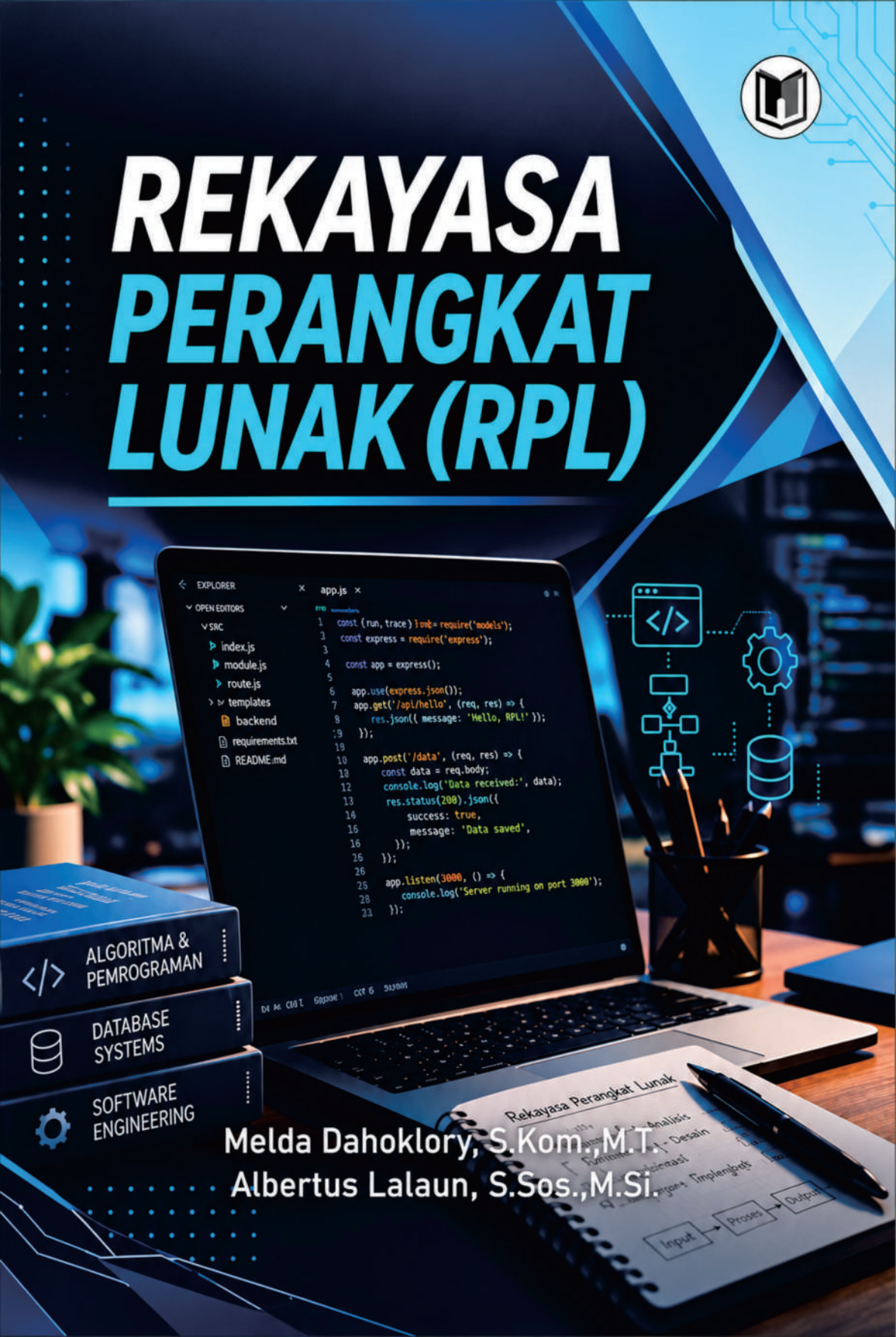
REKAYASA PERANGKAT LUNAK (RPL)



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

REKAYASA PERANGKAT LUNAK (RPL)

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

Penerbit widina

REKAYASA PERANGKAT LUNAK (RPL)

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:
978-634-317-515-5

Cetakan Pertama:
November, 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 ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga buku Rekayasa Perangkat Lunak ini dapat diselesaikan dengan baik. Buku ini disusun sebagai salah satu sumber belajar yang diharapkan mampu memberikan pemahaman yang komprehensif mengenai konsep, prinsip, metode, serta penerapan rekayasa perangkat lunak sesuai dengan perkembangan teknologi informasi dan kebutuhan industri saat ini.

Di tengah era transformasi digital yang pesat, perangkat lunak telah menjadi tulang punggung inovasi di hampir seluruh sektor industri. Kemampuan untuk merancang, mengembangkan, dan memelihara sistem perangkat lunak yang berkualitas bukan lagi sekadar keahlian teknis, melainkan sebuah kompetensi strategis yang fundamental. Namun, penguasaan rekayasa perangkat lunak tidak cukup hanya melalui pemahaman teoretis. Diperlukan sebuah jembatan yang menghubungkan konsep-konsep abstrak dengan tantangan praktis di dunia nyata.

Transformasi digital yang berlangsung di berbagai sektor telah menjadikan perangkat lunak sebagai fondasi utama dalam mendukung proses bisnis, pelayanan publik, pendidikan, kesehatan, hingga berbagai aktivitas masyarakat. Oleh karena itu, penguasaan rekayasa perangkat lunak tidak lagi terbatas pada kemampuan menulis kode program, tetapi juga mencakup kemampuan menganalisis kebutuhan, merancang solusi, mengimplementasikan sistem, melakukan pengujian, hingga memelihara perangkat lunak secara berkelanjutan.

Akhir kata, semoga buku ini dapat memberikan manfaat yang sebesar-besarnya, tidak hanya dalam menunjang keberhasilan akademis, tetapi juga dalam membekali pembelajar dengan fondasi yang kokoh untuk berkarir dan berkontribusi di dunia teknologi yang terus berkembang.

DAFTAR ISI

KATA PENGANTAR ..... **iii**
DAFTAR ISI ..... **iv**
BAB 1 KONSEP DASAR REKAYASA PERANGKAT LUNAK **1**
A. Pentingnya Rekayasa Perangkat Lunak di Era Digital 1
B. Peran dalam Memahami Teori Rekayasa Perangkat Lunak 3
C. Tantangan Pengembangan Software Modern 4
BAB 2 ANALISIS KEBUTUHAN **11**
A. Pengantar Analisis Kebutuhan 11
B. Identifikasi *Stakeholder* dan Aktor 14
C. Teknik Pengumpulan Kebutuhan 18
D. Klasifikasi Kebutuhan 22
E. Pemodelan Kebutuhan 25
F. Penyusunan SRS (*Software Requirement Specification*) 28
G. Validasi dan Verifikasi Kebutuhan 31
H. Studi Kasus 34
BAB 3 DESAIN SISTEM **45**
A. Konsep Dasar Desain Sistem 45
B. Desain Arsitektur Sistem 48
C. Pemodelan UML (*Unified Modeling Language*) 51
D. Class Diagram 53
E. Activity Diagram 56
F. Sequence Diagram 59
G. Desain Basis Data 62
H. Desain Antarmuka (UI/UX) 65
I. Integrasi Desain Sistem 69
BAB 4 IMPLEMENTASI **77**
A. Konsep Dasar Implementasi 77
B. Persiapan Lingkungan Pengembangan 80
C. Pengenalan Bahasa Pemrograman 83
D. Implementasi Database 85
E. Implementasi Backend 88
F. Implementasi Frontend 91
G. Integrasi Sistem 94
H. *Version Control* (Git) 96
I. *Debugging dan Error Handling* 99
J. *Tools* Pendukung 102

BAB 5 PENGUJIAN 111
A. Konsep Dasar Pengujian Perangkat Lunak 111
B. Prinsip-Prinsip Pengujian 115
C. Jenis-Jenis Pengujian 118
D. Metode Pengujian 122
E. Teknik Pengujian Black Box 127
F. Penyusunan Test Case 130
G. Eksekusi Pengujian 133
H. Dokumentasi Hasil Pengujian 136
I. Tools Pengujian 139
BAB 6 *DEPLOYMENT* 145
A. Konsep Dasar *Deployment* 145
B. Jenis *Deployment* 148
C. Proses *Deployment* Aplikasi 150
D. Pengujian Setelah *Deployment* 153
E. Manajemen Versi dan Update 156
F. Keamanan Dasar dalam *Deployment* 158
G. Monitoring dan *Maintenance* 160
H. Dokumentasi *Deployment* 163
BAB 7 PROYEK AKHIR 171
A. Pilihan Studi Kasus Proyek Akhir 171
B. Tahapan Pengerjaan Proyek 174
C. Kriteria Penilaian Proyek 176
D. Jadwal Pengerjaan Proyek (Saran) 177
E. Tips Sukses Pengerjaan Proyek 178
F. Struktur Detail Laporan Akhir 179
BAB 8 PENGUJIAN PERANGKAT LUNAK 181
A. Konsep Dasar Pengujian 181
B. Tingkatan Pengujian (*Testing Levels*) 183
C. Strategi Pengujian 184
D. Desain Kasus Uji (*Test Case*) 185
E. Dokumentasi dan Pelaporan Bug 186
GLOSARIUM 193
DAFTAR PUSTAKA 201
PROFIL PENULIS 210

BAB 1

KONSEP DASAR REKAYASA PERANGKAT LUNAK

A. PENTINGNYA REKAYASA PERANGKAT LUNAK DI ERA DIGITAL

Di tengah gelombang transformasi digital yang melanda seluruh sektor industri, dari keuangan hingga kesehatan, rekayasa perangkat lunak telah berevolusi dari sekadar fungsi pendukung teknis menjadi tulang punggung strategis bagi keberlangsungan dan inovasi organisasi. Perangkat lunak kini menjadi medium utama di mana nilai bisnis diciptakan, layanan diberikan kepada pelanggan, dan efisiensi operasional dicapai. Kemampuan untuk mengembangkan sistem perangkat lunak yang andal, aman, dan dapat diskalakan secara cepat merupakan faktor pembeda kompetitif yang signifikan. Tanpa penerapan prinsip-prinsip rekayasa yang disiplin, perusahaan berisiko menghasilkan produk yang tidak stabil, sulit dipelihara, dan gagal memenuhi ekspektasi pasar yang dinamis.

Oleh karena itu, memahami urgensi RPL di era ini adalah langkah pertama bagi setiap calon praktisi teknologi. Disiplin ini menyediakan kerangka kerja yang memungkinkan tim pengembang untuk mengelola kompleksitas, memitigasi risiko, dan memastikan bahwa produk akhir selaras dengan tujuan bisnis. Para ahli seperti Pressman dan Maxim (2020) menekankan bahwa rekayasa perangkat lunak adalah tentang menciptakan fondasi berkualitas untuk program komputer dengan menerapkan proses, metode, dan alat yang terbukti. Dengan demikian, RPL bukan hanya tentang menulis kode, melainkan sebuah pendekatan holistik yang mencakup manajemen proyek, analisis kebutuhan, desain arsitektur, hingga jaminan kualitas untuk menghasilkan solusi digital yang berdampak.

Peran rekayasa perangkat lunak sebagai pilar utama inovasi digital terlihat jelas pada bagaimana perusahaan modern beroperasi dan bersaing. Dalam ekonomi digital, model bisnis baru sering kali didasarkan sepenuhnya pada platform perangkat lunak, seperti yang ditunjukkan oleh kesuksesan

BAB 2

ANALISIS KEBUTUHAN

A. PENGANTAR ANALISIS KEBUTUHAN

Analisis kebutuhan, atau *requirements engineering*, adalah proses sistematis untuk menemukan, menganalisis, mendokumentasikan, dan memvalidasi kebutuhan atau layanan yang harus dipenuhi oleh sebuah sistem perangkat lunak. Ini adalah jembatan komunikasi esensial yang menghubungkan dunia pengguna dan pemangku kepentingan bisnis dengan dunia teknis para pengembang. Tanpa jembatan yang kokoh ini, tim pengembang berisiko membangun produk yang secara teknis mungkin sempurna, namun sama sekali tidak berguna karena tidak menyelesaikan masalah yang tepat atau tidak memenuhi ekspektasi pengguna. Oleh karena itu, tahap ini menjadi penentu arah dan lingkup keseluruhan proyek.

Tujuan utama dari analisis kebutuhan melampaui sekadar membuat daftar fitur. Proses ini bertujuan untuk mencapai pemahaman bersama (*shared understanding*) di antara semua pihak yang terlibat mengenai masalah yang akan dipecahkan dan batasan-batasan solusi yang akan dibangun. Para ahli seperti Wiegers dan Beatty (2013) menekankan bahwa investasi waktu dan tenaga pada fase awal ini akan memberikan pengembalian yang signifikan dengan mengurangi pengerjaan ulang (*rework*) yang mahal di kemudian hari. Dengan demikian, analisis kebutuhan berfungsi sebagai mekanisme mitigasi risiko yang proaktif, memastikan bahwa sumber daya pengembangan difokuskan untuk membangun produk yang benar (*build the right product*).

1. Definisi Analisis Kebutuhan

Analisis kebutuhan secara formal dapat didefinisikan sebagai sebuah proses interdisipliner yang mencakup semua aktivitas yang terlibat dalam penemuan, elaborasi, negosiasi, spesifikasi, validasi, dan manajemen kebutuhan untuk sebuah sistem perangkat lunak. Ini bukan sekadar aktivitas tunggal, melainkan sebuah sub-disiplin dalam rekayasa perangkat lunak yang

BAB 3

DESAIN SISTEM

A. KONSEP DASAR DESAIN SISTEM

Desain sistem adalah fase dalam siklus hidup pengembangan perangkat lunak di mana spesifikasi kebutuhan yang telah dianalisis ditransformasikan menjadi sebuah rencana detail untuk implementasi. Ini adalah proses kreatif namun sistematis untuk mendefinisikan arsitektur, komponen, modul, antarmuka, dan data untuk sebuah sistem agar dapat memenuhi kebutuhan yang telah ditetapkan. Fase ini menjawab pertanyaan "bagaimana" sistem akan bekerja secara internal untuk menyediakan fungsionalitas yang diinginkan secara eksternal. Sebuah desain yang baik adalah kunci untuk membangun sistem yang tidak hanya berfungsi, tetapi juga andal, dapat dipelihara, dan mampu beradaptasi dengan perubahan di masa depan.

Tujuan utama dari desain sistem adalah untuk menghasilkan model atau representasi sistem yang dapat digunakan oleh tim pengembang sebagai panduan untuk implementasi. Model ini harus cukup detail untuk menghindari ambiguitas selama proses pengkodean. Menurut para ahli seperti Pressman dan Maxim (2020), desain yang baik harus menunjukkan hierarki komponen perangkat lunak, struktur data yang jelas, dan representasi antarmuka yang presisi. Dengan kata lain, desain menciptakan peta jalan yang terstruktur, mengurangi kemungkinan pengembang tersesat atau membangun komponen yang tidak terintegrasi dengan baik satu sama lain.

1. Pengertian Desain Sistem

Desain sistem dapat didefinisikan sebagai proses penerapan berbagai teknik dan prinsip dengan tujuan untuk mendefinisikan sebuah perangkat, proses, atau sistem secara cukup detail untuk memungkinkan realisasi fisiknya. Dalam konteks rekayasa perangkat lunak, ini berarti proses mendefinisikan elemen-elemen perangkat lunak seperti arsitektur, modul, dan komponen; antarmuka yang berbeda dari elemen-elemen tersebut; dan

BAB 4

IMPLEMENTASI

A. KONSEP DASAR IMPLEMENTASI

Implementasi adalah fase dalam *Software Development Life Cycle* (SDLC) di mana tim pengembang secara aktif menulis kode untuk membangun komponen-komponen perangkat lunak sesuai dengan spesifikasi desain yang telah disetujui. Ini adalah proses konstruksi di mana cetak biru arsitektur dan desain detail diubah menjadi produk yang dapat dieksekusi. Fase ini sering kali merupakan fase yang paling memakan waktu dalam banyak proyek dan memerlukan kombinasi keterampilan teknis, pemecahan masalah, dan disiplin untuk menghasilkan kode berkualitas tinggi.

Tujuan utama dari tahap implementasi bukan hanya untuk membuat perangkat lunak yang memenuhi kebutuhan fungsional, tetapi juga untuk melakukannya dengan cara yang selaras dengan keputusan desain dan atribut kualitas yang telah ditetapkan. Ini berarti kode yang ditulis harus efisien, aman, dan dapat diandalkan. Menurut McConnell (2004) dalam karyanya yang komprehensif tentang konstruksi perangkat lunak, kualitas kode yang dihasilkan selama implementasi memiliki dampak terbesar pada kemudahan pengujian, *debugging*, dan pemeliharaan sistem di masa depan.

1. Pengertian Implementasi Sistem

Implementasi sistem, dalam konteks rekayasa perangkat lunak, adalah proses realisasi fisik dari desain sistem. Ini mencakup serangkaian aktivitas teknis, dengan yang utama adalah penulisan kode program dalam bahasa pemrograman yang telah dipilih. Namun, pengertian implementasi melampaui sekadar pengkodean. Ini juga mencakup aktivitas penting lainnya seperti pembuatan skema basis data, konfigurasi lingkungan server, instalasi *library* atau *framework* pihak ketiga, dan integrasi awal antar komponen yang telah dibangun. Pada dasarnya, ini adalah semua langkah yang diperlukan untuk mengubah spesifikasi desain menjadi sistem yang berjalan, meskipun mungkin belum sepenuhnya diuji.

BAB 5

PENGUJIAN

A. KONSEP DASAR PENGUJIAN PERANGKAT LUNAK

Memasuki dunia pengujian perangkat lunak memerlukan pemahaman yang kuat tentang terminologi dan tujuan dasarnya. Pengujian bukanlah sekadar menjalankan program dan melihat apakah ada yang rusak. Ini adalah proses intelektual yang menuntut pemikiran kritis, perhatian terhadap detail, dan sedikit "pesimisme yang konstruktif". Menurut Myers, Sandler, dan Badgett (2011), pengujian adalah proses mengeksekusi sebuah program dengan tujuan menemukan kesalahan. Definisi ini secara fundamental mengubah pola pikir: tujuan pengujian bukanlah untuk membuktikan bahwa program bekerja, tetapi untuk menemukan skenario di mana program tersebut gagal.

Memahami konsep dasar ini sangat penting karena ia membentuk dasar untuk semua aktivitas pengujian lainnya. Ini membantu membedakan antara verifikasi dan validasi, dua aspek kualitas yang sering kali tertukar. Verifikasi bertanya, "Apakah kita membangun produk dengan benar?" (sesuai spesifikasi), sementara validasi bertanya, "Apakah kita membangun produk yang benar?" (sesuai kebutuhan pengguna). Pengujian perangkat lunak mencakup kedua aktivitas ini, memastikan bahwa perangkat lunak tidak hanya berfungsi secara teknis tetapi juga memberikan nilai bisnis yang diharapkan (Ammann & Offutt, 2016).

1. Pengertian Pengujian (*Software Testing*)

Pengujian perangkat lunak (*software testing*) adalah sebuah proses empiris dan teknis yang dirancang untuk mengevaluasi satu atau lebih properti dari sebuah perangkat lunak dengan cara mengamatinya saat dieksekusi. Menurut standar ISO/IEC/IEEE 29119-1, pengujian didefinisikan sebagai serangkaian aktivitas yang dilakukan untuk menentukan bahwa sebuah sistem perangkat lunak memenuhi kebutuhan yang telah ditentukan dan untuk mendeteksi cacat. Proses ini melibatkan perancangan kasus uji

BAB 6

DEPLOYMENT

A. KONSEP DASAR *DEPLOYMENT*

Deployment perangkat lunak adalah kulminasi dari semua upaya yang dilakukan selama siklus hidup pengembangan. Ini adalah proses yang kompleks untuk memindahkan perangkat lunak dari lingkungan pengembangan ke lingkungan di mana pengguna dapat mengaksesnya. Memahami konsep dasar di balik *deployment* sangat penting untuk memastikan bahwa proses ini berjalan lancar dan tidak menjadi sumber masalah baru. *Deployment* yang berhasil tidak hanya berarti aplikasi berjalan, tetapi juga berjalan dengan andal, aman, dan berperforma baik.

Dalam pengembangan modern, *deployment* tidak lagi dilihat sebagai peristiwa besar yang menakutkan dan jarang terjadi. Sebaliknya, tujuannya adalah untuk membuat *deployment* menjadi aktivitas rutin, dapat diprediksi, dan berisiko rendah, yang dapat dilakukan sesuai permintaan. Filosofi ini, yang merupakan inti dari DevOps, memungkinkan organisasi untuk merespons perubahan pasar dengan lebih cepat dan memberikan nilai kepada pelanggan secara lebih sering (Kim et al., 2016).

1. Pengertian *Deployment*

Deployment perangkat lunak, dalam definisi yang paling mendasar, adalah proses yang mencakup semua aktivitas yang diperlukan untuk membuat sistem perangkat lunak tersedia untuk digunakan. Ini melibatkan instalasi, konfigurasi, dan pengaktifan aplikasi di lingkungan target. Proses ini dapat bervariasi secara signifikan tergantung pada jenis aplikasi dan arsitekturnya. Untuk aplikasi *desktop* sederhana, *deployment* mungkin hanya berarti membuat sebuah *installer*. Namun, untuk aplikasi web yang kompleks, *deployment* melibatkan serangkaian langkah yang rumit, termasuk mengunggah file kode, mengkonfigurasi *web server*, menyiapkan basis data, dan mengatur variabel lingkungan (Ebert et al., 2016).

BAB 7

PROYEK AKHIR

A. PILIHAN STUDI KASUS PROYEK AKHIR

Berikut adalah beberapa pilihan studi kasus yang dapat dikembangkan menjadi Proyek Akhir . Setiap tim/individu harus memilih satu studi kasus dan mengembangkannya secara penuh. Pilihlah studi kasus yang paling menarik dan sesuai dengan pemahaman tim Anda.

1. Studi Kasus: Sistem Informasi Penjualan UMKM
    - Latar Belakang: Banyak Usaha Mikro, Kecil, dan Menengah (UMKM) seperti kedai kopi, toko kelontong, atau butik kecil masih mengelola penjualan secara manual menggunakan buku catatan. Hal ini menyulitkan mereka dalam melacak stok, menghitung pendapatan harian, dan menganalisis produk terlaris.
    - Tujuan Sistem: Mengembangkan sistem *Point of Sale* (POS) sederhana berbasis web atau desktop untuk membantu UMKM mengelola transaksi penjualan, mencatat produk, mengelola stok barang, dan menghasilkan laporan penjualan sederhana.
    - Fitur Utama:
        1) Manajemen Produk (CRUD: Tambah, Lihat, Ubah, Hapus Produk beserta harga dan stok awal).
        2) Proses Transaksi Penjualan (memilih produk, memasukkan jumlah, sistem menghitung total).
        3) Pencatatan Riwayat Transaksi.
        4) Manajemen Stok (stok otomatis berkurang saat ada penjualan).
        5) Laporan Penjualan Harian/Bulanan.

BAB 8

PENGUJIAN PERANGKAT LUNAK

A. KONSEP DASAR PENGUJIAN

Pengujian perangkat lunak adalah sebuah proses, atau serangkaian proses, yang dirancang untuk memastikan bahwa kode komputer melakukan apa yang seharusnya dilakukan dan tidak melakukan apa yang tidak seharusnya dilakukan. Ini adalah bagian integral dari jaminan kualitas perangkat lunak (*Software Quality Assurance* - SQA) yang bertujuan untuk menyediakan informasi objektif dan independen tentang kualitas perangkat lunak kepada para pemangku kepentingan. Informasi ini membantu mereka dalam membuat keputusan bisnis, misalnya apakah perangkat lunak sudah siap untuk dirilis atau tidak.

Pengujian yang efektif bukanlah aktivitas yang serampangan atau acak. Ia memerlukan perencanaan, desain, dan eksekusi yang cermat. Proses ini dimulai dari fase analisis kebutuhan, di mana kebutuhan yang dapat diuji didefinisikan, dan berlanjut hingga setelah perangkat lunak di-*deploy* dalam bentuk pengujian pemeliharaan.

1. Tujuan dan Prinsip Pengujian

Tujuan utama pengujian adalah menemukan cacat (defects/bugs). Setiap *bug* yang ditemukan dan diperbaiki sebelum produk sampai ke tangan pengguna adalah sebuah kemenangan bagi kualitas. Tujuan penting lainnya adalah memverifikasi bahwa perangkat lunak memenuhi kebutuhan yang telah ditetapkan dalam dokumen SRS. Pengujian berfungsi sebagai mekanisme untuk memeriksa apakah sistem yang dibangun adalah sistem yang benar (*build the right system*). Selain itu, pengujian juga bertujuan untuk membangun kepercayaan diri terhadap kualitas produk. Dengan serangkaian pengujian yang komprehensif dan berhasil, tim dapat memiliki tingkat kepercayaan yang lebih tinggi untuk merilis produk mereka.

GLOSARIUM

A

***Agile*:** Sekelompok metodologi pengembangan perangkat lunak yang didasarkan pada pengembangan iteratif dan inkremental, di mana kebutuhan dan solusi berkembang melalui kolaborasi antara tim yang terorganisir dan pelanggan/pengguna akhir.

Aktor (*Actor*): Peran yang dimainkan oleh pengguna atau sistem eksternal lain saat berinteraksi dengan sistem. Aktor digambarkan sebagai *stick figure* dalam Diagram Use Case.

Analisis Kebutuhan (*Requirements Engineering*): Proses sistematis untuk menemukan, menganalisis, mendokumentasikan, dan memvalidasi kebutuhan untuk sebuah sistem perangkat lunak.

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

Arsitektur Perangkat Lunak (*Software Architecture*): Struktur tingkat tinggi dari sebuah sistem perangkat lunak, yang terdiri dari komponen-komponen utama, hubungan di antaranya, dan prinsip-prinsip yang mengatur desain dan evolusinya.

B

***Backend*:** Bagian dari aplikasi perangkat lunak yang berjalan di sisi server dan tidak terlihat langsung oleh pengguna. Bertanggung jawab atas logika bisnis, interaksi basis data, dan otentikasi.

***Black-box Testing*:** Strategi pengujian di mana penguji tidak memiliki pengetahuan tentang struktur internal atau kode sistem. Pengujian didasarkan murni pada spesifikasi dan kebutuhan (input dan output).

DAFTAR PUSTAKA

Alexander, I., & Robertson, S. (2012). *Mastering the requirements process: Getting requirements right* (3rd ed.). Addison-Wesley Professional.

Ammann, P., & Offutt, J. (2016). *Introduction to software testing* (2nd ed.). Cambridge University Press.

Bass, L., Clements, P., & Kazman, R. (2022). *Software architecture in practice* (4th ed.). Addison-Wesley Professional.

Beck, K. (2003). *Test-driven development: By example*. Addison-Wesley Professional.

Beizer, B. (1990). *Software testing techniques* (2nd ed.). Van Nostrand Reinhold.

Beizer, B. (1995). *Black-box testing: Techniques for functional testing of software and systems*. John Wiley & Sons.

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site reliability engineering: How Google runs production systems*. O'Reilly Media.

Beyer, H., & Holtzblatt, K. (1998). *Contextual design: Defining customer-centered systems*. Morgan Kaufmann.

Black, R. (2009). *Managing the testing process: Practical tools and techniques for managing hardware and software testing* (3rd ed.). John Wiley & Sons.

Boehm, B., & Basili, V. R. (2001). Software defect reduction top 10 list. *Computer, 34*(1), 135–137. https://doi.org/10.1109/2.962984

Bourne, L. (2015). *Making projects work: Effective stakeholder and communication management*. CRC Press.

Brooks, F. P. (1995). *The mythical man-month: Essays on software engineering*. Addison-Wesley Professional.

Card, D. N. (1990). *Measuring software design quality*. Prentice Hall.

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress. https://git-scm.com/book/en/v2

Cockburn, A. (2001). *Writing effective use cases*. Addison-Wesley Professional.

Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Cohn, M. (2009). *Succeeding with agile: Software development using scrum*. Addison-Wesley Professional.

Colantonio, J. (2021). *The API testing handbook*. Independently published.

Cooper, A., Reimann, R., & Cronin, D. (2014). *About face: The essentials of interaction design* (4th ed.). Wiley.

Crispin, L., & Gregory, J. (2009). *Agile testing: A practical guide for testers and agile teams*. Addison-Wesley Professional.

Davis, K., & Daniels, D. (2016). *Effective DevOps: Building a culture of collaboration, affinity, and tooling at scale*. O'Reilly Media.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software, 33*(3), 94–100. https://doi.org/10.1109/MS.2016.68

Erl, T., Puttini, R., & Mahmood, Z. (2013). *Cloud computing: Concepts, technology & architecture*. Pearson.

Fewster, M., & Graham, D. (1999). *Software test automation*. Addison-Wesley.

Fowler, M. (2003). *UML distilled: A brief guide to the standard object modeling language* (3rd ed.). Addison-Wesley Professional.

Fowler, M. (2019). *Blue-green deployment*. https://martinfowler.com/bliki/BlueGreenDeployment.html

Fowler, M. (2019). *Microservices resource guide*. https://martinfowler.com/microservices/

Freeman, E., & Robson, E. (2014). *Head first design patterns*. O'Reilly Media.

Garousi, V., & Mäntylä, M. V. (2016). A systematic literature review of software release engineering. *Information and Software Technology, 73*, 1–24. https://doi.org/10.1016/j.infsof.2016.01.006

Ghezzi, A., & Cavallo, A. (2020). Agile business model innovation in digital entrepreneurship. *International Journal of Innovation and Technology Management, 17*(06), 2050040. https://doi.org/10.1142/S021987702050040X

Gilb, T., & Graham, D. (1993). *Software inspection*. Addison-Wesley.

Goucher, A., & Rice, R. (2017). *Beautiful testing*. O'Reilly Media.

Hull, E., Jackson, K., & Dick, J. (2017). *Requirements engineering* (4th ed.). Springer.

Humble, J., & Farley, D. (2010). *Continuous delivery*. Addison-Wesley Professional.

IEEE. (2008). *IEEE standard for software and system test documentation (IEEE Std 829-2008)*.

Jorgensen, P. C. (2013). *Software testing: A craftsman's approach* (4th ed.). CRC Press.

Kaner, C. (2017). *Foundations of software testing*. Context-Driven Press.

Kaner, C., Falk, J., & Nguyen, H. Q. (2013). *Testing computer software*. John Wiley & Sons.

Kim, G., Debois, P., Willis, J., & Humble, J. (2016). *The DevOps handbook*. IT Revolution Press.

Kolb, D. A. (1984). *Experiential learning*. Prentice-Hall.

Langdon, M., & Souter, A. (2011). *A practitioner's guide to software test design*. Artech House.

Laplante, P. A. (2017). *Requirements engineering for software and systems* (3rd ed.). CRC Press.

Laudon, K. C., & Laudon, J. P. (2020). *Management information systems* (16th ed.). Pearson.

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2019). DevOps implementation. *Journal of Systems and Software, 157*, 104611. https://doi.org/10.1016/j.jss.2019.07.069

Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, 2*(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

McConnell, S. (2004). *Code complete* (2nd ed.). Microsoft Press.

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing (SP 800-145)*. https://doi.org/10.6028/NIST.SP.800-145

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.

Narayanan, A., & Shmatikov, V. (2005). Robust re-authentication. *IEEE Symposium on Security and Privacy*, 301–315. https://doi.org/10.1109/SP.2005.27

Newman, S. (2021). *Building microservices* (2nd ed.). O'Reilly Media.

Norman, D. (2013). *The design of everyday things*. Basic Books.

Osherove, R. (2009). *The art of unit testing*. Manning Publications.

OWASP Foundation. (2021). *OWASP Top Ten*. https://owasp.org/www-project-top-ten/

Ozkaya, I. (2020). What is technical debt. *IEEE Software, 37*(3), 10–13. https://doi.org/10.1109/MS.2020.2981978

Patton, R. (2019). *Software testing*. Sams Publishing.

Pohl, K. (2016). *Requirements engineering*. Springer.

Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

Puppet. (2021). *State of DevOps report*. https://puppet.com/resources/report/2021-state-of-devops-report

Richards, M. (2021). *Fundamentals of software architecture*. O'Reilly Media.

Richards, M., & Ford, N. (2020). *Fundamentals of software architecture*. O'Reilly Media.

Sharma, N., & Trewin, S. (2017). *Cloud system administration*. Addison-Wesley Professional.

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.

Stack Overflow. (2021). *Developer survey*. https://insights.stackoverflow.com/survey/2021

Valacich, J. S., & George, J. F. (2020). *Modern systems analysis and design* (9th ed.). Pearson.

Veracode. (2021). *State of software security report*. https://www.veracode.com/state-of-software-security-report

Whittaker, J. A. (2009). *Exploratory software testing*. Addison-Wesley.

Wiegers, K., & Beatty, J. (2013). *Software requirements* (3rd ed.). Microsoft Press.

Zhu, H., Hall, P. A. V., & May, J. H. R. (1997). Software unit test coverage. *ACM Computing Surveys, 29*(4), 366–427. https://doi.org/10.1145/267580.267592

INDEKS

A

Agile, 1.1, 1.3, 2.5.3, 3.1.3
Aktor (Actor), 2.2, 2.2.3, 2.5.1, 3.1.3
Analisis Kebutuhan, 1.2, Bab 2 (keseluruhan)
Analis Sistem, 2.1.3
API (Application Programming Interface), 1.3, 3.2.2, 4.5.3, 4.6.2
Arsitektur Perangkat Lunak, 1.1, 3.1.1, 3.2 (keseluruhan)
    Client-Server, 3.2.2
    Microservices, 1.3, 3.2.2, 3.2.3
    Monolithic, 1.3, 3.2.2, 3.2.3
Atribut (Kelas), 3.4.1
Atribut Kualitas, 1.1, 3.1.1, 3.2.1

B

Backend, 4.5 (keseluruhan), 4.6.2, 4.7.1
Basis Data (Database), 3.7 (keseluruhan), 4.4 (keseluruhan)
Black-box Testing, 8.3.1
Bug, 1.3, 4.7.3, 4.9 (keseluruhan), 8.5
Bug Life Cycle, 8.5.2

C

Class Diagram (Diagram Kelas), 3.3.2, 3.4 (keseluruhan)
Clean Code, 4.0, 4.1.2
Coding, 1.1, 4.1 (keseluruhan)
Commit (Git), 4.8.1, 4.8.2
Composer, 4.2.2, 4.10.3
CRUD (Create, Read, Update, Delete), 4.4.4
CSS (Cascading Style Sheets), 4.6.1

D

Debugging, 4.7.3, 4.9 (keseluruhan)
Deployment, 1.1, 1.2
Desain Antarmuka (UI/UX), 3.8 (keseluruhan)
Desain Sistem, Bab 3 (keseluruhan)

PROFIL PENULIS

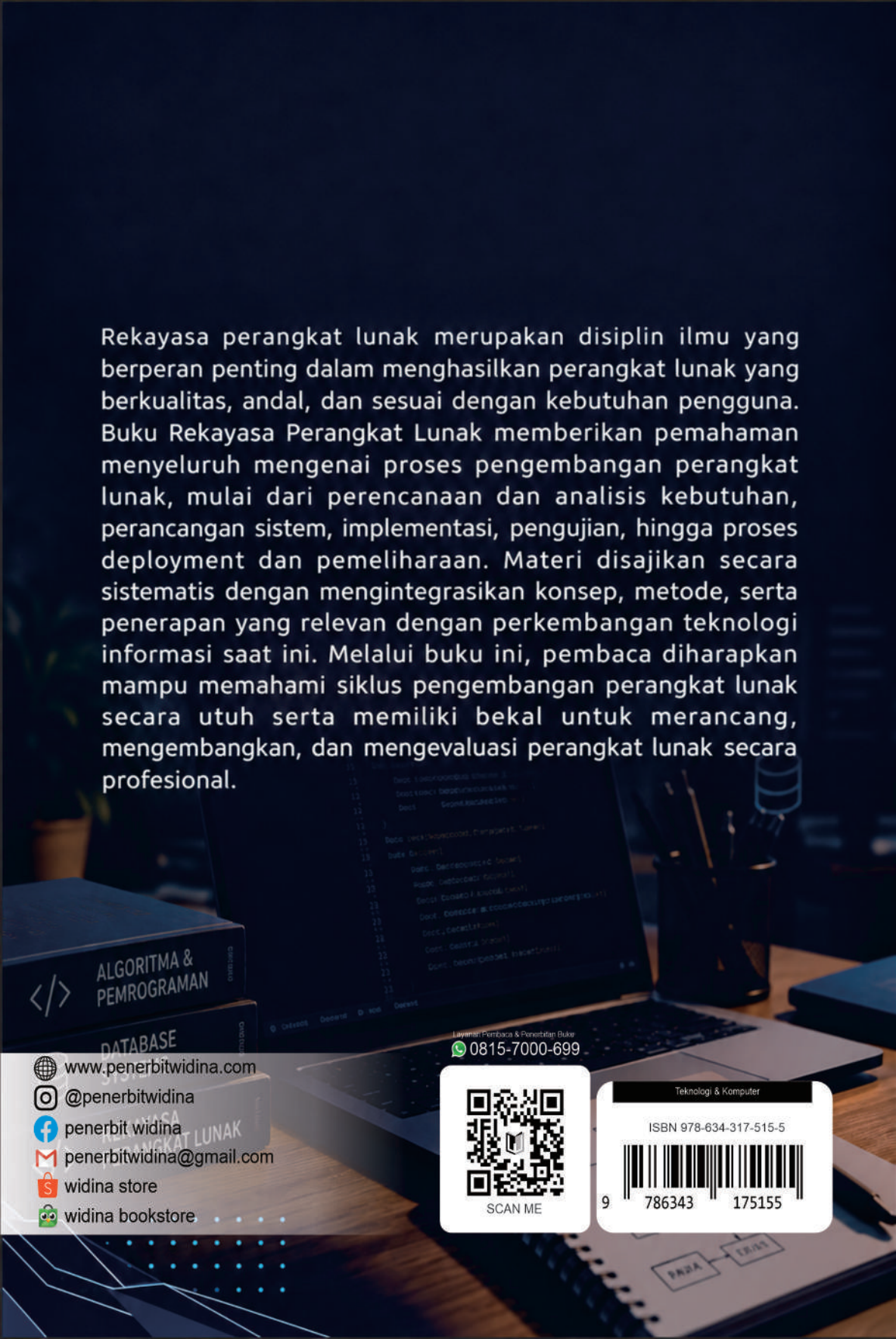


Melda Dahoklory, S.Kom., M.T. lahir di Kepulauan Aru (Dobo) pada tanggal 11 Maret 1988. Penulis menyelesaikan pendidikan Sarjana (S1) pada Program Studi Teknik Informatika di STMIK Akakom Yogyakarta (yang kini bernama Universitas Teknologi Digital Indonesia) pada tahun 2010. Selanjutnya, saya melanjutkan 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).

Rekayasa perangkat lunak merupakan disiplin ilmu yang berperan penting dalam menghasilkan perangkat lunak yang berkualitas, andal, dan sesuai dengan kebutuhan pengguna. Buku Rekayasa Perangkat Lunak memberikan pemahaman menyeluruh mengenai proses pengembangan perangkat lunak, mulai dari perencanaan dan analisis kebutuhan, perancangan sistem, implementasi, pengujian, hingga proses deployment dan pemeliharaan. Materi disajikan secara sistematis dengan mengintegrasikan konsep, metode, serta penerapan yang relevan dengan perkembangan teknologi informasi saat ini. Melalui buku ini, pembaca diharapkan mampu memahami siklus pengembangan perangkat lunak secara utuh serta memiliki bekal untuk merancang, mengembangkan, dan mengevaluasi perangkat lunak secara profesional.



Layanan Pembaca & Penerbitan Buku
0815-7000-699

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

SCAN ME

Teknologi & Komputer

ISBN 978-634-317-515-5

9 786343 175155