SMART SOFTWARE ENGINEERING

PENDEKATAN MODERN DALAM PENGEMBANGAN SISTEM



Melda Dahoklory, S.Kom.,M.T.
Albertus Lalaun, S.Sos.,M.Si

SMART SOFTWARE ENGINEERING

Pendekatan Modern Dalam Pengembangan Sistem

Melda Dahoklory, S.Kom.,M.T.
Albertus Lalaun, S.Sos.,M.Si

Penerbit widina

SMART SOFTWARE ENGINEERING:
PENDEKATAN MODERN DALAM PENGEMBANGAN SISTEM

Tim Penulis:
Melda Dahoklory, S.Kom., M.T.
Albertus Lalaun, S.Sos., M.Si.

Desain Cover:
Andi Zuliandi

Sumber Ilustrasi:
www.freepik.com

Tata Letak:
Handarini Rohana

Editor:
Bila Nurfadillah

ISBN:
978-634-246-971-2

Cetakan Pertama:
September, 2026

Hak Cipta Dilindungi Oleh Undang-Undang
by Penerbit Widina Media Utama

Dilarang keras menerjemahkan, memfotokopi, atau memperbanyak sebagian atau seluruh isi buku ini tanpa izin tertulis dari Penerbit.

PENERBIT:
WIDINA MEDIA UTAMA
Komplek Puri Melia Asri Blok C3 No. 17 Desa Bojong Emas
Kec. Solokan Jeruk Kabupaten Bandung, Provinsi Jawa Barat

Anggota IKAPI No. 519/JBA/2025
Website: www.penerbitwidina.com
Instagram: @penerbitwidina
Telepon (022) 87355370

KATA PENGANTAR

Puji syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya, buku ajar yang berjudul "Smart Software Engineering: Pendekatan Modern Dalam Pengembangan Sistem" ini dapat diselesaikan dengan baik. Buku ini disusun sebagai respons atas dinamika industri teknologi yang bergerak sangat cepat, di mana rekayasa perangkat lunak tidak lagi sekadar tentang menulis kode, melainkan sebuah disiplin ilmu yang memadukan seni, ilmu pengetahuan, dan manajemen untuk menciptakan solusi digital yang andal, skalabel, dan aman.

Di era transformasi digital ini, perangkat lunak telah menjadi tulang punggung bagi hampir setiap aspek kehidupan dan industri. Dari aplikasi sederhana di ponsel kita hingga sistem *enterprise* kompleks yang menggerakkan ekonomi global, tuntutan akan perangkat lunak berkualitas tinggi semakin meningkat. Hal ini menuntut para insinyur perangkat lunak masa depan untuk tidak hanya menguasai aspek teknis, tetapi juga memahami proses, metodologi, dan prinsip-prinsip yang mendasari pengembangan sistem yang sukses.

Buku ini dirancang untuk menjadi jembatan antara fondasi teoretis rekayasa perangkat lunak dan praktik modern yang diterapkan di industri saat ini. Kami menyajikannya secara sistematis, dimulai dari konsep-konsep dasar, bergerak menuju metodologi Agile dan DevOps, menyelami setiap fase dalam siklus hidup pengembangan—mulai dari analisis, perancangan, implementasi, pengujian, hingga pemeliharaan—dan diakhiri dengan pandangan ke masa depan yang dibentuk oleh AI, *Low-code*, dan IoT.

Salah satu keunikan buku ini adalah penekanannya pada studi kasus nyata. Kami percaya bahwa pemahaman sejati lahir dari melihat bagaimana teori diterapkan untuk memecahkan masalah konkret. Melalui studi kasus dengan kompleksitas yang bervariasi, kami berharap pembaca dapat memperoleh intuisi rekayasa yang tajam dan melihat bagaimana berbagai konsep saling terkait dalam sebuah proyek nyata.

Buku ini ditujukan bagi para mahasiswa di bidang Ilmu Komputer, Teknik Informatika, Sistem Informasi, dan bidang terkait lainnya. Selain itu, para praktisi pemula atau siapa pun yang ingin memperbarui pengetahuannya tentang praktik rekayasa perangkat lunak modern juga akan menemukan buku ini bermanfaat.

Kami menyadari bahwa buku ini masih jauh dari kesempurnaan. Oleh karena itu, kritik dan saran yang membangun dari para pembaca sangat kami harapkan untuk perbaikan di edisi-edisi mendatang. Akhir kata, semoga buku ini dapat memberikan kontribusi positif bagi pengembangan sumber daya manusia di bidang teknologi informasi di Indonesia dan menjadi teman belajar yang berharga bagi para pembacanya.

Selamat belajar dan berkarya!
Penulis

DAFTAR ISI

KATA PENGANTAR iii
DAFTAR ISI v
BAB 1 PENDAHULUAN 1
A. Latar Belakang Perkembangan Software 2
B. Peran Software dalam Transformasi Digital 6
C. Tantangan Pengembangan Sistem Modern 11
D. Tujuan dan Ruang Lingkup Buku 16
BAB 2 KONSEP DASAR REKAYASA PERANGKAT LUNAK 27
A. Definisi RPL 28
B. Karakteristik Perangkat Lunak 33
C. Prinsip-prinsip RPL 37
D. Kualitas Perangkat Lunak (ISO/IEC) 41
BAB 3 *SOFTWARE DEVELOPMENT LIFE CYCLE* (SDLC) 53
A. Konsep SDLC 54
B. Tahapan SDLC 59
C. Perbandingan Model Pengembangan 63
BAB 4 METODOLOGI MODERN PENGEMBANGAN 71
A. Agile Methodology 72
B. Scrum Framework 76
C. Kanban 80
D. DevOps (CI/CD) 84
E. Perbandingan Agile vs. Waterfall 88
BAB 5 ANALISIS KEBUTUHAN SISTEM 97
A. Teknik Pengumpulan Kebutuhan 98
B. Kebutuhan Fungsional & Non-fungsional 102
C. *Use Case* 106
D. *Software Requirement Specification* (SRS) 109
BAB 6 PERANCANGAN SISTEM 119
A. Arsitektur Perangkat Lunak 120
B. UML (*Unified Modeling Language*) 124
C. ERD (*Entity-Relationship Diagram*) 128
D. UI/UX Design 131
BAB 7 IMPLEMENTASI PERANGKAT LUNAK 139
A. *Coding Best Practices* 140
B. Struktur Program 144

C. *Framework* & Teknologi Modern 147
D. *Version Control* (Git & GitHub) 151
BAB 8 PENGUJIAN PERANGKAT LUNAK 161
A. Konsep *Testing* 162
B. *Black Box* & *White Box* 166
C. *Unit Testing* 169
D. *Integration Testing* 172
E. *Automation Testing* 175
BAB 9 *DEPLOYMENT* DAN *DEVOPS* 183
A. Konsep *Deployment* 184
B. *Continuous Integration* & *Continuous Deployment* (CI/CD) 187
C. *Cloud Computing* (Dasar) 191
D. *Container* (Docker Dasar) 194
BAB 10 KEAMANAN PERANGKAT LUNAK 203
A. Konsep Dasar Keamanan 204
B. *Vulnerability* Umum 208
C. *Secure Coding* 211
D. *Authentication* & *Authorization* 213
BAB 11 MANAJEMEN PROYEK PERANGKAT LUNAK 223
A. Perencanaan Proyek 224
B. Estimasi Biaya & Waktu 226
C. Manajemen Tim 229
D. Manajemen Risiko 231
BAB 12 *MAINTENANCE* DAN EVOLUSI SISTEM 239
A. Jenis *Maintenance* 240
B. *Refactoring* 242
C. *Scalability* 246
D. Monitoring Sistem 248
BAB 13 STUDI KASUS PROYEK NYATA 257
A. Studi Kasus: Sistem Informasi Penjualan Toko (Point of Sale - POS) 258
B. Studi Kasus: Sistem Informasi Akademik Kampus 262
C. Studi Kasus: Sistem Informasi *E-Commerce* Berbasis Web 265
D. Studi Kasus*: E-Supply Chain* Management pada Perusahaan Distribusi 269
BAB 14 TREN MASA DEPAN *SOFTWARE ENGINEERING* 275
A. AI dalam *Software Development* 276
B. *Low-code / No-code* 279
C. *Microservices* 282
D. *Internet of Things* (IoT) 284

GLOSARIUM **289**
DAFTAR PUSTAKA **297**
INDEKS **301**
PROFIL PENULIS **306**

BAB 1

PENDAHULUAN

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan evolusi historis perangkat lunak dari era awal hingga komputasi modern.
2. Menganalisis peran strategis perangkat lunak sebagai pendorong utama transformasi digital di berbagai sektor industri.
3. Mengidentifikasi dan menguraikan tantangan-tantangan utama yang dihadapi dalam pengembangan sistem perangkat lunak modern.
4. Memahami tujuan, sasaran, dan ruang lingkup pembahasan yang akan diuraikan dalam buku "Smart Software Engineering: Pendekatan Modern dalam Pengembangan Sistem".

Pendahuluan

Perkembangan peradaban manusia modern tidak dapat dipisahkan dari evolusi teknologi perangkat lunak. Sejak kemunculannya, perangkat lunak telah bertransformasi dari sekadar alat bantu komputasi menjadi tulang punggung yang menopang hampir seluruh aspek kehidupan dan industri. Era digital saat ini ditandai oleh interkonektivitas global, automasi proses bisnis, dan pengambilan keputusan berbasis data, yang semuanya dimungkinkan oleh sistem perangkat lunak yang semakin kompleks dan cerdas. Kehadirannya telah mengubah cara kita berkomunikasi, bekerja, belajar, dan bahkan berinteraksi dengan dunia di sekitar kita (Sommerville, 2016). Oleh karena itu, memahami esensi dari rekayasa perangkat lunak menjadi sebuah keniscayaan bagi para profesional teknologi di masa kini dan masa depan.

Pentingnya perangkat lunak dalam mendorong inovasi dan efisiensi menempatkannya sebagai aset strategis bagi organisasi di seluruh dunia. Transformasi digital, sebuah proses integrasi teknologi digital ke dalam semua area bisnis, secara fundamental mengubah cara operasi dan

BAB 2

KONSEP DASAR REKAYASA PERANGKAT LUNAK

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan definisi formal Rekayasa Perangkat Lunak (RPL) dan membedakannya dari pemrograman biasa.
2. Mengidentifikasi dan menganalisis karakteristik unik dari perangkat lunak yang membedakannya dari produk rekayasa lainnya.
3. Menguraikan prinsip-prinsip fundamental yang menjadi panduan dalam praktik rekayasa perangkat lunak yang baik.
4. Memahami kerangka kerja kualitas perangkat lunak berdasarkan standar ISO/IEC 25010 dan menjelaskan berbagai atribut kualitasnya.

Pendahuluan

Setelah memahami konteks historis dan peran strategis perangkat lunak pada bab sebelumnya, langkah selanjutnya adalah membangun fondasi teoretis yang kokoh mengenai disiplin ilmu yang bertanggung jawab atas penciptaannya: Rekayasa Perangkat Lunak (RPL). Istilah ini sering kali digunakan secara bergantian dengan "pemrograman" atau "coding", namun pada kenyataannya, RPL memiliki cakupan yang jauh lebih luas dan mendalam. RPL bukanlah sekadar aktivitas menulis kode, melainkan sebuah pendekatan sistematis, disiplin, dan terukur untuk pengembangan, operasi, dan pemeliharaan perangkat lunak. Disiplin ini menerapkan prinsip-prinsip rekayasa untuk mengatasi kompleksitas dan memastikan bahwa produk perangkat lunak yang dihasilkan berkualitas tinggi, andal, dan memenuhi kebutuhan pengguna secara efektif (Pressman & Maxim, 2020).

Pentingnya memahami konsep dasar RPL terletak pada kemampuannya untuk mengubah proses pengembangan perangkat lunak dari aktivitas yang bersifat *ad-hoc* dan tidak terprediksi menjadi sebuah proses rekayasa yang

BAB 3

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep dasar dan pentingnya *Software Development Life Cycle* (SDLC) sebagai kerangka kerja proses dalam rekayasa perangkat lunak.
2. Mengidentifikasi dan menguraikan tahapan-tahapan generik yang ada dalam sebuah SDLC.
3. Menganalisis dan membandingkan berbagai model proses SDLC, termasuk model prediktif (Waterfall) dan model iteratif (Prototype, Spiral).
4. Mengevaluasi kelebihan dan kekurangan dari setiap model SDLC untuk menentukan kesesuaiannya dengan berbagai jenis proyek.

Pendahuluan

Membangun perangkat lunak berkualitas tinggi, terutama untuk sistem yang kompleks, bukanlah proses yang dapat dilakukan secara acak. Seperti yang telah dibahas pada bab sebelumnya, disiplin rekayasa perangkat lunak bertujuan untuk membawa keteraturan dan prediktabilitas ke dalam proses pengembangan. Salah satu alat fundamental untuk mencapai tujuan ini adalah dengan mengadopsi sebuah *Software Development Life Cycle* (SDLC) atau Siklus Hidup Pengembangan Perangkat Lunak. SDLC adalah sebuah kerangka kerja terstruktur yang mendefinisikan serangkaian tahapan atau fase yang harus dilalui oleh sebuah proyek perangkat lunak, mulai dari konsepsi awal hingga masa pensiunnya. Kerangka kerja ini menyediakan peta jalan yang jelas bagi tim pengembang untuk diikuti, memastikan bahwa aktivitas-aktivitas penting dilakukan dalam urutan yang logis (Pressman & Maxim, 2020).

BAB 4

METODOLOGI MODERN PENGEMBANGAN

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan filosofi dan nilai-nilai inti dari Agile Manifesto sebagai landasan metodologi pengembangan modern.
2. Menganalisis Scrum Framework, termasuk peran, artefak, dan acara (events) yang ada di dalamnya.
3. Memahami prinsip-prinsip visualisasi alur kerja dan pembatasan *Work in Progress* (WIP) dalam metode Kanban.
4. Menjelaskan konsep DevOps sebagai budaya dan praktik yang mengintegrasikan pengembangan dan operasi, termasuk CI/CD.
5. Membandingkan secara kritis antara pendekatan Agile dan model Waterfall tradisional.

Pendahuluan

Pada bab sebelumnya, kita telah membahas model-model proses SDLC tradisional seperti Waterfall dan Spiral. Meskipun model-model tersebut memberikan struktur, kelemahan utamanya, terutama pada model prediktif, adalah ketidakmampuannya untuk beradaptasi secara efektif terhadap perubahan. Di dunia bisnis modern yang bergerak sangat cepat, di mana tuntutan pasar dan teknologi dapat berubah dalam hitungan bulan, pendekatan yang kaku dan berbasis rencana jangka panjang sering kali gagal. Kebutuhan akan kecepatan, fleksibilitas, dan responsivitas yang lebih tinggi mendorong lahirnya gelombang baru metodologi pengembangan perangkat lunak yang secara kolektif dikenal sebagai "Agile" (Pressman & Maxim, 2020).

Agile bukanlah sebuah metodologi tunggal, melainkan sebuah payung filosofi atau pola pikir (*mindset*) yang menghargai individu dan interaksi, perangkat lunak yang berfungsi, kolaborasi dengan pelanggan, dan respons terhadap perubahan di atas proses dan alat yang komprehensif,

BAB 5

ANALISIS KEBUTUHAN SISTEM

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan berbagai teknik pengumpulan (elisitasi) kebutuhan dan memilih teknik yang sesuai untuk konteks proyek yang berbeda.
2. Membedakan secara jelas antara kebutuhan fungsional dan non-fungsional serta mengklasifikasikan contoh-contohnya.
3. Memahami konsep *Use Case* dan mampu membuat deskripsi *Use Case* untuk memodelkan interaksi pengguna dengan sistem.
4. Menjelaskan tujuan, struktur, dan karakteristik dari dokumen *Software Requirement Specification* (SRS) sesuai standar industri.

Pendahuluan

Setelah memahami berbagai model dan metodologi pengembangan pada bab-bab sebelumnya, kita sekarang memasuki salah satu aktivitas teknis paling fundamental dan kritis dalam rekayasa perangkat lunak: Analisis Kebutuhan Sistem. Tahap ini, yang juga dikenal sebagai rekayasa kebutuhan (*requirements engineering*), adalah proses untuk menemukan, menganalisis, mendokumentasikan, dan memvalidasi kebutuhan atau layanan yang harus dipenuhi oleh sebuah sistem. Keberhasilan sebuah proyek perangkat lunak sangat bergantung pada seberapa baik tahap ini dilakukan. Sejumlah besar proyek gagal bukan karena kesalahan teknis dalam pengkodean, melainkan karena produk akhir tidak sesuai dengan apa yang sebenarnya dibutuhkan oleh pengguna dan pemangku kepentingan (Pressman & Maxim, 2020).

Pentingnya analisis kebutuhan yang cermat tidak dapat dilebih-lebihkan. Kesalahan yang dibuat pada tahap ini memiliki efek domino dan biayanya akan berlipat ganda jika baru ditemukan pada tahap-tahap selanjutnya. Sebuah kebutuhan yang salah dipahami dapat menyebabkan desain yang keliru, yang kemudian diimplementasikan dalam ratusan atau ribuan baris

BAB 6

PERANCANGAN SISTEM

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan pentingnya arsitektur perangkat lunak dan mengidentifikasi berbagai gaya arsitektur umum.
2. Memahami tujuan dan penggunaan *Unified Modeling Language* (UML) sebagai bahasa pemodelan standar.
3. Membuat dan menginterpretasikan diagram UML fundamental, termasuk *Use Case Diagram*, *Activity Diagram*, dan *Class Diagram*.
4. Merancang model data logis menggunakan *Entity-Relationship Diagram* (ERD).
5. Menjelaskan prinsip-prinsip dasar dari desain antarmuka pengguna dan pengalaman pengguna (UI/UX).

Pendahuluan

Setelah berhasil mendefinisikan "apa" yang harus dilakukan oleh sistem melalui analisis kebutuhan pada bab sebelumnya, kita kini beralih ke tahap perancangan sistem, yang berfokus pada "bagaimana" sistem akan dibangun untuk memenuhi kebutuhan tersebut. Tahap perancangan adalah jembatan krusial antara pemahaman masalah (analisis) dan solusi teknis (implementasi). Keputusan-keputusan yang dibuat pada tahap ini akan membentuk struktur, perilaku, dan kualitas sistem secara keseluruhan. Perancangan yang baik adalah fondasi untuk perangkat lunak yang kokoh, dapat dipelihara, dan skalabel, sementara perancangan yang buruk dapat menyebabkan sistem yang rapuh, sulit diubah, dan mahal untuk dipelihara, tidak peduli seberapa baik kodenya ditulis (Pressman & Maxim, 2020).

Perancangan sistem bukanlah aktivitas tunggal, melainkan serangkaian kegiatan yang mencakup berbagai tingkat abstraksi. Dimulai dari tingkat tertinggi, yaitu perancangan arsitektur, yang mendefinisikan struktur makro dari sistem-komponen-komponen utamanya dan bagaimana mereka saling

BAB 7

IMPLEMENTASI PERANGKAT LUNAK

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan dan menerapkan praktik terbaik dalam penulisan kode (*coding best practices*) untuk menghasilkan kode yang bersih dan mudah dipelihara.
2. Memahami pentingnya struktur program yang baik dan prinsip-prinsip desain yang mendukungnya.
3. Mengidentifikasi peran dan manfaat dari *framework* perangkat lunak modern dalam mempercepat pengembangan.
4. Menguasai konsep dasar sistem kontrol versi dan mampu menggunakan Git dan GitHub untuk kolaborasi tim.

Pendahuluan

Tahap implementasi, atau pengkodean, adalah proses di mana desain perangkat lunak yang telah dibuat pada bab sebelumnya diterjemahkan menjadi kode program yang dapat dieksekusi oleh komputer. Bagi banyak orang di luar dunia rekayasa perangkat lunak, tahap inilah yang sering dianggap sebagai "pekerjaan" yang sebenarnya. Namun, dalam konteks rekayasa perangkat lunak yang matang, implementasi adalah kelanjutan logis dari analisis dan desain yang cermat. Kualitas dari tahap ini tidak hanya diukur dari apakah program berjalan sesuai fungsi, tetapi juga dari seberapa baik kode tersebut ditulis-apakah bersih, dapat dipahami, dan mudah untuk dimodifikasi di masa depan (Pressman & Maxim, 2020).

Menulis kode yang "berfungsi" adalah satu hal, tetapi menulis kode yang berkualitas tinggi adalah hal lain. Kode yang ditulis dengan buruk, meskipun fungsional, dapat menjadi mimpi buruk untuk dipelihara dan dikembangkan lebih lanjut. Kode semacam ini sering disebut sebagai *technical debt* (utang teknis), di mana jalan pintas yang diambil hari ini akan "dibayar" dengan bunga berupa waktu dan biaya yang lebih besar di masa

BAB 8

PENGUJIAN PERANGKAT LUNAK

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan tujuan fundamental dari pengujian perangkat lunak dan perannya dalam siklus hidup pengembangan.
2. Membedakan antara strategi pengujian *Black Box* dan *White Box* serta teknik-teknik yang terkait.
3. Memahami tujuan dan karakteristik dari *Unit Testing* sebagai fondasi dari piramida pengujian.
4. Menjelaskan pentingnya *Integration Testing* untuk memverifikasi interaksi antar komponen.
5. Mengidentifikasi manfaat dan peran *Automation Testing* dalam praktik pengembangan modern seperti CI/CD.

Pendahuluan

Setelah perangkat lunak dirancang dan diimplementasikan, bagaimana kita bisa yakin bahwa perangkat lunak tersebut bekerja sesuai dengan yang diharapkan? Bagaimana kita menemukan cacat (*defect* atau *bug*) sebelum perangkat lunak tersebut sampai ke tangan pengguna? Jawabannya terletak pada aktivitas krusial yang disebut pengujian perangkat lunak. Pengujian adalah proses mengevaluasi dan memverifikasi bahwa sebuah produk atau aplikasi perangkat lunak melakukan apa yang seharusnya dilakukannya. Ini adalah proses investigasi yang dilakukan untuk memberikan informasi kepada pemangku kepentingan tentang kualitas produk (Pressman & Maxim, 2020). Pengujian bukan lagi sekadar fase akhir yang dilakukan sebelum rilis, melainkan aktivitas integral yang terjadi di seluruh siklus hidup pengembangan.

Tujuan pengujian sering kali disalahpahami. Banyak yang berpikir bahwa tujuan pengujian adalah untuk membuktikan bahwa perangkat lunak tidak memiliki *bug*. Namun, seperti yang dinyatakan oleh Edsger

BAB 9

DEPLOYMENT DAN *DEVOPS*

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep *deployment* dan membandingkan berbagai strategi *deployment* modern.
2. Memahami alur kerja dan manfaat dari *Continuous Integration* dan *Continuous Deployment* (CI/CD).
3. Menjelaskan definisi, karakteristik esensial, dan model layanan dari *Cloud Computing* menurut NIST.
4. Memahami konsep dasar kontainerisasi dan mampu menjelaskan peran Docker dalam proses pengembangan dan *deployment*.

Pendahuluan

Setelah melalui serangkaian tahap mulai dari analisis, desain, implementasi, hingga pengujian, langkah terakhir dalam siklus pengiriman perangkat lunak adalah membuat aplikasi tersebut tersedia untuk digunakan oleh pengguna akhir. Proses ini dikenal sebagai *deployment* (penyebaran). Secara historis, *deployment* adalah aktivitas yang menegangkan, berisiko tinggi, dan jarang dilakukan, sering kali dijadwalkan pada akhir pekan atau tengah malam untuk meminimalkan gangguan. Proses manual yang rawan kesalahan ini sering kali menjadi sumber utama dari waktu henti (*downtime*) dan masalah produksi (Humble & Farley, 2010).

Namun, lanskap pengembangan perangkat lunak modern telah berubah secara dramatis. Munculnya budaya DevOps-sebuah perpaduan filosofi, praktik, dan alat-telah meruntuhkan tembok antara tim Pengembangan (Dev) dan Operasi (Ops), dengan tujuan untuk mengotomatiskan dan menyederhanakan proses pengiriman perangkat lunak. DevOps mempromosikan kolaborasi dan tanggung jawab bersama atas seluruh siklus hidup aplikasi, dari ide hingga produksi. Inti dari implementasi teknis DevOps adalah praktik *Continuous Integration* (CI) dan *Continuous*

BAB 10

KEAMANAN PERANGKAT LUNAK

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan konsep dasar keamanan perangkat lunak, termasuk pilar CIA Triad dan pentingnya pendekatan *Shift Left*.
2. Mengidentifikasi dan menjelaskan beberapa kerentanan perangkat lunak paling umum yang tercantum dalam OWASP Top 10.
3. Memahami dan menerapkan prinsip-prinsip dasar pengkodean yang aman (*secure coding*) dalam proses pengembangan.
4. Membedakan antara otentikasi dan otorisasi serta menjelaskan mekanisme implementasi dasar untuk keduanya.

Pendahuluan

Dalam dunia yang semakin terhubung secara digital, perangkat lunak telah menjadi tulang punggung hampir setiap aspek kehidupan modern, mulai dari perbankan, kesehatan, hingga infrastruktur kritis. Seiring dengan meningkatnya peran perangkat lunak, meningkat pula potensi dampak negatif jika perangkat lunak tersebut disalahgunakan atau gagal berfungsi akibat serangan siber. Keamanan perangkat lunak bukan lagi sekadar fitur tambahan atau pemikiran setelah produk jadi; ia telah menjadi atribut kualitas non-fungsional yang paling fundamental. Sebuah perangkat lunak, secanggih apa pun fiturnya, akan menjadi tidak berharga jika tidak dapat dipercaya untuk melindungi data dan menjaga integritas operasinya (McGraw, 2006).

Secara tradisional, keamanan sering kali diperlakukan sebagai aktivitas yang dilakukan pada akhir siklus pengembangan-sebuah "lapisan" yang ditambahkan sebelum rilis. Pendekatan ini terbukti sangat tidak efektif dan mahal. Cacat keamanan yang ditemukan pada tahap akhir pengembangan jauh lebih sulit dan mahal untuk diperbaiki daripada cacat fungsional. Filosofi modern, yang sering disebut DevSecOps atau *Shift Left Security*,

BAB 11

MANAJEMEN PROYEK PERANGKAT LUNAK

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan proses dan artefak kunci dalam perencanaan proyek perangkat lunak.
2. Membandingkan berbagai teknik estimasi biaya dan waktu, baik tradisional maupun Agile.
3. Memahami dinamika dan praktik terbaik dalam manajemen tim pengembangan perangkat lunak.
4. Mengidentifikasi, menganalisis, dan merencanakan respons terhadap risiko dalam proyek perangkat lunak.

Pendahuluan

Keberhasilan sebuah proyek perangkat lunak tidak hanya ditentukan oleh keunggulan teknis dari kode yang dihasilkan, tetapi juga oleh seberapa baik proyek tersebut dikelola. Manajemen Proyek Perangkat Lunak adalah disiplin ilmu yang menerapkan pengetahuan, keterampilan, alat, dan teknik pada aktivitas proyek untuk memenuhi persyaratan proyek. Tanpa manajemen yang efektif, bahkan tim pengembang yang paling berbakat pun dapat gagal menghasilkan produk yang tepat waktu, sesuai anggaran, dan memenuhi kebutuhan pemangku kepentingan. Proyek bisa terjebak dalam *scope creep* (penambahan lingkup yang tidak terkendali), mengalami pembengkakan biaya, atau melewatkan tenggat waktu yang krusial (Pressman & Maxim, 2020).

Secara historis, manajemen proyek perangkat lunak sangat dipengaruhi oleh disiplin ilmu teknik sipil dan manufaktur, yang mengarah pada pendekatan prediktif atau *plan-driven* seperti model Waterfall. Pendekatan ini mengasumsikan bahwa persyaratan dapat sepenuhnya dipahami di awal dan menekankan pada perencanaan yang mendetail di muka. Namun, sifat perangkat lunak yang tidak berwujud dan sering kali berubahnya

BAB 12

MAINTENANCE DAN EVOLUSI SISTEM

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Membedakan empat jenis utama pemeliharaan perangkat lunak (*maintenance*) dan memahami hukum evolusi perangkat lunak.
2. Menjelaskan tujuan dan proses *refactoring* sebagai cara untuk mengelola utang teknis (*technical debt*).
3. Memahami konsep skalabilitas (*scalability*) dan membedakan antara penskalaan vertikal dan horizontal.
4. Menjelaskan pentingnya pemantauan (*monitoring*) sistem dan komponen-komponen utamanya (metrik, *logging*, *tracing*).

Pendahuluan

Pekerjaan pada sebuah sistem perangkat lunak tidak berakhir saat ia berhasil di-*deploy* ke produksi. Sebaliknya, *deployment* hanyalah awal dari fase terpanjang dan sering kali paling mahal dalam siklus hidup perangkat lunak: fase pemeliharaan dan evolusi. Perangkat lunak yang sukses dan bermanfaat adalah perangkat lunak yang digunakan, dan perangkat lunak yang digunakan pasti akan membutuhkan perubahan. Baik itu untuk memperbaiki *bug* yang baru ditemukan, beradaptasi dengan lingkungan baru (seperti sistem operasi yang diperbarui), atau menambahkan fungsionalitas baru untuk memenuhi kebutuhan pengguna yang terus berubah, perubahan adalah suatu keniscayaan (Pressman & Maxim, 2020).

Fase pasca-*deployment* ini sering kali dipandang sebelah mata, dianggap kurang glamor dibandingkan dengan pengembangan fitur baru dari awal. Namun, sebagian besar total biaya kepemilikan (*total cost of ownership*) sebuah sistem perangkat lunak justru dihabiskan untuk pemeliharaan. Mengabaikan kesehatan jangka panjang sistem demi kecepatan jangka pendek dapat menyebabkan akumulasi utang teknis (*technical debt*), di mana kualitas internal kode menurun dari waktu ke

BAB 13

STUDI KASUS PROYEK NYATA

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menerapkan proses analisis kebutuhan untuk mendefinisikan lingkup proyek nyata.
2. Merancang arsitektur, database (ERD), dan antarmuka (UI/UX) untuk sebuah sistem informasi.
3. Mengaplikasikan konsep dari metodologi Agile (seperti Scrum) dalam perencanaan dan eksekusi proyek.
4. Mengidentifikasi tantangan umum dalam siklus hidup pengembangan perangkat lunak dan merumuskan solusinya berdasarkan konsep yang telah dipelajari.

Pendahuluan

Teori, prinsip, dan metodologi rekayasa perangkat lunak memberikan fondasi yang kuat, tetapi pemahaman sejati datang dari penerapan pengetahuan tersebut dalam praktik. Bab-bab sebelumnya telah membekali kita dengan berbagai alat konseptual, mulai dari analisis kebutuhan, perancangan dengan UML, metodologi Agile, hingga praktik DevOps dan keamanan. Sekarang, saatnya untuk melihat bagaimana semua bagian tersebut menyatu dalam konteks proyek nyata. Bab ini akan menjauh dari pembahasan teoretis dan beralih ke narasi pengembangan beberapa sistem dari awal hingga akhir, menyoroti keputusan, tantangan, dan kompromi yang dibuat di sepanjang jalan.

Tujuan dari studi kasus ini adalah untuk menjembatani kesenjangan antara "mengetahui" dan "melakukan". Dengan mengikuti perjalanan pengembangan sebuah sistem, kita dapat melihat bagaimana konsep abstrak seperti *user story*, diagram kelas, atau *pipeline* CI/CD diimplementasikan dalam skenario yang konkret. Setiap studi kasus akan berfungsi sebagai laboratorium mini di mana kita dapat mengamati

BAB 14

TREN MASA DEPAN
SOFTWARE ENGINEERING

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

1. Menjelaskan peran dan dampak Kecerdasan Buatan (AI), khususnya *Generative AI*, dalam berbagai fase siklus hidup pengembangan perangkat lunak.
2. Memahami konsep, keuntungan, dan batasan dari platform pengembangan *Low-code* dan *No-code*.
3. Menganalisis evolusi dan tantangan masa depan dari arsitektur *microservices*.
4. Mengidentifikasi karakteristik unik dan tantangan rekayasa perangkat lunak dalam pengembangan sistem *Internet of Things* (IoT).

Pendahuluan

Disiplin rekayasa perangkat lunak tidak pernah statis; ia terus-menerus berevolusi sebagai respons terhadap kemajuan teknologi, perubahan ekspektasi pasar, dan pemahaman kita yang semakin dalam tentang cara membangun sistem yang kompleks. Praktik-praktik yang dianggap canggih satu dekade lalu kini menjadi standar, dan alat-alat yang kita gunakan hari ini mungkin akan usang di masa depan. Sebagai calon insinyur perangkat lunak, memahami lintasan evolusi ini dan mengantisipasi tren masa depan adalah kunci untuk tetap relevan dan efektif dalam karier (Pressman & Maxim, 2020).

Pergeseran paradigma terjadi secara berkala dalam industri kita. Dari pemrograman terstruktur ke pemrograman berorientasi objek, dari model Waterfall ke Agile, dari server fisik ke *cloud computing*, setiap gelombang inovasi membawa cara-cara baru dalam berpikir dan bekerja. Saat ini, kita berada di tengah-tengah beberapa pergeseran transformatif yang didorong

GLOSARIUM

A

***Agile*:** Sebuah filosofi pengembangan perangkat lunak yang menekankan pada pengembangan iteratif, kolaborasi tim, umpan balik pelanggan yang berkelanjutan, dan kemampuan untuk merespons perubahan dengan cepat.

API (*Application Programming Interface*): Sekumpulan aturan dan protokol yang memungkinkan berbagai aplikasi perangkat lunak untuk berkomunikasi satu sama lain.

***API Gateway*:** Sebuah titik masuk tunggal yang mengelola, merutekan, dan mengamankan permintaan dari klien ke berbagai layanan *backend* (biasanya dalam arsitektur *microservices*).

Arsitektur Perangkat Lunak (*Software Architecture*): Struktur fundamental dari sebuah sistem perangkat lunak, yang terdiri dari komponen-komponen utamanya, hubungan antar komponen, dan prinsip-prinsip yang memandu desain dan evolusinya.

***Authentication* (Otentikasi):** Proses untuk memverifikasi identitas seorang pengguna atau sistem. Menjawab pertanyaan, "Siapa Anda?".

***Authorization* (Otorisasi):** Proses untuk menentukan hak akses atau izin yang dimiliki oleh pengguna yang telah terotentikasi. Menjawab pertanyaan, "Apa yang boleh Anda lakukan?".

B

***Black Box Testing*:** Teknik pengujian perangkat lunak di mana penguji tidak memiliki pengetahuan tentang struktur internal atau kode dari sistem. Pengujian dilakukan berdasarkan spesifikasi dan kebutuhan.

***Branch* (Git):** Sebuah versi independen dari basis kode. *Branching* memungkinkan pengembang untuk bekerja pada fitur atau perbaikan *bug* secara terisolasi tanpa mengganggu kode utama.

DAFTAR PUSTAKA

Abbott, M. L., & Fisher, M. T. (2015). *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise* (2nd ed.). Addison-Wesley Professional.

Amalia, H., dkk. (2024). *Rekayasa Perangkat Lunak*. PT Serasi Media Teknologi.

Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley Professional.

Bird, C., dkk. (2020). The Evolving Landscape of Software Development with AI. *Proceedings of the 42nd International Conference on Software Engineering*. https://doi.org/10.1145/3377814.3381710

Bishop, M. (2002). *Computer Security: Art and Science*. Addison-Wesley Professional.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice Hall.

Brooks, F. P., Jr. (1995). *The Mythical Man-Month: Essays on Software Engineering* (Anniversary ed.). Addison-Wesley Professional.

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress.

Chen, X., dkk. (2021). *Evaluating Large Language Models Trained on Code*. arXiv preprint arXiv:2107.03374. https://arxiv.org/abs/2107.03374

Cohn, M. (2005). *Agile Estimating and Planning*. Prentice Hall.

Cooper, A., Reimann, R., Cronin, D., & Noessel, C. (2014). *About Face: The Essentials of Interaction Design* (4th ed.). Wiley.

Davis, K., & Daniels, D. (2016). *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*. O'Reilly Media.

DeMarco, T. (1999). *The Deadline: A Novel About Project Management*. Dorset House.

Docker Inc. (2024). *Docker Documentation*. https://docs.docker.com/

Dowd, M., McDonald, J., & Schuh, J. (2006). *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional.

Erl, T., Puttini, R., & Mahmood, Z. (2013). *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code* (2nd ed.). Addison-Wesley Professional.

INDEKS

A

Activity Diagram, 6.2, 13.1
Agile
    Estimasi (Estimation), 11.2
    Manifesto, 4.1
    Metodologi, Bab 4
    Prinsip, 4.1
    vs. Waterfall, 4.5, 11.1
AI (Artificial Intelligence)
    dalam Pengembangan Perangkat Lunak, 14.1
    AIOps, 14.1
    Generative AI, 14.1
Analisis Kebutuhan, Bab 5, 13.1, 13.2
Analisis Risiko, 11.4
API (Application Programming Interface), 7.3, 10.4, 13.2, 13.3
API Gateway, 13.3, 13.4
Arsitektur Perangkat Lunak, 6.1, 12.2, 12.3
    Event-Driven (EDA), 13.4, 14.3
    Microservices, 13.3, 13.4, 14.3
    Monolitik, 6.1, 13.1
    SOA (Service-Oriented), 13.2
    Tiga Lapis (3-tier), 6.1, 13.1
Authentication (Otentikasi), 10.4
Authorization (Otorisasi), 10.4
Automation Testing, 8.5

B

Black Box Testing, 8.2
Brooks, Frederick (Hukum Brooks), 11.2
Budget (Anggaran), 11.2
Bug (Cacat), 2.1, 8.1, 12.1

C

Cache (Caching), 12.3

PROFIL PENULIS



Melda Dahoklory, S.Kom., M.T, lahir di Kepulauan Aru (Dobo) pada tanggal 11 Maret 1988. Penulis menyelesaikan pendidikan Sarjana (S1) pada Program Studi Teknik Informatika di STMIK Akakom Yogyakarta (yang kini bernama Universitas Teknologi Digital Indonesia) pada tahun 2010. Selanjutnya, saya melanjutkan pendidikan Magister (S2) pada Program Studi Teknik Informatika di Universitas Atma Jaya Yogyakarta dan memperoleh gelar Magister Teknik pada tahun 2013. Latar belakang pendidikan tersebut menjadi dasar dalam pengembangan kompetensi saya di bidang Enterprise Information System dan Rekayasa Perangkat Lunak. Penulis Merupakan Akademisi Di Bidang Teknologi Informasi yang memiliki komitmen terhadap pengembangan ilmu pengetahuan serta pelaksanaan Tri Dharma Perguruan Tinggi, yang meliputi pendidikan, penelitian, dan pengabdian kepada Masyarakat. Saat ini penulis menjabat sebagai Koordinator Program Studi Teknik Informatika pada Politeknik Negeri Ambon. Dalam pelaksanaan tugas profesional, penulis aktif dalam kegiatan pendidikan, penelitian, dan pengabdian kepada masyarakat. Penulis juga terlibat dalam berbagai kegiatan pengembangan sistem informasi serta pendampingan pemanfaatan teknologi bagi masyarakat dan institusi. Adapun bidang minat saya mencakup pengembangan aplikasi berbasis kebutuhan pengguna, peningkatan literasi digital, serta inovasi di bidang teknologi informasi. Saat ini saya berdomisili di Jln. Dr. J. Leimena, Kecamatan Hative Besar, Kota Ambon, Provinsi Maluku. Ke depan, saya berkomitmen untuk terus meningkatkan kompetensi akademik dan profesional, serta memberikan kontribusi yang berkelanjutan bagi pengembangan ilmu pengetahuan dan penerapan teknologi informasi di Indonesia.



Albertus Lalaun, S.Sos., M.Si. lahir di Romean (Kabupaten Kepulauan Tanimbar, Provinsi Maluku) pada 21 November 1976. Penulis menempuh pendidikan tinggi pada Program Studi Ilmu Pemerintahan, Fakultas Ilmu Sosial dan Ilmu Politik di Universitas Pattimura Ambon dan berhasil menyelesaikan studi sarjana pada tahun 2001. Selama masa perkuliahan, penulis menaruh minat pada kajian pemerintahan, pembangunan daerah, serta dinamika sosial masyarakat. Keinginan untuk memperdalam pengetahuan mengenai pembangunan mendorong penulis melanjutkan pendidikan pada Program Magister Studi Pembangunan di Universitas Kristen Satya Wacana Salatiga. Pada tahun 2004 penulis memperoleh gelar Magister Sains (M.Si.). Pendidikan tersebut menjadi bekal dengan pemahaman yang lebih komprehensif mengenai kebijakan publik, perencanaan pembangunan, dan pemberdayaan masyarakat. Saat ini penulis menjalani karier, sebagai seorang Dosen dan Peneliti pada Program Studi Akuntansi Sektor Publik Politeknik Negeri Ambon. Penulis juga aktif sebagai anggota dalam Perkumpulan Akademisi Periset Indonesia (PAKAR).

"SMART SOFTWARE ENGINEERING: PENDEKATAN MODERN DALAM PENGEMBANGAN SISTEM" adalah panduan komprehensif yang dirancang untuk membekali mahasiswa dan praktisi dengan pengetahuan fundamental dan keterampilan canggih dalam rekayasa perangkat lunak modern. Buku ini membawa pembaca dalam perjalanan terstruktur, dimulai dari konsep dasar seperti SDLC dan prinsip-prinsip rekayasa, hingga menyelami metodologi modern seperti Agile, Scrum, dan DevOps yang menjadi standar industri saat ini.

Pembaca akan dibimbing melalui setiap fase krusial dalam siklus hidup pengembangan: mulai dari analisis kebutuhan yang cermat, perancangan sistem yang solid menggunakan UML dan arsitektur modern, hingga praktik implementasi, pengujian, dan deployment yang efektif. Aspek-aspek non-fungsional yang vital seperti keamanan perangkat lunak, manajemen proyek, dan skalabilitas sistem dibahas secara mendalam.

Untuk menjembatani teori dan praktik, buku ini menyajikan serangkaian studi kasus nyata yang mengilustrasikan penerapan konsep dalam proyek dengan skala berbeda-dari sistem POS sederhana hingga platform e-commerce berbasis microservices. Sebagai penutup, sebuah bab khusus didedikasikan untuk menjelajahi tren masa depan yang akan membentuk industri, termasuk dampak revolusioner dari AI, Low-code/No-code, dan Internet of Things (IoT). Buku ini adalah peta jalan esensial bagi siapa pun yang ingin membangun perangkat lunak berkualitas tinggi di era digital.

www.penerbitwidina.com
@penerbitwidina
penerbit widina
penerbitwidina@gmail.com
widina store
widina bookstore

Layanan Pembaca & Penerbitan Buku
0815-7000-699

SCANME

Teknologi & Komputer

ISBN 978-634-246-971-2

9 786342 469712