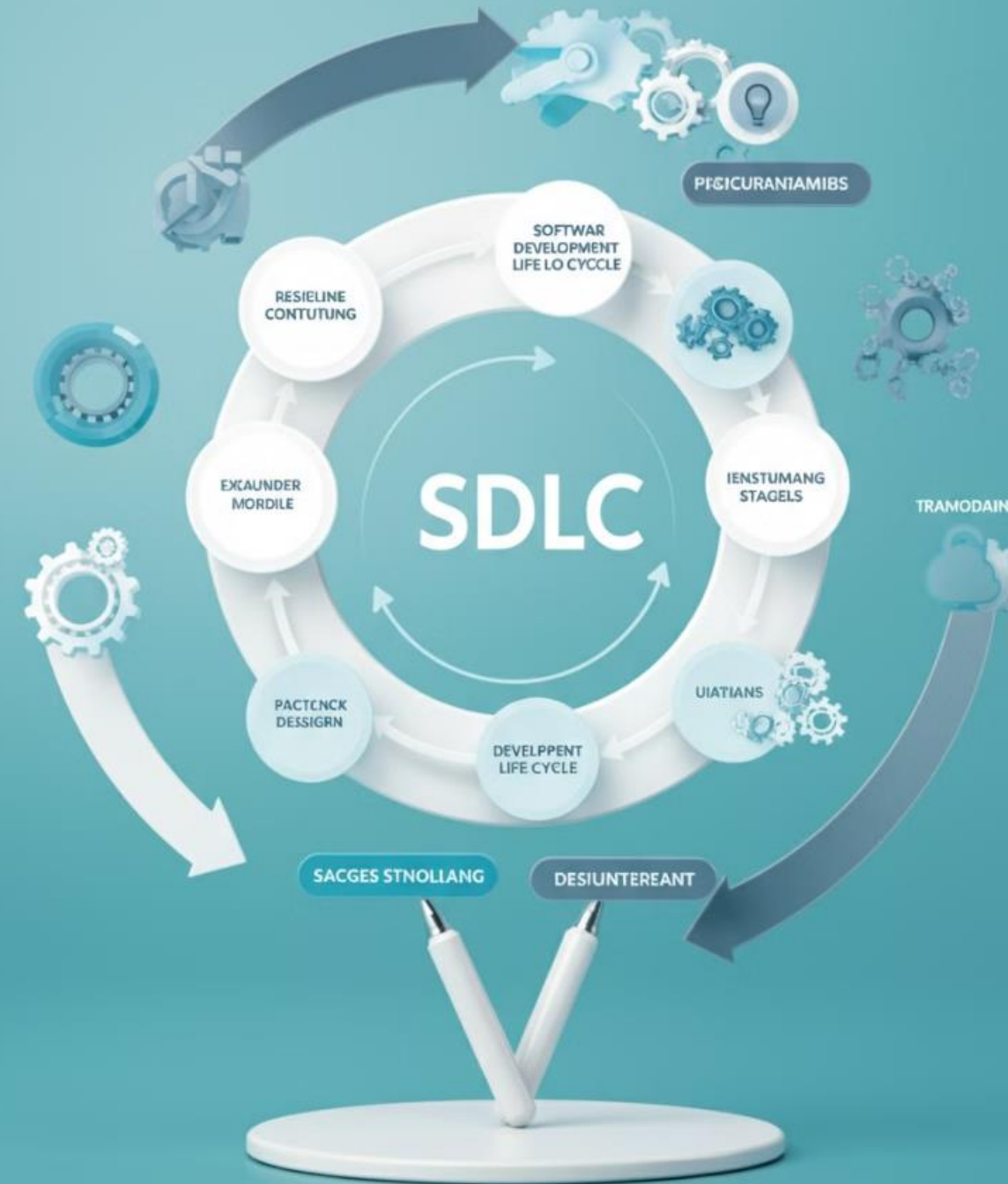
Penerapan

Merilis sistem ke lingkungan produksi.

3

Pemeliharaan

Mendukung sistem yang berjalan, memperbaiki bug, dan menambahkan fitur baru.



Fase Perencanaan



Definisi Proyek

Menentukan tujuan, ruang lingkup, dan batasan proyek.

Studi Kelayakan

Menilai kelayakan teknis, operasional, dan ekonomi proyek.

Penjadwalan

Membuat jadwal proyek dan mengalokasikan sumber daya.

Manajemen Risiko

Mengidentifikasi dan merencanakan mitigasi risiko potensial.

Pentingnya Fase Perencanaan

Dasar yang Kuat

Perencanaan yang baik menciptakan fondasi untuk keberhasilan proyek. Ini memastikan semua pemangku kepentingan memiliki pemahaman yang sama tentang tujuan dan ekspektasi.

Efisiensi Sumber Daya

Dengan perencanaan yang tepat, sumber daya dapat dialokasikan secara efisien. Ini membantu menghindari pemborosan waktu dan biaya selama pengembangan.

Mitigasi Risiko

Mengidentifikasi risiko potensial sejak awal memungkinkan tim untuk mengembangkan strategi mitigasi. Hal ini dapat mencegah masalah besar di kemudian hari.



Fase Analisis

- 1** — Pengumpulan Persyaratan
Melakukan wawancara dengan pemangku kepentingan untuk mengumpulkan kebutuhan sistem.
- 2** — Analisis Kebutuhan
Menganalisis dan memvalidasi persyaratan yang dikumpulkan.
- 3** — Dokumentasi
Membuat dokumen spesifikasi persyaratan perangkat lunak (SRS).
- 4** — Persetujuan Stakeholder
Mendapatkan persetujuan pemangku kepentingan untuk persyaratan yang didokumentasikan.



Teknik Pengumpulan Persyaratan



Wawancara

Melakukan wawancara dengan pemangku kepentingan untuk mendapatkan informasi mendalam.



Survei

Menggunakan kuesioner untuk mengumpulkan data dari sejumlah besar pengguna.



Workshop

Mengadakan sesi brainstorming dengan kelompok pemangku kepentingan.



Observasi

Mengamati pengguna dalam lingkungan kerja mereka untuk memahami proses.



Dokumen Spesifikasi Persyaratan Perangkat Lunak (SRS)

1 Tujuan

Mendokumentasikan semua persyaratan sistem secara rinci dan terstruktur.

3 Format

Menggunakan bahasa yang jelas dan terstruktur untuk menghindari ambiguitas.

2 Komponen Utama

Mencakup persyaratan fungsional, non-fungsional, dan batasan sistem.

4 Manfaat

Menjadi acuan bagi tim pengembang dan dasar untuk pengujian sistem.



Fase Desain

1

Desain Arsitektur

Merancang struktur tingkat tinggi sistem, termasuk komponen utama dan interaksinya.

2

Desain Database

Merancang struktur basis data, termasuk tabel, relasi, dan indeks.

3

Desain Antarmuka

Membuat mockup dan prototipe antarmuka pengguna.

4

Desain Algoritma

Merancang algoritma kunci untuk fungsionalitas sistem yang kompleks.

Pola Desain Perangkat Lunak

Model-View-Controller (MVC)

Memisahkan logika aplikasi dari antarmuka pengguna.

Meningkatkan modularitas dan kemudahan pemeliharaan.

Singleton

Memastikan sebuah kelas hanya memiliki satu instance. Berguna untuk koordinasi aksi di seluruh sistem.

Factory

Menyediakan antarmuka untuk menciptakan objek dalam superclass. Memungkinkan subclass menentukan kelas objek yang dibuat.

LOREM IPSUM DOLON
DOLOR SIT EASIDEIT CIVIST.



LOREM IPSUM DLLON
DOLOR SIT AMEEIT CIMET.

Prinsip SOLID dalam Desain

1 Single Responsibility

Setiap kelas harus memiliki satu, dan hanya satu, alasan untuk berubah.

2 Open-Closed

Entitas perangkat lunak harus terbuka untuk ekstensi, tertutup untuk modifikasi.

3 Liskov Substitution

Objek dalam program harus dapat diganti dengan instance dari subtype mereka.

4 Interface Segregation

Banyak antarmuka klien-spesifik lebih baik daripada satu antarmuka umum.



Fase Implementasi

1

Pengkodean

Menulis kode program berdasarkan desain yang telah dibuat.

2

Code Review

Melakukan tinjauan kode untuk memastikan kualitas dan konsistensi.

3

Unit Testing

Membuat dan menjalankan tes unit untuk setiap komponen.

4

Integrasi

Menggabungkan komponen individual menjadi sistem yang lengkap.



Praktik Pengkodean Terbaik

Penamaan yang Jelas

Gunakan nama variabel dan fungsi yang deskriptif dan mudah dipahami.

Komentar yang Informatif

Tambahkan komentar untuk menjelaskan logika kompleks atau keputusan penting.

DRY (Don't Repeat Yourself)

Hindari duplikasi kode dengan membuat fungsi yang dapat digunakan kembali.

KISS (Keep It Simple, Stupid)

Jaga agar kode tetap sederhana dan mudah dipahami.

Manajemen Versi Kode

Git

Sistem kontrol versi terdistribusi yang populer. Memungkinkan kolaborasi efektif dan pelacakan perubahan kode.

Branching

Membuat cabang terpisah untuk fitur baru atau perbaikan bug. Memungkinkan pengembangan paralel tanpa mengganggu kode utama.

Merging

Menggabungkan perubahan dari satu cabang ke cabang lain. Penting untuk integrasi fitur baru ke dalam kode utama.

Fase Pengujian

1

Unit Testing

Menguji komponen individual untuk memastikan mereka berfungsi seperti yang diharapkan.

2

Integration Testing

Menguji interaksi antara komponen yang telah diintegrasikan.

3

System Testing

Menguji sistem secara keseluruhan terhadap persyaratan yang ditentukan.

4

Acceptance Testing

Memverifikasi bahwa sistem memenuhi kriteria penerimaan pengguna.



Jenis-jenis Pengujian



Jenis Pengujian	Tujuan	Contoh
Fungsional	Memverifikasi fungsionalitas sistem	Pengujian fitur login
Kinerja	Menguji kecepatan dan skalabilitas	Tes beban server
Keamanan	Mengidentifikasi kerentanan	Pengujian penetrasi
Kegunaan	Menilai kemudahan penggunaan	Pengujian antarmuka pengguna



Pengujian Otomatis

- 1 Keuntungan**
Meningkatkan efisiensi, konsistensi, dan cakupan pengujian.
- 2 Alat**
Selenium untuk pengujian web, JUnit untuk Java, pytest untuk Python.
- 3 Continuous Integration**
Integrasi dengan pipeline CI/CD untuk pengujian otomatis saat kode diperbarui.
- 4 Tantangan**
Memerlukan investasi awal dalam penulisan skrip dan pemeliharaan.

Fase Penerapan

1

Persiapan

Menyiapkan lingkungan produksi dan membuat rencana penerapan.

2

Migrasi Data

Mentransfer data dari sistem lama ke sistem baru jika diperlukan.

3

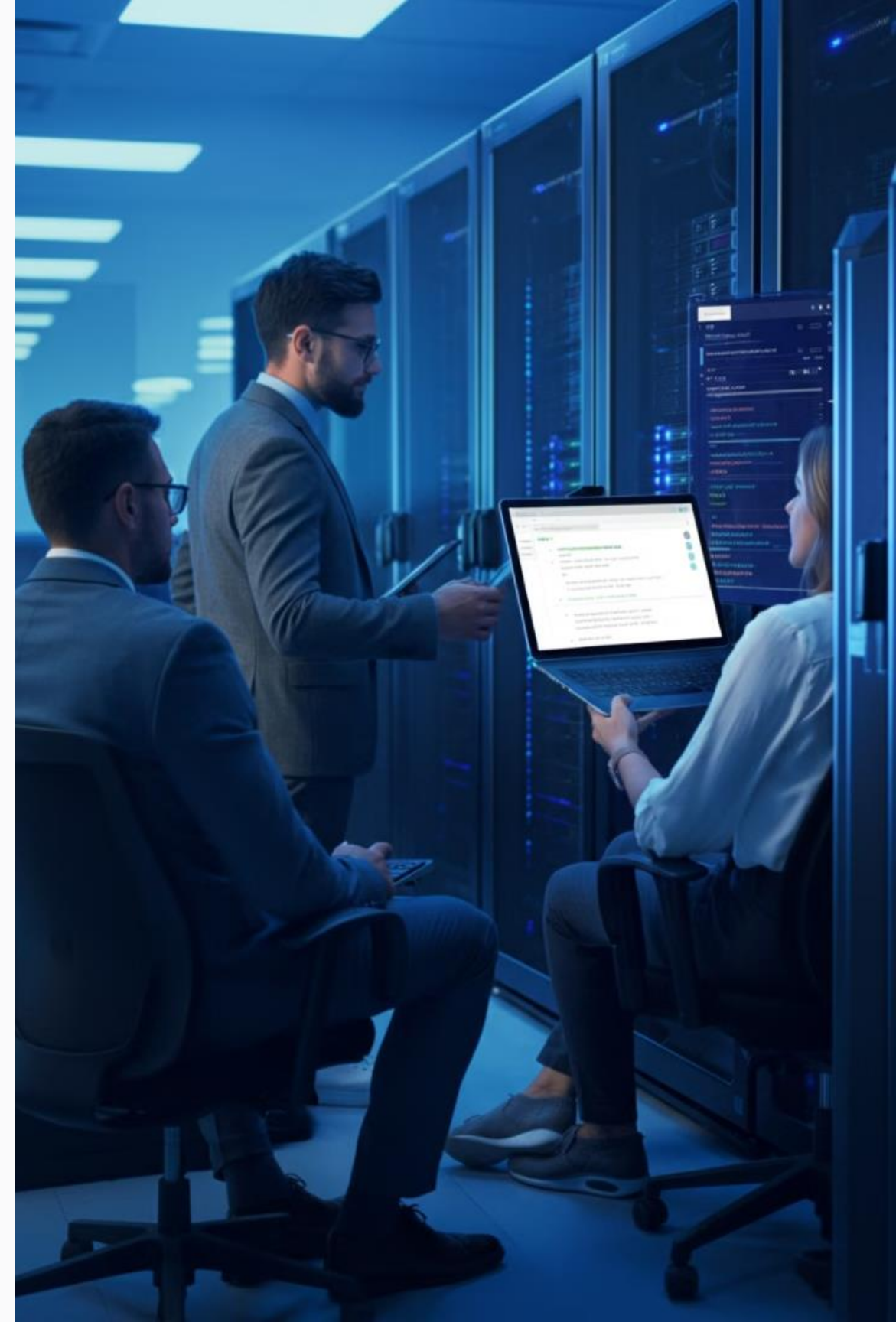
Instalasi

Menerapkan sistem baru ke lingkungan produksi.

4

Verifikasi

Memastikan sistem berjalan dengan benar di lingkungan produksi.



Strategi Penerapan

Big Bang

Mengganti sistem lama dengan sistem baru sekaligus. Berisiko tinggi tetapi cepat.

Bertahap

Menerapkan sistem baru secara bertahap. Mengurangi risiko tetapi memakan waktu lebih lama.

Paralel

Menjalankan sistem lama dan baru secara bersamaan. Aman tetapi memerlukan lebih banyak sumber daya.



Continuous Deployment

1

Integrasi Berkelanjutan

Mengintegrasikan kode ke repositori bersama secara teratur.

2

Pengujian Otomatis

Menjalankan suite pengujian otomatis untuk setiap perubahan.

3

Penerapan Otomatis

Secara otomatis menerapkan kode yang lolos pengujian ke produksi.

4

Pemantauan

Memantau kinerja dan kesehatan sistem secara real-time.



Fase Pemeliharaan

Perbaikan Bug

Mengidentifikasi dan memperbaiki kesalahan yang ditemukan setelah penerapan.

Peningkatan

Menambahkan fitur baru atau meningkatkan fungsionalitas yang ada.

Optimisasi

Meningkatkan kinerja dan efisiensi sistem.

Dukungan

Memberikan bantuan teknis kepada pengguna.

Manajemen Perubahan



1

Permintaan Perubahan

Pengguna atau pemangku kepentingan mengajukan permintaan perubahan.

2

Evaluasi

Tim teknis mengevaluasi dampak dan kelayakan perubahan.

3

Persetujuan

Komite perubahan memutuskan apakah akan menyetujui perubahan.

4

Implementasi

Perubahan yang disetujui diterapkan dan diuji.

Model SDLC

Waterfall

Model linear sekuensial. Setiap fase harus selesai sebelum yang berikutnya dimulai.

Agile

Pendekatan iteratif dan inkremental. Fokus pada fleksibilitas dan kolaborasi dengan pelanggan.

V-Model

Menekankan verifikasi dan validasi. Setiap fase pengembangan memiliki fase pengujian yang terkait.

Model Agile

- 1 Iteratif**
Pengembangan dilakukan dalam siklus pendek yang disebut sprint.
- 2 Adaptif**
Fleksibel terhadap perubahan persyaratan dan prioritas.
- 3 Kolaboratif**
Melibatkan pelanggan secara aktif dalam proses pengembangan.
- 4 Inkremental**
Menghasilkan perangkat lunak yang berfungsi di setiap iterasi.





Scrum dalam Agile



Sprint

Periode kerja tetap, biasanya 2-4 minggu, untuk menyelesaikan serangkaian tugas.



Product Backlog

Daftar prioritas fitur dan persyaratan yang harus dikembangkan.



Daily Standup

Pertemuan singkat harian untuk membahas kemajuan dan hambatan.



Sprint Review

Demonstrasi hasil sprint kepada pemangku kepentingan untuk mendapatkan umpan balik.



DevOps dalam SDLC

1

Integrasi

Menggabungkan kode dari berbagai pengembang secara teratur.

2

Pengujian Otomatis

Menjalankan suite pengujian otomatis untuk setiap perubahan kode.

3

Penerapan

Otomatisasi proses penerapan ke lingkungan produksi.

4

Pemantauan

Memantau kinerja aplikasi dan infrastruktur secara real-time.

Software Security

INTEGATED INTERRATION



Keamanan dalam SDLC

1

Perencanaan

Memasukkan persyaratan keamanan dalam definisi proyek.

2

Desain

Melakukan pemodelan ancaman dan merancang arsitektur yang aman.

3

Implementasi

Menggunakan praktik pengkodean aman dan alat analisis kode statis.

4

Pengujian

Melakukan pengujian keamanan, termasuk pengujian penetrasi.

Metrik SDLC



Metrik	Deskripsi	Manfaat
Cycle Time	Waktu dari ide hingga implementasi	Mengukur efisiensi proses
Defect Density	Jumlah cacat per unit kode	Mengukur kualitas perangkat lunak
Customer Satisfaction	Tingkat kepuasan pengguna	Mengukur keberhasilan proyek
Sprint Velocity	Jumlah pekerjaan yang diselesaikan per sprint	Membantu dalam perencanaan



Masa Depan SDLC

AI dan Machine Learning

Integrasi AI untuk otomatisasi tugas dan prediksi masalah.

Low-Code/No-Code

Platform yang memungkinkan pengembangan aplikasi dengan sedikit atau tanpa coding.

Cloud-Native Development

Pengembangan aplikasi yang dirancang khusus untuk lingkungan cloud.

DevSecOps

Integrasi keamanan lebih dalam ke dalam proses DevOps.