REKAYASA PERANGKAT LUNAK

Era digital yang dipenuhi dengan inovasi teknologi, perangkat lunak telah menjadi komponen vital dalam kehidupan sehari-hari kita. Dari perangkat mobile hingga sistem enterprise yang kompleks, perangkat lunak mendefinisikan cara kita berkomunikasi, bekerja, dan berinteraksi dengan dunia di sekitar kita. Di balik setiap aplikasi atau sistem yang kita gunakan, terdapat proses yang kompleks dan terstruktur yang dikenal sebagai rekayasa perangkat lunak.

Rekayasa perangkat lunak adalah disiplin ilmu yang berfokus pada pengembangan, pemeliharaan, dan evolusi perangkat lunak dengan menggunakan pendekatan sistematis, metodologi, dan praktik-praktik terbaik. Ini melibatkan sejumlah tahapan, mulai dari analisis kebutuhan, desain, pengkodean, pengujian, hingga pemeliharaan, yang bertujuan untuk menghasilkan perangkat lunak yang berkualitas tinggi, dapat diandalkan, dan memenuhi kebutuhan pengguna.

Buku ini membahas tentang Konsep Konsep dasar Rekayasa Perangkat Lunak, Tahap-Tahap Pengembangan Perangkat Lunak, Metodologi Pengembangan Perangkat Lunak, Kebutuhan Perangkat Lunak, Tools Rekayasa Perangkat Lunak, Desain Perangkat Lunak, Pengujian Perangkat Lunak, Pemelihara- an Perangkat Lunak.

PT MAFY MEDIA LITERASI INDONESIA
ANGGOTA IKAPI 041/SBA/2023
Email : penerbitmafy@gmail.com
Website : penerbitmafy.com
FB : Penerbit Mafy

PENDIDIKAN
ISBN 978-623-8638-03-1
9 786238 638031

REKAYASA PERANGKAT LUNAK

Jr. Mursalim Tonggiroh, S.Kom., M.Eng.
Victor Benny Alexsius Pardosi, S.Kom., M.Sc.
Basiroh, S.Kom., M.Kom
Fifto Nugroho, S.T., M.Kom.

REKAYASA PERANGKAT LUNAK

REKAYASA
PERANGKAT LUNAK

Sanksi Pelanggaran Pasal 113
Undang-Undang No. 28 Tahun 2014 Tentang Hak Cipta

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp 100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp 1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp 4.000.000.000,00 (empat miliar rupiah).

REKAYASA
PERANGKAT LUNAK

Ir. Mursalim Tonggiroh, S.Kom., M.Eng.
Victor Benny Alexsius Pardosi, S.Kom., M.Sc.
Basiroh, S.Kom., M.Kom
Fifto Nugroho, S.T., M.Kom.

MAFY media literasi INDONESIA

REKAYASA PERANGKAT LUNAK

Penulis:
Ir. Mursalim Tonggiroh, S.Kom., M.Eng.
Victor Benny Alexsius Pardosi, S.Kom., M.Sc.
Basiroh, S.Kom., M.Kom
Fifto Nugroho, S.T., M.Kom.

Editor:
Andi Asari, S.Kom.,M.A dan Shela Zahidah Wandani, S.IP.

Desainer:
Mafy Media

Sumber Gambar Cover:
www.freepik.com

Ukuran:
iv, 100 hlm, 15,5 cm x 23 cm

ISBN:
978-623-8638-03-1

Cetakan Pertama:
April 2024

PENERBIT PT MAFY MEDIA LITERASI INDONESIA
ANGGOTA IKAPI 041/SBA/2023
Kota Solok, Sumatera Barat, Kode Pos 27312
Kontak: 081374311814
Website: www.penerbitmafy.com
E-mail: penerbitmafy@gmail.com

Kata Pengantar.

Segala puji syukur kami panjatkan kepada Tuhan yang maha Esa, karena atas pertolongan dan limpahan rahmatnya sehingga penulis bisa menyelesaikan buku yang berjudul Rekayasa Perangkat Lunak. Buku ini di susun secara lengkap dengan tujuan untuk memudahkan para pembaca memahami isi buku ini. Buku ini membahas tentang Konsep Konsep dasar Rekayasa Perangkat Lunak, Tahap-Tahap Pengembangan Perangkat Lunak, Metodologi Pengembangan Perangkat Lunak, Kebutuhan Perangkat Lunak, Tools Rekayasa Perangkat Lunak, Desain Perangkat Lunak, Pengujian Perangkat Lunak, Pemeliharaan Perangkat Lunak.

Kami menyadari bahwa buku yang ada ditangan pembaca ini masih banyak kekurangan. Maka dari itu kami sangat mengharapkan saran untuk perbaikan buku ini dimasa yang akan datang. Dan tidak lupa kami mengucapkan terimakasih kepada semua pihak yang telah membantu dalam proses penerbitan buku ini. Semoga buku ini dapat membawa manfaat dan dampak positif bagi para pembaca.

Penulis, 21 Maret 2024

Daftar Isi.

PENDAHULUAN

Era digital yang dipenuhi dengan inovasi teknologi, perangkat lunak telah menjadi komponen vital dalam kehidupan sehari-hari kita. Dari perangkat mobile hingga sistem enterprise yang kompleks, perangkat lunak mendefinisikan cara kita berkomunikasi, bekerja, dan berinteraksi dengan dunia di sekitar kita. Di balik setiap aplikasi atau sistem yang kita gunakan, terdapat proses yang kompleks dan terstruktur yang dikenal sebagai rekayasa perangkat lunak.

Rekayasa perangkat lunak adalah disiplin ilmu yang berfokus pada pengembangan, pemeliharaan, dan evolusi perangkat lunak dengan menggunakan pendekatan sistematis, metodologi, dan praktik-praktik terbaik. Ini melibatkan sejumlah tahapan, mulai dari analisis kebutuhan, desain, pengkodean, pengujian, hingga pemeliharaan, yang bertujuan untuk menghasilkan perangkat lunak yang berkualitas tinggi, dapat diandalkan, dan memenuhi kebutuhan pengguna.

Buku ini akan mengulas secara menyeluruh tentang rekayasa perangkat lunak, mulai dari konsep dasarnya hingga tren terkini dalam praktik pengembangan perangkat lunak. Kami akan menjelajahi berbagai aspek dari rekayasa perangkat lunak, termasuk metodologi pengembangan, pengujian perangkat lunak, manajemen proyek, keamanan perangkat lunak, dan peran

pentingnya dalam menggerakkan inovasi teknologi di masa depan.mDengan pemahaman yang mendalam tentang rekayasa perangkat lunak, pembaca akan dapat menghargai kompleksitas dan tantangan yang terlibat dalam menciptakan perangkat lunak yang kuat, handal, dan dapat memenuhi kebutuhan masyarakat modern. Buku ini juga akan menyoroti peran penting rekayasa perangkat lunak dalam mendorong kemajuan teknologi, menginspirasi inovasi, dan memungkinkan kita untuk terhubung dengan dunia secara lebih efisien dan efektif.

BAB. 1

KONSEP DASAR REKAYASA PERANGKAT LUNAK

1.1 Definisi Rekayasa Perangkat Lunak

Rekayasa Perangkat Lunak (*Software Engineering*) adalah disiplin ilmu yang mencakup proses sistematis, metodologi, dan praktik-praktik terkait untuk merancang, mengembangkan, memelihara, dan mengelola perangkat lunak dengan tujuan menciptakan solusi perangkat lunak yang efisien, andal, mudah dipelihara, dan sesuai dengan kebutuhan pengguna serta persyaratan bisnis yang relevan (Pressman & Bruce R. Maxim, 2014). Disiplin ini melibatkan penggunaan pendekatan ilmiah dan teknik-teknik yang telah terbukti untuk mengatasi kompleksitas dalam pengembangan perangkat lunak. Proses rekayasa perangkat lunak mencakup beberapa tahap, mulai dari analisis kebutuhan, perancangan sistem, implementasi, pengujian, hingga pemeliharaan perangkat lunak yang telah dikembangkan.

Pentingnya rekayasa perangkat lunak tidak hanya terletak pada penciptaan solusi perangkat lunak yang berfungsi, tetapi juga dalam memastikan bahwa solusi tersebut memenuhi standar kualitas yang ditetapkan, seperti keamanan, kinerja, dan keandalan. Selain itu, dalam konteks pengembangan perangkat

lunak yang semakin kompleks dan berkelanjutan, praktik rekayasa perangkat lunak juga mencakup penggunaan metodologi pengembangan yang tepat, manajemen proyek yang efektif, serta pemahaman yang mendalam tentang aspek-aspek seperti arsitektur perangkat lunak, pengujian otomatis, dan manajemen konfigurasi. Dengan memadukan prinsip-prinsip ilmiah, kreativitas, dan pemahaman mendalam tentang kebutuhan pengguna, rekayasa perangkat lunak bertujuan untuk menyediakan solusi yang inovatif, andal, dan dapat memenuhi tantangan-tantangan dalam lingkungan teknologi informasi yang terus berkembang.

1.1.1 Tujuan Rekayasa Perangkat Lunak

Tujuan utama dari Rekayasa Perangkat Lunak adalah untuk mengembangkan perangkat lunak yang berkualitas tinggi secara efisien. Ini melibatkan aplikasi metode, praktik, dan proses yang sistematis untuk merancang, mengembangkan, memelihara, dan memperbarui perangkat lunak. Tujuan utama meliputi memenuhi kebutuhan pengguna dengan akurat, menjaga keandalan, kinerja, dan keamanan sistem, serta memastikan bahwa perangkat lunak dapat berkembang dan beradaptasi dengan perubahan kebutuhan dan lingkungan yang berkembang (Braude & Bernstein, 2016). Rekayasa perangkat lunak juga bertujuan untuk meningkatkan produktivitas pengembangan dengan mengadopsi praktik terbaik, alat, dan teknologi yang sesuai. Selain itu, tujuan lainnya adalah meningkatkan kualitas dan keandalan perangkat lunak dengan pengujian yang cermat dan jaminan kualitas yang terus-menerus. Selain itu, dalam era digital yang terus berkembang, tujuan rekayasa perangkat lunak juga melibatkan inovasi, adaptasi terhadap perkembangan teknologi baru, dan mempertahankan daya saing produk di pasar yang semakin kompetitif. Dengan memperhatikan semua tujuan ini, rekayasa perangkat lunak bertujuan untuk menghasilkan solusi perangkat lunak yang memenuhi kebutuhan pengguna, berkualitas tinggi, dan dapat memberikan nilai tambah bagi organisasi dan masyarakat secara keseluruhan.

1.1.2 Contoh Rekayasa Perangkat Lunak

Salah satu contoh nyata dari rekayasa perangkat lunak adalah pengembangan aplikasi mobile untuk platform Android. Proses dimulai dengan analisis kebutuhan yang melibatkan pemahaman mendalam tentang keinginan pengguna dan tujuan bisnis aplikasi tersebut. Setelah itu, tim pengembang merancang arsitektur perangkat lunak yang sesuai, menentukan fitur-fitur utama, dan merencanakan alur pengguna. Tahap implementasi melibatkan penulisan kode sumber menggunakan bahasa pemrograman Java atau Kotlin, mengintegrasikan berbagai komponen seperti antarmuka pengguna, database, dan logika bisnis. Selanjutnya, dilakukan pengujian secara menyeluruh untuk memastikan kualitas dan keandalan aplikasi, termasuk pengujian fungsional, pengujian kinerja, dan pengujian keamanan. Selama proses pengembangan, penggunaan sistem kontrol versi seperti Git memungkinkan kolaborasi tim yang efisien dan pemeliharaan riwayat perubahan kode. Setelah aplikasi selesai dikembangkan, pemeliharaan rutin diperlukan untuk memperbaiki bug, menyesuaikan fitur dengan umpan balik pengguna, dan memperbarui aplikasi sesuai dengan perubahan pada platform Android. Dengan menerapkan praktik rekayasa perangkat lunak yang baik, aplikasi mobile Android dapat berhasil memenuhi kebutuhan pengguna, memberikan pengalaman pengguna yang memuaskan, dan memberikan nilai tambah bagi organisasi yang mengembangkannya.

1.2 *Requirement Analysis* (Analisis Kebutuhan)

Analisis kebutuhan (*Requirement Analysis*) adalah tahap kritis dalam proses rekayasa perangkat lunak di mana kebutuhan sistem secara komprehensif dipahami, didokumentasikan, dan disetujui oleh semua pihak terkait. Proses ini dimulai dengan identifikasi pemangku kepentingan dan pemahaman mendalam tentang tujuan bisnis serta tantangan yang ingin diatasi oleh perangkat lunak yang akan dikembangkan (Lamsweerde, 2018). Tim analisis kebutuhan bekerja sama dengan pemangku kepentingan untuk mengumpulkan, menganalisis, dan merumus-

kan kebutuhan fungsional dan non-fungsional yang jelas dan terperinci. Ini melibatkan teknik seperti wawancara, observasi, dan studi dokumen untuk memahami proses bisnis yang ada dan mengidentifikasi kebutuhan pengguna yang sebenarnya.

Selain itu, analisis kebutuhan juga memperhatikan aspek keamanan, keandalan, kinerja, dan skalabilitas yang dapat memengaruhi desain dan implementasi sistem. Dokumen kebutuhan yang dihasilkan, seperti dokumen spesifikasi kebutuhan, diagram use case, dan skenario penggunaan, menjadi panduan untuk tim pengembangan dalam merancang dan mengembangkan solusi perangkat lunak yang sesuai. Pentingnya analisis kebutuhan tidak hanya terbatas pada tahap awal pengembangan, tetapi juga berlanjut sepanjang siklus hidup pengembangan perangkat lunak, dengan revisi dan iterasi yang diperlukan seiring perubahan kebutuhan atau pemahaman yang lebih baik tentang sistem yang akan dikembangkan. Kesalahan atau kekurangan dalam analisis kebutuhan dapat memiliki dampak yang signifikan, termasuk biaya tambahan, penundaan proyek, atau bahkan kegagalan sistem. Oleh karena itu, analisis kebutuhan yang teliti dan komprehensif merupakan langkah penting dalam memastikan kesuksesan proyek pengembangan perangkat lunak.

1.3 *Design* (Perancangan)

Perancangan (*Design*) adalah tahap kritis dalam proses rekayasa perangkat lunak di mana konsep-konsep dan keputusan yang telah dihasilkan dari analisis kebutuhan diterjemahkan menjadi struktur yang konkret dan rinci untuk sistem yang akan dibangun (Gamma et al., 1994). Proses perancangan melibatkan pemodelan sistem secara komprehensif, identifikasi komponen-komponen utama, dan menentukan hubungan dan interaksi antara komponen-komponen tersebut. Hal ini dilakukan untuk memastikan bahwa sistem yang akan dibangun memenuhi kebutuhan pengguna dengan akurat, efisien, dan dapat diandalkan.

Selama tahap perancangan, beberapa aspek penting harus dipertimbangkan, termasuk arsitektur perangkat lunak, antarmuka pengguna, basis data, dan algoritma yang akan digunakan. Arsitektur perangkat lunak menggambarkan struktur sistem secara keseluruhan, termasuk komponen-komponen utama dan hubungan antara mereka. Antarmuka pengguna dirancang untuk memastikan pengalaman pengguna yang intuitif dan efisien, sementara desain basis data berfokus pada struktur dan hubungan data yang diperlukan untuk mendukung fungsionalitas sistem. Selain itu, perancangan juga melibatkan pemilihan teknologi dan alat yang sesuai untuk implementasi sistem. Keputusan ini didasarkan pada kriteria seperti kebutuhan fungsional dan non-fungsional, kemampuan tim pengembangan, dan ketersediaan sumber daya. Penggunaan pola desain dan prinsip rekayasa perangkat lunak yang baik juga menjadi bagian integral dari proses perancangan untuk memastikan bahwa solusi yang dihasilkan bersifat skalabel, mudah dipelihara, dan dapat berkembang seiring waktu.

Hasil dari tahap perancangan adalah dokumen desain yang mendetail, termasuk diagram arsitektur, diagram kelas, diagram urutan, dan spesifikasi teknis lainnya. Dokumen ini menjadi panduan untuk implementasi sistem dalam tahap berikutnya. Pentingnya perancangan yang baik tidak hanya terletak pada memastikan sistem yang dibangun sesuai dengan kebutuhan dan spesifikasi yang telah ditetapkan, tetapi juga pada kemampuannya untuk beradaptasi dengan perubahan kebutuhan dan lingkungan yang berkembang. Dengan demikian, perancangan yang cermat adalah langkah penting dalam memastikan kesuksesan proyek pengembangan perangkat lunak secara keseluruhan.

1.4 *Implementation* (Implementasi)

Implementasi (*Implementation*) adalah tahap penting dalam proses rekayasa perangkat lunak di mana desain yang telah dibuat sebelumnya diterjemahkan menjadi kode program yang dapat dieksekusi oleh komputer. Proses implementasi melibatkan

penulisan, pengujian, dan integrasi komponen-komponen perangkat lunak untuk membentuk sistem yang lengkap dan berfungsi. Tim pengembang bekerja sesuai dengan spesifikasi teknis dan desain yang telah ditetapkan dalam tahap sebelumnya untuk menghasilkan kode yang efisien, dapat dipelihara, dan sesuai dengan standar industri. Selama tahap implementasi, pengembang menggunakan bahasa pemrograman yang telah ditentukan dan alat pengembangan yang sesuai untuk menerjemahkan desain menjadi kode sumber. Proses ini melibatkan penulisan fungsi, kelas, dan modul yang sesuai dengan tugas dan fungsionalitas yang ditentukan. Selain menulis kode, pengembang juga bertanggung jawab untuk mengimplementasikan unit pengujian untuk memastikan bahwa setiap komponen berfungsi dengan benar secara terisolasi sebelum diintegrasikan ke dalam sistem yang lebih besar (Martin, 2008).

Pada tahap ini, penerapan praktik pengembangan perangkat lunak yang baik, seperti pemrograman berorientasi objek, komentar kode yang jelas, dan dokumentasi yang akurat, sangat penting. Selain itu, penggunaan alat pengembangan yang canggih dan sistem kontrol versi memungkinkan pengembang untuk mengelola kode secara efisien, berkolaborasi dengan anggota tim, dan melacak perubahan yang dilakukan. Setelah selesai, kode yang dihasilkan akan melewati serangkaian pengujian untuk memastikan keandalan, kinerja, dan keamanannya. Ini termasuk pengujian unit, pengujian integrasi, dan pengujian sistem secara keseluruhan. Hasil dari tahap implementasi adalah sistem perangkat lunak yang lengkap, siap untuk diuji lebih lanjut dan diimplementasikan ke lingkungan produksi. Implementasi yang berhasil membutuhkan kerja tim yang terkoordinasi, keterampilan teknis yang kuat, dan pemahaman mendalam tentang spesifikasi dan desain sistem. Dengan mematuhi praktik terbaik dan standar industri, tahap implementasi dapat berhasil menghasilkan produk perangkat lunak yang berkualitas tinggi dan memenuhi kebutuhan pengguna dengan baik.

1.5 *Testing* (Pengujian)

Pengujian (*Testing*) adalah tahap penting dalam proses rekayasa perangkat lunak di mana kualitas, kinerja, dan keandalan sistem dievaluasi secara menyeluruh sebelum diluncurkan ke lingkungan produksi. Tujuan utama dari pengujian adalah untuk menemukan bug, kesalahan logika, dan ketidaksesuaian dengan spesifikasi yang mungkin ada dalam perangkat lunak, serta memastikan bahwa sistem berfungsi sesuai dengan harapan dan kebutuhan pengguna. Tahap pengujian melibatkan serangkaian aktivitas, mulai dari perencanaan pengujian hingga pelaksanaan dan evaluasi hasilnya. Perencanaan pengujian melibatkan pengembangan strategi pengujian, pembuatan kasus uji, dan penentuan lingkup dan prioritas pengujian. Kasus uji dirancang untuk mencakup berbagai skenario penggunaan yang mungkin terjadi, termasuk situasi normal dan ekstrem, serta kondisi yang berpotensi menyebabkan kegagalan sistem (Desikan & Ramesh, 2006).

Selama tahap implementasi, kasus uji dijalankan oleh tim pengujian untuk menguji fungsionalitas, kinerja, keamanan, dan keandalan sistem. Ini meliputi pengujian unit untuk menguji komponen individu, pengujian integrasi untuk menguji interaksi antara komponen, dan pengujian sistem untuk menguji keseluruhan sistem dalam lingkungan yang mirip dengan produksi. Hasil pengujian, termasuk bug dan masalah yang ditemukan, didokumentasikan secara cermat dan dilaporkan kepada tim pengembangan untuk diperbaiki. Proses ini sering melibatkan siklus pengujian dan perbaikan berulang, di mana perangkat lunak diperbaiki, pengujian ulang dilakukan, dan siklus berlanjut hingga semua masalah teratasi.

Selain pengujian fungsional, penting juga untuk melakukan pengujian non-fungsional, seperti pengujian kinerja, keamanan, dan skalabilitas. Ini bertujuan untuk memastikan bahwa perangkat lunak mampu menangani beban kerja yang diharapkan, menjaga keamanan data pengguna, dan beroperasi secara efisien dalam skala yang diperlukan. Pengujian yang efektif membutuhkan

penggunaan alat pengujian yang tepat, pengaturan lingkungan pengujian yang sesuai, dan kerja tim yang terkoordinasi antara pengembang dan pengujian. Dengan memperhatikan semua aspek ini, pengujian dapat membantu memastikan bahwa perangkat lunak yang dihasilkan memenuhi standar kualitas yang tinggi, memenuhi kebutuhan pengguna, dan dapat diandalkan saat digunakan dalam lingkungan produksi.

1.6 Pemeliharaan (Maintenance)

Pemeliharaan (*Maintenance*) merupakan tahap yang krusial dalam siklus hidup perangkat lunak di mana perangkat lunak yang telah dikembangkan dipelihara dan diperbaiki untuk memastikan kinerjanya tetap optimal dan relevan seiring berjalannya waktu. Tahap ini menjadi penting karena perubahan kebutuhan pengguna, lingkungan teknologi yang berubah, serta penemuan bug atau kelemahan baru yang mungkin muncul setelah perangkat lunak diluncurkan (April & Abran, 2008).

Pemeliharaan dapat dibagi menjadi beberapa jenis, termasuk pemeliharaan perbaikan (*corrective maintenance*), pemeliharaan adaptif (*adaptive maintenance*), pemeliharaan preventif (*preventive maintenance*), dan pemeliharaan perbaikan evolusioner (*evolutionary maintenance*). Pemeliharaan perbaikan melibatkan perbaikan bug dan kelemahan yang ditemukan setelah perangkat lunak diluncurkan, sementara pemeliharaan adaptif berkaitan dengan penyesuaian perangkat lunak terhadap perubahan lingkungan atau kebutuhan bisnis. Pemeliharaan preventif bertujuan untuk mencegah masalah dengan melakukan pembaruan atau perubahan sebelum terjadi kegagalan, sedangkan pemeliharaan perbaikan evolusioner melibatkan peningkatan atau perluasan fungsionalitas perangkat lunak berdasarkan umpan balik pengguna dan perkembangan teknologi baru.

Selama tahap pemeliharaan, perubahan atau perbaikan yang diperlukan didokumentasikan dengan baik dan diprioritaskan berdasarkan urgensi dan dampaknya terhadap sistem. Tim pengembangan bekerja sama dengan pemangku kepentingan

untuk memahami dan mengevaluasi setiap perubahan yang diajukan, dan kemudian melakukan implementasi perubahan sesuai dengan proses pengembangan perangkat lunak yang telah ditetapkan. Selain perbaikan dan penyesuaian, pemeliharaan juga mencakup tindakan pemantauan dan pengawasan yang teratur terhadap kinerja sistem secara keseluruhan. Ini melibatkan pemantauan kinerja, penggunaan, dan keamanan sistem untuk mendeteksi potensi masalah sebelum mereka menjadi serius. Tindakan ini dapat membantu dalam mencegah kegagalan sistem dan mempertahankan keandalan operasional yang tinggi.

Pentingnya pemeliharaan dalam siklus hidup perangkat lunak tidak boleh diabaikan. Dengan melakukan pemeliharaan yang teratur dan tepat waktu, perangkat lunak dapat tetap relevan, efisien, dan aman digunakan dalam jangka waktu yang panjang. Hal ini juga membantu organisasi untuk mengurangi risiko downtime yang tidak diinginkan dan biaya tambahan yang terkait dengan perbaikan darurat atau kegagalan sistem. Oleh karena itu, pemeliharaan adalah investasi yang krusial untuk memastikan kesuksesan dan kinerja perangkat lunak dalam jangka waktu yang panjang.

1.7 *Version Control* (Kontrol Versi)

Kontrol Versi (*Version Control*) adalah sistem yang memungkinkan pengembang perangkat lunak untuk melacak perubahan dalam kode sumber selama pengembangan perangkat lunak. Ini memungkinkan pengembang untuk menyimpan riwayat perubahan, melacak siapa yang melakukan perubahan, kapan perubahan dilakukan, dan mengembalikan kode ke versi sebelumnya jika diperlukan. Dengan menggunakan kontrol versi seperti Git, pengembang dapat bekerja secara kolaboratif, mengelola kode dengan lebih efisien, dan mengurangi risiko kehilangan kode atau konflik yang tidak diinginkan (Loeliger & McCullough, 2012).

BAB. 2

TAHAP-TAHAP PENGEMBANGAN PERANGKAT LUNAK

2.1 Analisis Kebutuhan

Analisis Persyaratan/Kebutuhan Perangkat Lunak (*Software Requirements Analysis*/*SRA*) terdiri dari berbagai aktivitas seperti perolehan persyaratan, analisis, pemodelan, dan manajemen. Ini adalah fase penting dalam siklus hidup pengembangan perangkat lunak yang memastikan keberhasilan penataan sistem perangkat lunak dan mencakup identifikasi, dokumentasi, dan pengaturan persyaratan sistem perangkat lunak. Kualitas *SRA* sangat dipengaruhi oleh pengembangan, pembangunan, dan pemodelan perangkat lunak (Deshpande & Mangalwede, 2018; Kun-wu, 2013). Salah satu fase penting dalam pengembangan perangkat lunak adalah proses analisis kebutuhan, yang biasanya terdiri dari tahapan berikut (Ali & Lai, 2017; Poo, 1992; Saavedra et al., 2013):

1. *Requirement Elication* adalah proses pengumpulan persyaratan dari pemangku kepentingan. Ini melibatkan memahami apa yang dibutuhkan, apa yang diharapkan, dan apa yang menjadi kendala pengguna. Pada tahap ini, wawancara, observasi, survei, dan *focus group* digunakan.

2. Analisis Persyaratan: Pada tahap ini, persyaratan yang dikumpulkan pada tahap sebelumnya dievaluasi. Ini mencakup menemukan persyaratan yang tidak konsisten, ambigu, atau konflik. Selain itu, persyaratan diprioritaskan berdasarkan kepentingan dan kelayakannya.
3. Pemodelan Persyaratan: Tahap ini melibatkan pembuatan model sistem yang disesuaikan dengan kebutuhan. Model ini digunakan untuk memahami struktur, perilaku, dan interaksi sistem, dan membantu dalam mengidentifikasi bagian sistem dan hubungannya satu sama lain.
4. Manajemen Persyaratan: Tahap ini mengatur persyaratan sepanjang proses pengembangan perangkat lunak. Ini termasuk mencatat perubahan persyaratan, memastikan bahwa persyaratan dipenuhi, dan memastikan bahwa persyaratan dipenuhi.
5. Validasi dan Verifikasi: Ini adalah tahap terakhir dari Proses Analisis Kebutuhan. Ini mencakup mengevaluasi apakah persyaratan terpenuhi dan apakah sistem beroperasi sesuai dengan harapan. Ini termasuk membuat penyesuaian dan menguji sistem untuk memenuhi persyaratan.

Untuk memahami kebutuhan, harapan, dan kendala pengguna, teknik pengumpulan informasi untuk analisis kebutuhan perangkat lunak sangat penting. Beberapa metode yang umum termasuk (Ko, 1999; Moore & Shipman, 2001):

1. Wawancara terstruktur melibatkan pertanyaan terbuka untuk meminta informasi dari pemangku kepentingan. Anda dapat melakukan wawancara terstruktur satu lawan satu atau kelompok.
2. Sesi Kelompok: Metode ini melibatkan pemangku kepentingan untuk berkumpul untuk berbicara tentang kebutuhan dan harapan mereka. Sesi kelompok dapat membantu menemukan masalah yang sama dan ketidakkonsistenan dalam persyaratan.
3. Analisis Skenario: Ini adalah proses membuat situasi teoretis untuk mengetahui bagaimana pengguna akan berinteraksi

dengan sistem. Analisis skenario dapat membantu menemukan masalah potensial dan persyaratan.

4. Kuesioner: Ini digunakan untuk mengetahui kebutuhan dan harapan pemangku kepentingan dengan mengirimkan serangkaian pertanyaan kepada mereka. Ini dapat digunakan dalam kasus di mana komunikasi tatap muka tidak mungkin.
5. Antarmuka Pengguna Grafis (GUI): Ini melibatkan pengumpulan persyaratan melalui representasi visual sistem. Untuk pengumpulan kebutuhan awal dan memungkinkan "*design by doing*", GUI dapat membantu berkomunikasi.
6. Pemodelan Objek: Pemodelan objek melibatkan pembuatan model domain masalah terkait objek, klasifikasinya, dan hubungan antar objek. Ini dapat membantu menemukan struktur sistem dan kebutuhan fungsional.
7. Pemodelan Persyaratan Fungsional: Ini melibatkan pembuatan model objek awal untuk melengkapi spesifikasi analisis persyaratan. Ini membantu dalam menentukan kebutuhan fungsional sistem.

Teknik ini digunakan untuk mengumpulkan kebutuhan informasi dan memastikan proses pengembangan perangkat lunak berjalan dengan baik. Persyaratan yang menentukan tindakan atau perilaku sistem disebut persyaratan fungsional. Persyaratan fungsional umumnya adalah spesifik, terukur, dapat dicapai, relevan, dan terikat waktu (SMART). Contoh persyaratan fungsional ini meliputi:

1. Sistem harus menampilkan informasi profil pengguna
2. Sistem harus memungkinkan pengguna untuk mencari informasi.
3. Sistem harus menyediakan fitur untuk menyimpan dan mengambil informasi.

Sebaliknya, persyaratan non-fungsional adalah persyaratan yang menunjukkan kualitas sistem atau seberapa baik kinerjanya. Persyaratan ini tidak langsung terkait dengan fungsionalitas sistem, tetapi dengan pengalaman pengguna atau kinerja sistem. Contoh persyaratan non-fungsional meliputi:

1. Sistem harus ramah pengguna.
2. Sistem harus aman.
3. Sistem harus dapat diskalakan.

Penting untuk mengidentifikasi kebutuhan fungsional dan non-fungsional selama proses analisis kebutuhan perangkat lunak untuk memastikan bahwa sistem memenuhi kebutuhan dan harapan pengguna (Kaur & Aggarwal, 2023; Rashwan, 2012).

2.2 Perancangan Sistem

Proses membuat sistem perangkat lunak yang memenuhi kebutuhan dan harapan pengguna dikenal sebagai desain sistem perangkat lunak. Proses ini mencakup konsep-konsep berikut (Klüter et al., 2000; Muqsith & Sarjoughian, 2010):

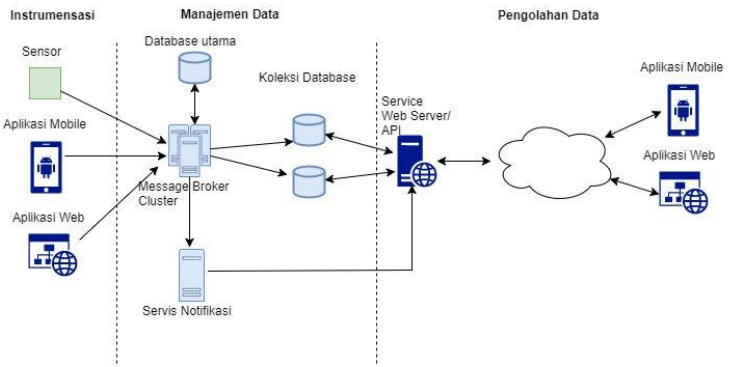
1. Arsitektur Berorientasi Layanan (SOA) adalah pendekatan desain yang menggambarkan sistem perangkat lunak sebagai kumpulan layanan yang digabungkan secara longgar yang masing-masing memiliki fitur tertentu.
2. Alat Simulasi: Perangkat lunak yang digunakan untuk menyimulasikan dan menguji perilaku sistem, terutama sistem perangkat lunak berbasis layanan.
3. Analisis Persyaratan Perangkat Lunak Berorientasi Objek: Sebuah metodologi yang memodelkan dan menganalisis domain masalah dengan menggunakan objek dan hubungan-nya. Fokus metodologi adalah pembuatan model domain masalah terkait dengan objek, klasifikasinya, dan hubungannya satu sama lain.
4. Pemodelan Persyaratan Fungsional adalah tahapan dari proses analisis persyaratan yang melengkapi spesifikasi analisis persyaratan dan mendefinisikan persyaratan fungsional suatu sistem. Ini dibangun berdasarkan model objek awal.
5. Analisis Persyaratan Perangkat Lunak: Proses mengidentifi-kasi, mencatat, dan mengatur kebutuhan sistem perangkat lunak. Ini adalah komponen penting dari keberhasilan penataan sistem perangkat lunak.

6. Teknik Pemunculan Persyaratan: Wawancara terstruktur, sesi kelompok, analisis skenario, kuesioner, dan antarmuka pengguna grafis (GUI) adalah beberapa metode yang digunakan untuk mengumpulkan kebutuhan pemangku kepentingan.
7. Manajemen Persyaratan: Proses untuk memastikan bahwa persyaratan dipenuhi selama proses pengembangan perangkat lunak. Ini termasuk mencatat perubahan dan memastikan bahwa persyaratan dipenuhi.

Konsep-konsep ini sangat penting dalam merancang sistem perangkat lunak yang memenuhi kebutuhan dan harapan pengguna serta memastikan bahwa sistem tersebut berkualitas tinggi, dapat diperluas, dan dapat dipelihara.

2.2.1 Desain Arsitektur Sistem

Desain arsitektur sistem adalah proses membuat struktur dan organisasi keseluruhan suatu sistem, termasuk komponennya, koneksinya, dan interaksinya. Proses ini juga mencakup pengambilan keputusan desain untuk setiap langkah dalam sistem, seperti otentikasi pengguna, pembuatan peringatan, komunikasi, pengakuan dan manajemen peringatan, pengingat dan eskalasi peringatan, dan dokumentasi peringatan. Tujuannya adalah untuk memenuhi persyaratan tautan komunikasi, ukuran LRM, dan toleransi kesalahan sekaligus mengurangi kompleksitas interkoneksi sistem. Sistem yang dirancang dengan baik dioptimalkan dan disesuaikan untuk penggunaan tertentu dan terintegrasi dengan proses servis dan pemeliharaan yang mendukung sistem sepanjang siklus hidupnya.



Gambar 1 Contoh Desain Arsitektur Sistem
Sumber: Internet

Secara umum, proses desain arsitektur sistem terdiri dari beberapa tahap (Bryan, 2018):

1. Desain Konseptual adalah tahap awal di mana arsitektur sistem secara keseluruhan ditentukan. Ini melibatkan menentukan tujuan sistem, komponen utamanya, dan cara mereka berinteraksi satu sama lain.
2. Desain Awal: Pada tahap ini, arsitektur sistem disempurnakan dan detail komponen dan interaksinya ditentukan. Biasanya, tahap ini melibatkan pembuatan diagram dan model sistem secara menyeluruh.
3. Desain Terperinci: Tahap ini melibatkan pembuatan spesifikasi rinci untuk setiap komponen sistem. Ini termasuk menentukan antarmuka antar komponen, struktur data, dan algoritma yang digunakan dalam sistem.
4. Implementasi: Tahap ini melibatkan pembuatan sistem sesuai dengan spesifikasi desain. Ini mencakup pengkodean perangkat lunak, perakitan perangkat keras, dan mengintegrasikan komponen.
5. Pengujian: Tahap ini melibatkan pengujian sistem untuk memastikan bahwa itu berfungsi dengan benar dan memenuhi spesifikasi desain. Ini termasuk pengujian unit, pengujian integrasi, dan pengujian sistem.

6. Pemasangan dan Pemeliharaan: Setelah sistem diuji dan dianggap siap digunakan, sistem dipasang di tempat yang diinginkannya. Tahap ini juga mencakup pembaruan dan pemeliharaan berkelanjutan.

2.2.2 Desain Antarmuka Pengguna (UI/UX)

Desain antarmuka pengguna (UI) dan pengalaman pengguna (UX) adalah komponen penting dalam pembuatan aplikasi perangkat lunak atau situs web yang sukses. UI/UX mengacu pada proses pembuatan aspek visual dan interaktif dari produk digital dengan tujuan membuatnya mudah, efisien, dan menyenangkan bagi pengguna untuk berinteraksi dengannya. Dalam kebanyakan kasus, proses desain terdiri dari beberapa tahap (Marbun et al., 2022; Yehdeya et al., 2023):

1. Memahami Pengguna: Tahap ini mencakup mengidentifikasi kebutuhan dan preferensi audiens target. Ini dapat dicapai melalui pengujian, wawancara, dan survei.
2. Merancang Antarmuka Pengguna: Proses ini mencakup pembuatan *wireframe* dan maket antarmuka pengguna. Ini termasuk perancangan tata letak, skema warna, tipografi, dan elemen visual lainnya.
3. Pembuatan Prototipe: Tahap ini melibatkan membuat prototipe antarmuka pengguna yang berfungsi. Ini memungkinkan desainer untuk menguji antarmuka dan melakukan perubahan yang diperlukan.
4. Pengujian Kegunaan: Proses ini menguji prototipe dengan pengguna nyata untuk menemukan masalah kegunaan dan membuat perbaikan.
5. Iterasi dan Penyempurnaan: Berdasarkan hasil pengujian kegunaan, desain disesuaikan dan ditingkatkan. Beberapa putaran pengujian dan iterasi mungkin diperlukan dalam proses ini.
6. Implementasi: Desain diterapkan pada produk setelah selesai.
7. Pemeliharaan dan Pembaruan: Setelah produk diluncurkan, penting untuk memperbarui dan memelihara antarmuka

pengguna secara berkala untuk memastikannya tetap ramah pengguna dan relevan.

2.3 Implementasi dan Pengujian

Proses membangun dan menerapkan perangkat lunak berdasarkan spesifikasi desain dikenal sebagai tahap implementasi perangkat lunak, yang mencakup beberapa langkah penting (Abdoli, 2023; Che et al., 2001; Saeeda et al., 2020):

1. *Coding*: Ini adalah proses penulisan kode sebenarnya untuk perangkat lunak. Hal ini biasanya dilakukan dengan menggunakan bahasa pemrograman dan melibatkan penerjemahan spesifikasi desain ke dalam kode fungsional.
2. Pengujian: Setelah kode ditulis, itu diuji untuk memastikan bahwa itu beroperasi dengan benar dan memenuhi spesifikasi desain. Ini dapat termasuk pengujian unit, integrasi, dan sistem.
3. *Debugging*: Masalah atau *bug* apa pun yang ditemukan selama pengujian diperbaiki dan perangkat lunak diuji ulang, yang dapat mencakup beberapa putaran pengujian dan *debugging*.
4. Penerapan: Setelah perangkat lunak diuji dan dianggap siap digunakan, perangkat lunak diterapkan ke lingkungan yang diinginkan.
5. Pemeliharaan dan Pembaruan: Perangkat lunak harus diperbarui dan diperbarui secara rutin untuk memastikan bahwa itu berfungsi dan aman. Ini mungkin mencakup perbaikan bug, penambahan fitur baru, dan penyelesaian masalah apa pun yang muncul.

Metode pengujian perangkat lunak digunakan untuk mengevaluasi kualitas perangkat lunak dan memastikan bahwa itu memenuhi spesifikasi yang diinginkan. Ada beberapa jenis metode pengujian perangkat lunak (Mohialden et al., 2022; Mustafa et al., 2009).

1. Pengujian Unit: Teknik ini menguji setiap komponen atau unit perangkat lunak secara terpisah. Ini biasanya dilakukan oleh pengembang dan digunakan untuk memastikan bahwa setiap komponen bekerja dengan benar.

2. Pengujian Integrasi: Teknik ini digunakan setelah pengujian unit dan digunakan untuk memastikan bahwa komponen bekerja sama dengan benar.
3. Pengujian Sistem: Teknik ini menguji perangkat lunak secara keseluruhan, biasanya dilakukan oleh tim pengujian terpisah, untuk memastikan bahwa perangkat lunak memenuhi persyaratan dan beroperasi dengan benar di lingkungan yang dimaksudkan.
4. Pengujian Penerimaan: Teknik ini menguji perangkat lunak dari perspektif pengguna atau pelanggan, biasanya dilakukan oleh pengguna atau pelanggan, untuk memastikan bahwa perangkat lunak memenuhi kebutuhan dan harapan mereka.
5. Pengujian Regresi: Metode ini biasanya dilakukan setelah perubahan pada perangkat lunak dan digunakan untuk memastikan bahwa perubahan tidak menyebabkan bug atau masalah baru. Ini menguji perangkat lunak untuk memastikan bahwa perangkat lunak tetap stabil dan dapat diandalkan.
6. Pengujian Kinerja: Teknik ini menguji perangkat lunak dalam berbagai kondisi, seperti beban tinggi atau kompetisi tinggi. Tim pengujian terpisah biasanya melakukannya untuk memastikan bahwa perangkat lunak memenuhi persyaratan kinerja yang diinginkan.
7. Pengujian Keamanan: Teknik ini biasanya dilakukan oleh tim pengujian terpisah dan digunakan untuk memastikan bahwa perangkat lunak aman dan aman dari berbagai serangan.
8. Pengujian Kegunaan: Teknik ini biasanya dilakukan oleh tim pengujian terpisah dan memastikan bahwa perangkat lunak mudah digunakan dan dipahami. Ini memastikan bahwa perangkat lunak memenuhi persyaratan kegunaan yang diinginkan.
9. Pengujian Eksplorasi: Metode ini menggunakan pengujian perangkat lunak yang tidak terstruktur dan fleksibel, biasanya dilakukan oleh tim pengujian terpisah dan digunakan untuk menemukan masalah yang mungkin tidak ditemukan oleh metode pengujian terstruktur.

10. Pengujian Otomatis: Teknik ini melibatkan penggunaan alat dan skrip untuk mengotomatiskan proses pengujian; ini biasanya dilakukan oleh tim pengujian terpisah dan digunakan untuk meningkatkan akurasi dan efisiensi proses.

Selain itu, perangkat lunak seperti JUnit, TestNG, Selenium, dan Appium tersedia untuk membantu proses pengujian.

2.4 Peluncuran, Pemeliharaan, dan Perbaikan

Peluncuran merupakan tahap di mana perangkat lunak disebarkan dan di-*instal* pada platform target, termasuk konfigurasi dan pemeriksaan sistem. Pemeliharaan melibatkan upaya berkelanjutan untuk menjaga perangkat lunak tetap mutakhir dan berfungsi dengan melakukan *patching*, perbaikan *bug*, dan penyesuaian kinerja. Sedangkan perbaikan fokus pada penanganan masalah yang timbul pada perangkat lunak, termasuk pemecahan masalah, diagnosis, dan penyelesaian masalah.

BAB. 8

PEMELIHARAAN PERANGKAT LUNAK

8.1 Jenis Pemeliharaan Perangkat Lunak

Pemeliharaan perangkat lunak adalah serangkaian kegiatan yang dilakukan setelah pengembangan perangkat lunak selesai, bertujuan untuk menjaga agar perangkat lunak tetap berfungsi dengan baik, aman, dan sesuai dengan kebutuhan pengguna. Jenis-jenis pemeliharaan perangkat lunak dapat dibagi menjadi empat kategori utama:

8.1.1 Pemeliharaan Perbaikan

Pemeliharaan Perbaikan (*Corrective Maintenance*) adalah salah satu aspek penting dalam siklus hidup perangkat lunak yang berkaitan dengan mengatasi kegagalan atau *bug* yang terjadi setelah perangkat lunak diluncurkan ke pengguna. Fokus utamanya adalah memperbaiki masalah yang dapat mengganggu fungsionalitas atau kinerja perangkat lunak, sehingga pengguna dapat terus menggunakan aplikasi tersebut tanpa hambatan (Purba et al., 2019). Kegagalan atau bug ini dapat berasal dari berbagai sumber, seperti kesalahan dalam kode, kegagalan integrasi, atau ketidakcocokan dengan lingkungan operasional yang kompleks.

Proses pemeliharaan perbaikan dimulai dengan identifikasi masalah yang dilaporkan oleh pengguna atau dideteksi oleh tim pengembang melalui pemantauan atau pengujian perangkat lunak. Setelah masalah diidentifikasi, dilakukan analisis mendalam untuk memahami akar penyebabnya. Analisis ini melibatkan pemahaman terhadap kode perangkat lunak, arsitektur sistem, serta interaksi dengan komponen lainnya. Langkah selanjutnya adalah merancang dan menerapkan perbaikan yang diperlukan untuk memulihkan fungsionalitas yang hilang atau rusak. Tujuan utama dari pemeliharaan perbaikan adalah memastikan bahwa masalah yang ditemukan dalam perangkat lunak dapat segera diatasi. Hal ini penting untuk menjaga kepuasan pengguna dan reputasi perangkat lunak di pasaran. Selain itu, pemeliharaan perbaikan juga memungkinkan tim pengembang untuk memperbaiki kesalahan yang mungkin terjadi selama proses pengembangan dan memperbaiki perangkat lunak agar lebih stabil dan handal.

8.1.2 Pemeliharaan Adaptif (*Adaptive Maintenance*)

Pemeliharaan Adaptif merupakan jenis pemeliharaan perangkat lunak yang fokus pada penyesuaian perangkat lunak dengan perubahan lingkungan operasionalnya. Lingkungan operasional perangkat lunak dapat berubah dari waktu ke waktu karena faktor-faktor seperti perubahan perangkat keras, sistem operasi, atau kebijakan organisasi. Tujuan utama dari pemeliharaan adaptif adalah memastikan bahwa perangkat lunak tetap berfungsi dengan baik dan optimal dalam menghadapi perubahan-perubahan tersebut.

Perubahan perangkat keras, seperti *upgrade* atau penggantian komponen perangkat keras, seringkali memerlukan penyesuaian perangkat lunak untuk memastikan kompatibilitas dan kinerja yang optimal (Halawa, 2016). Pemeliharaan adaptif juga diperlukan ketika terjadi perubahan sistem operasi, baik karena *upgrade* versi sistem operasi yang ada maupun migrasi ke sistem operasi yang berbeda. Ini melibatkan penyesuaian kode perangkat lunak agar tetap dapat berjalan dengan baik di lingkungan sistem operasi yang baru.

8.1.3 Pemeliharaan Perfektif (*Perfective Maintenance*)

Pemeliharaan Perfektif (*Perfective Maintenance*) adalah jenis pemeliharaan perangkat lunak yang fokus pada peningkatan kinerja atau fungsionalitas perangkat lunak yang sudah ada. Proses ini melibatkan berbagai kegiatan, termasuk penambahan fitur baru, perbaikan desain, atau peningkatan performa, dengan tujuan memenuhi kebutuhan atau harapan pengguna yang berkembang. Penambahan fitur baru merupakan bagian penting dari pemeliharaan perfektif, di mana fitur-fitur ini bisa meningkatkan fungsionalitas atau nilai tambah dari perangkat lunak. Fitur baru ini bisa muncul sebagai respons terhadap umpan balik pengguna, perkembangan teknologi, atau perubahan kebutuhan pasar. Selain itu, perbaikan desain juga dapat dilakukan untuk memperbaiki struktur internal perangkat lunak, sehingga membuatnya lebih mudah dipelihara, dimengerti, dan dikembangkan di masa depan.

Peningkatan performa juga merupakan fokus utama dalam pemeliharaan perfektif. Ini bisa melibatkan optimisasi kode, algoritma, atau proses yang ada untuk meningkatkan efisiensi dan kecepatan perangkat lunak. Tujuannya adalah untuk membuat perangkat lunak berjalan lebih cepat, menggunakan sumber daya yang lebih sedikit, dan memberikan pengalaman pengguna yang lebih baik secara keseluruhan (Hende et al., 2018). Secara keseluruhan, tujuan dari pemeliharaan perfektif adalah meningkatkan kualitas dan kinerja perangkat lunak secara keseluruhan. Dengan melakukan penambahan fitur baru, perbaikan desain, dan peningkatan performa secara teratur, perangkat lunak dapat tetap relevan, kompetitif, dan berdaya saing dalam pasar yang terus berubah. Selain itu, pemeliharaan perfektif juga dapat meningkatkan kepuasan pengguna, memperkuat citra merek, dan mendukung pertumbuhan dan evolusi bisnis yang berkelanjutan.

8.1.4 Pemeliharaan Preventif (*Preventive Maintenance*)

Pemeliharaan Preventif merupakan strategi yang difokuskan pada pencegahan timbulnya masalah di masa depan dengan melakukan serangkaian kegiatan proaktif. Ini termasuk

pemeriksaan rutin, pembaruan keamanan, dan perbaikan kecil yang bertujuan untuk mencegah kerusakan atau kegagalan yang mungkin terjadi. Tujuannya adalah memastikan keandalan, keamanan, dan ketersediaan perangkat lunak dalam jangka waktu yang lebih panjang. Pemeriksaan rutin melibatkan pengawasan secara berkala terhadap perangkat lunak untuk mendeteksi potensi masalah atau kerentanan sebelum mereka menjadi masalah yang serius. Ini bisa mencakup pemeriksaan terhadap log aplikasi, pemantauan kinerja sistem, atau audit kode untuk menemukan bug atau kelemahan potensial.

Pembaruan keamanan adalah bagian penting dari pemeliharaan preventif yang bertujuan untuk memperbarui perangkat lunak dengan patch atau pemutakhiran keamanan terbaru. Dengan memperbarui perangkat lunak secara teratur, organisasi dapat mengatasi celah keamanan yang telah diidentifikasi atau yang mungkin terungkap di masa depan, mengurangi risiko serangan dan pelanggaran keamanan data. Selain itu, perbaikan kecil yang dilakukan secara preventif dapat membantu mencegah kerusakan atau kegagalan yang mungkin terjadi di masa depan. Ini bisa mencakup perbaikan bug kecil, peningkatan performa, atau penyesuaian konfigurasi untuk mengoptimalkan kinerja perangkat lunak.

8.2 Siklus Hidup Pemeliharaan Sistem SMLC

Siklus Hidup Pemeliharaan Sistem (SMLC) terdiri dari serangkaian tahapan yang membantu dalam mengelola pemeliharaan perangkat lunak secara efisien dari awal hingga akhir. Berikut adalah tahapan-tahapan yang umumnya terdapat dalam SMLC:

1. Memahami Permintaan Pemeliharaan
   Tahap pertama adalah memahami dengan baik permintaan pemeliharaan yang diajukan. Ini melibatkan identifikasi masalah atau perubahan yang diperlukan dalam sistem, baik itu berdasarkan umpan balik pengguna, masalah yang

dilaporkan, atau perubahan lingkungan yang mempe- ngaruhi sistem.

2. Mentransformasi Permintaan Pemeliharaan Menjadi Pengubahan
   Permintaan pemeliharaan yang dipahami kemudian diubah menjadi bentuk perubahan yang konkret yang dapat diimplementasikan dalam sistem (Utami et al., 2018). Hal ini melibatkan penentuan lingkup perubahan yang diperlukan dan bagaimana cara untuk mengimplemen- tasikannya.
3. Menspesifikasi Perubahan
   Tahap ini melibatkan penyusunan spesifikasi perubahan yang jelas dan terperinci. Spesifikasi ini mencakup deskripsi lengkap tentang perubahan yang akan dilakukan, termasuk fitur baru yang akan ditambahkan, perbaikan yang diperlukan, atau modifikasi yang diinginkan.
4. Mengembangkan Perubahan
   Setelah spesifikasi perubahan disetujui, tahap pengem- bangan dimulai. Ini melibatkan proses pengkodean atau pemodifikasi perangkat lunak untuk menerapkan perubahan sesuai dengan spesifikasi yang telah ditetapkan.
5. Menguji Perubahan
   Perubahan yang telah dikembangkan kemudian diuji secara menyeluruh untuk memastikan bahwa mereka berfungsi dengan baik dan tidak menyebabkan dampak negatif pada sistem. Pengujian mencakup pengujian fungsionalitas, integrasi, kinerja, dan keamanan.
6. Melatih Pengguna dan Melakukan Test Penerimaan
   Sebelum perubahan diluncurkan ke lingkungan produksi, pengguna perlu dilatih untuk menggunakan fitur baru atau perubahan yang telah diterapkan. Selain itu, tes penerimaan dilakukan untuk memastikan bahwa perubahan memenuhi kebutuhan pengguna dan kriteria penerimaan yang telah ditetapkan.
7. Pengkonversian dan Meluncurkan Operasi
   Setelah berhasil melewati tahap pengujian dan tes penerimaan, perubahan kemudian dikonversi dan diluncurkan

ke dalam lingkungan produksi. Ini melibatkan proses migrasi data, konfigurasi sistem, dan peluncuran operasional perubahan.

8. Mengupdate Dokumen
   Selama atau setelah implementasi, dokumen sistem seperti dokumentasi pengguna, panduan pengguna, atau dokumentasi teknis perlu diperbarui sesuai dengan perubahan yang telah dilakukan dalam sistem.
9. Melakukan Pemeriksaan Pasca-implementasi
   Tahap terakhir adalah melakukan pemeriksaan pasca-implementasi untuk memastikan bahwa perubahan telah diimplementasikan dengan sukses dan sistem berfungsi dengan baik setelah implementasi. Masukan dari pengguna dan pemantauan kinerja sistem dapat digunakan untuk mengidentifikasi dan menangani masalah yang mungkin timbul di masa depan.

8.3 Maintability

Peningkatan *maintainability* (kemampuan pemeliharaan sistem) adalah kunci dalam memastikan bahwa sistem perangkat lunak dapat dikelola dengan efisien dan efektif selama siklus hidupnya. Berikut adalah beberapa prosedur yang dapat diterapkan untuk meningkatkan maintainability sistem:

1. Menerapkan SDLC dan SWDLC
   Mengadopsi pendekatan Siklus Hidup Pengembangan Perangkat Lunak (SDLC) dan Siklus Hidup Pengembangan Perangkat Lunak Perangkat Keras (SWDLC) yang baik akan membantu memastikan bahwa proses pengembangan, pengujian, dan pemeliharaan sistem dilakukan secara terstruktur dan terdokumentasi dengan baik (Sari et al., 2022).
2. Menspesifikasikan Definisi Data Standar
   Menggunakan definisi data standar akan memudahkan dalam pengelolaan dan pemeliharaan sistem. Ini termasuk penentuan format dan struktur data yang konsisten, sehingga

memudahkan pemahaman dan penggunaan data di seluruh sistem.

3. Menggunakan Bahasa Pemrograman Standar
   Memilih bahasa pemrograman yang umum digunakan dan memiliki dukungan komunitas yang luas akan membantu dalam memelihara sistem. Bahasa pemrograman standar seringkali memiliki alat bantu dan dokumentasi yang melimpah, serta lebih mudah untuk menemukan pengembang yang memiliki keahlian dalam bahasa tersebut.
4. Merancang Modul-Modul yang Terstruktur dengan Baik
   Memecah sistem menjadi modul-modul yang terstruktur dengan baik akan memudahkan dalam pemeliharaan dan pengembangan lebih lanjut. Modul-modul yang terpisah dapat diubah tanpa mempengaruhi bagian lain dari sistem, sehingga meminimalkan risiko kerusakan atau kegagalan.
5. Mempekerjakan Modul yang Dapat Digunakan Kembali
   Menerapkan prinsip penggunaan kembali (*reusability*) akan mengurangi upaya pemeliharaan dengan memanfaatkan kembali modul-modul atau komponen-komponen yang telah ada dan teruji. Hal ini memungkinkan untuk menghemat waktu dan upaya dalam pengembangan serta memperbaikan sistem.
6. Mempersiapkan Dokumentasi yang Jelas, Terbaru, dan Komprehensif
   Dokumentasi yang baik sangat penting untuk memudahkan pemeliharaan sistem. Ini termasuk dokumentasi tentang arsitektur sistem, desain modul, panduan pengguna, dan petunjuk pemecahan masalah. Pastikan dokumentasi tetap terbaru dan komprehensif agar informasi yang diberikan relevan dan berguna.
7. Menginstal Perangkat Lunak, Dokumentasi, dan Soal-Soal Test di dalam Sentral Repositori Sistem CASE atau CMS
   Mengelola semua elemen sistem, termasuk perangkat lunak, dokumentasi, dan soal-soal test di dalam repositori sistem yang terpusat akan memudahkan dalam pengelolaan,

pemeliharaan, dan distribusi perubahan atau pembaruan sistem secara konsisten.

8.4 Mengelola pemeliharaan sistem

Mengelola pemeliharaan sistem merupakan aspek penting dalam menjaga kinerja dan keandalan sistem perangkat lunak. Berikut adalah beberapa langkah yang dapat dilakukan untuk mengelola pemeliharaan sistem dengan efektif:

1. Menetapkan Kegiatan Pemeliharaan Sistem
   Tentukan secara jelas kegiatan pemeliharaan sistem yang perlu dilakukan, baik itu pemeliharaan preventif, adaptif, korektif, atau perfektif. Identifikasi perangkat lunak yang memerlukan pemeliharaan, prioritasnya, serta frekuensi dan jadwal pemeliharaan yang diperlukan (Jiang, 2020).
2. Mengawali dan Merekam Kegiatan Pemeliharaan Sistem Tidak Terjadwal
   Buat formulir atau lembar kerja pemeliharaan (*Maintenance Work Order*) yang mencatat detail pekerjaan yang diperlukan atau dilakukan, perkiraan waktu yang dibutuhkan, waktu yang sebenarnya, kode pemeliharaan, dan biaya pemeliharaan. Ini membantu dalam pelacakan dan evaluasi kinerja pemeliharaan.
3. Menggunakan Sistem Perangkat Lunak *Help-Desk*
   Implementasikan sistem perangkat lunak *help-desk* untuk memfasilitasi pelaporan, pelacakan, dan penanganan permintaan pemeliharaan dari pengguna atau staf internal. Sistem ini memungkinkan untuk mengatur prioritas, menetapkan tugas, dan melacak status pemeliharaan dengan lebih efisien.
4. Mengevaluasi Aktivitas Pemeliharaan Sistem
   Secara rutin, lakukan evaluasi terhadap aktivitas pemeliharaan sistem yang telah dilakukan. Tinjau efektivitas dan efisiensi pemeliharaan, identifikasi pola kegagalan atau masalah yang sering terjadi, serta cari cara untuk meningkatkan proses pemeliharaan di masa mendatang.

5. Mengoptimalkan Program Pemeliharaan Sistem
   Gunakan hasil evaluasi untuk mengoptimalkan program pemeliharaan sistem. Identifikasi area yang membutuhkan perbaikan atau peningkatan, sesuaikan jadwal pemelihara- an, tingkatkan efisiensi dalam penanganan permintaan pemeliharaan, dan terapkan langkah-langkah untuk mencegah masalah yang sering terjadi.

8.5 Risiko CMS

Sistem Manajemen Perubahan (CMS) dapat membantu dalam menghindari berbagai risiko yang terkait dengan pengelolaan perangkat lunak dan sistem informasi. Berikut adalah beberapa risiko yang dapat dihindari oleh CMS:

1. Kekurangan Inventaris Program Perangkat Lunak yang Akurat
   CMS dapat menyediakan inventaris yang akurat tentang semua program perangkat lunak yang digunakan dalam organisasi. Dengan demikian, risiko kehilangan jejak atau kekurangan pemahaman tentang program yang digunakan dapat diminimalkan(Bolatan et al., 2016).
2. Ketidaklengkapan Sejarah Perubahan Program
   CMS secara teratur merekam dan melacak semua perubahan yang dilakukan pada program perangkat lunak. Hal ini membantu dalam mempertahankan sejarah perubahan yang lengkap dan memudahkan untuk mengetahui siapa yang melakukan perubahan serta alasan di baliknya.
3. Modul-Modul Program Perangkat Lunak Terduplikasi
   Dengan CMS, pengembang dapat dengan mudah mengiden- tifikasi apakah ada modul-program yang terduplikasi. Hal ini membantu dalam menghindari pemborosan sumber daya dan memastikan bahwa kode program diorganisir dengan baik.
4. Perubahan Program Perangkat Lunak yang Tidak Sah
   CMS memungkinkan untuk menerapkan kontrol akses dan otorisasi, sehingga hanya pengguna yang berwenang yang dapat membuat perubahan pada program perangkat lunak. Ini

membantu mencegah perubahan yang tidak sah atau tidak diotorisasi.

5. Kekurangan Dokumentasi yang Jelas, Komprehensif, dan Terbaru

   CMS memfasilitasi penyimpanan dan manajemen dokumen, termasuk dokumentasi perangkat lunak. Hal ini memastikan bahwa dokumen-dokumen tersebut tetap teratur, terkini, dan mudah diakses oleh pengguna yang membutuhkannya.

6. Rendahnya Kualitas dan Reliabilitas Perangkat Lunak

   Dengan menggunakan CMS, organisasi dapat menerapkan praktik-praktik pengelolaan perubahan yang disiplin dan terstruktur. Hal ini dapat meningkatkan kualitas dan reliabilitas perangkat lunak secara keseluruhan dengan memastikan bahwa perubahan dipantau, diuji, dan diverifikasi dengan baik sebelum diterapkan ke dalam produksi.

KESIMPULAN

Era digital yang terus berkembang, rekayasa perangkat lunak menjadi salah satu bidang yang paling vital dan dinamis. Dalam buku ini, kami telah menjelajahi berbagai aspek dari rekayasa perangkat lunak, dari pengujian hingga pengembangan, serta pentingnya memastikan kualitas dalam setiap tahap proses.

Pertama-tama, kami membahas pentingnya pengujian perangkat lunak dalam memastikan keandalan, fungsionalitas, dan keamanan produk perangkat lunak. Dengan berbagai metode pengujian seperti pengujian fungsional, pengujian non-fungsional, pengujian regresi, dan lainnya, tim pengembang dapat memastikan bahwa perangkat lunak yang dihasilkan memenuhi standar kualitas yang ditetapkan. Selanjutnya, kami menyoroti tahapan proses pengujian perangkat lunak, mulai dari perencanaan hingga pelaporan hasil. Perencanaan yang cermat, desain kasus uji yang teliti, eksekusi uji yang sistematis, dan pelaporan hasil yang informatif merupakan komponen kunci dari proses yang efektif dan efisien. Tidak hanya itu, kami juga membahas tentang aspek kualitas dalam alat dan teknologi pengujian perangkat lunak. Dengan menggunakan perangkat lunak otomatisasi pengujian, alat manajemen uji, dan alat pelaporan yang berkualitas, tim pengembangan dapat meningkatkan efisiensi, akurasi, dan kualitas pengujian perangkat lunak.

Keseluruhan, rekayasa perangkat lunak merupakan disiplin yang terus berkembang dan penting untuk memastikan keberhasilan produk perangkat lunak di pasar yang kompetitif. Dengan memahami dan menerapkan prinsip-prinsip pengujian dan pengembangan yang tepat, kita dapat menciptakan perangkat lunak yang andal, inovatif, dan memenuhi kebutuhan pengguna dengan baik. Dengan demikian, buku ini menyimpulkan bahwa rekayasa perangkat lunak memainkan peran kunci dalam memastikan kualitas dan keberhasilan produk perangkat lunak di era digital saat ini.

Daftar Pustaka.

Abdoli, S. (2023). A modelling framework to support integrated design of production systems at early design stages. *International Journal on Interactive Design and Manufacturing*, *17*(1), 353–370. https://doi.org/10.1007/S12008-022-00987-X

Ali, N., & Lai, R. (2017). A method of requirements elicitation and analysis for Global Software Development. *Journal of Software: Evolution and Process*, *29*(4). https://doi.org/10.1002/SMR.1830

Aman, M., & Suroso. (2021). Pengembangan Sistem Informasi Wedding Organizer Menggunakan Pendekatan Sistem Berorientasi Objek Pada CV Pesta. *Jurnal Janitra Informatika Dan Sistem Informasi*, *1*(1), 47–60. https://doi.org/10.25008/janitra.v1i1.119

Ameller, D., Farré, C., Franch, X., & Rufian, G. (2016). *A Survey on Software Release Planning Models* (pp. 48–65). https://doi.org/10.1007/978-3-319-49094-6_4

April, A., & Abran, A. (2008). *Software Maintenance Management: Evaluation and Continuous Improvement*. Wiley.

Armano, G., & Marchesi, M. (2000). A rapid development process with UML. *ACM SIGAPP Applied Computing Review*, *8*(1), 4–11. https://doi.org/10.1145/361651.361653

Bolatan, G. I. S., Gozlu, S., Alpkan, L., & Zaim, S. (2016). The Impact of Technology Transfer Performance on Total Quality Management and Quality Performance. *Procedia - Social and Behavioral Sciences*, *235*, 746–755. https://doi.org/10.1016/J.SBSPRO.2016.11.076

Bolung, M., & Tampangela, H. R. K. (2017). Analisa penggunaan metodologi pengembangan perangkat lunak. *Jurnal ELTIKOM: Jurnal Teknik Elektro, Teknologi Informasi Dan Komputer*, *1*(1), 1–10.

Braude, E. J., & Bernstein, M. E. (2016). *Software Engineering: Modern Approaches, Second Edition*. John Wiley.

Bryan, W. (2018). *Mission and System Architecture Design*.

Budi, D. S., & Abijono, H. (2016). Analisis pemilihan penerapan proyek metodologi pengembangan rekayasa perangkat lunak. *Teknika*, *5*(1), 24–31.

Budi, D. S., Siswa, T. A. Y., & Abijono, H. (2015). Analisis Pemilihan Penerapan Proyek Metodologi Pengembangan Rekayasa Perangkat Lunak. *25th International Conference on Computer Theory and Applications, ICCTA 2015 - Proceedings*, *5*(November), 106–111. https://doi.org/10.1109/ICCTA37466.2015.9513455

Che, H., Hajian, M., Ligthart, L. P., & Prasad, R. (2001). Software Radio is Walking into Implementation Stage. *Software Radio*, 81–92. https://doi.org/10.1007/978-1-4471-0343-1_7

Deshpande, S. B., & Mangalwede, S. R. (2018). Analysis of Software Requirements for an M-Learning Framework. *International Journal of Computer Applications*, *181*(5), 17–20. https://doi.org/10.5120/IJCA2018917524

Desikan, S., & Ramesh, G. (2006). *Software Testing: Principles and Practice*. Pearson Education Canada.

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: improving software quality and reducing risk*. Pearson Education.

FarisRosyid, R., & R.Soelistijadi. (2019). Perancangan Aplikasi Pemesanan Makanan Ringan Berbasis Object. *Prosiding SENDI_U 2019*, *2*(1), 277–284.

Gallaba, K., Lamothe, M., & McIntosh, S. (2022). Lessons from eight years of operational data from a continuous integration service: An exploratory case study of circleci. *Proceedings of the 44th International Conference on Software Engineering*, 1330–1342.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

Halawa, S. (2016). Perancangan Aplikasi Pembelajaran Topologi Jaringan Komputer Untuk Sekolah Menengah Kejuruan (Smk) Teknik Komputer Dan Jaringan (Tkj) Dengan Metode Computer Based Instruction. *JURIKOM (Jurnal Riset Komputer)*, *3*(1), 66–71. https://doi.org/10.30865/jurikom.v3i1.53

Haniva, D. T., Ramadhan, J. A., & Suharso, A. (2023). Systematic Literature Review Penggunaan Metodologi Pengembangan Sistem Informasi Waterfall, Agile, dan Hybrid. *(Journal of Information Engineering and Educational Technology*, *7*(1), 36–42.

Hartadi, M. G., Swandi, I. W., & Mudra, I. W. (2020). WARNA DAN PRINSIP DESAIN USER INTERFACE (UI) DALAM APLIKASI SELULER "BUKALOKA." *Jurnal Dimensi DKV: Seni Rupa Dan Desain*, *5*(1), 105–119. https://doi.org/10.25105/JDD.V5I1.6865

Hende, R. Y. L., Setiawan, N. Y., & Mursityo, Y. T. (2018). Perancangan Perbaikan Bisnis Proses Menggunakan Metode Business Process Improvement Pada Layanan Penerbitan Majalah (Studi Pada PT. East Java Liberty Coy). *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *2*(3), 1328–1336.

Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

Izzat, S., & Saleem, N. N. (2023). Software Testing Techniques and Tools: A Review. *Journal of Education and Science*, *32*(2), 31–40. https://doi.org/10.33899/edusj.2023.137480.1305

Jayanti, W. E., & Hendini, A. (2021). Pengembangan Perangkat Lunak Pengujian Kendaraan Bermotor (Tanjidor) Dengan Model Waterfall Pada Dinas Perhubungan. *Jurnal Khatulistiwa Informatika*, *9*(1), 59–67. https://doi.org/https://doi.org/10.31294/jki.v9i1.10099

Jiang, D. (2020). The construction of smart city information system based on the Internet of Things and cloud computing. *Computer Communications*, *150*, 158–166. https://doi.org/10.1016/J.COMCOM.2019.10.035

Kaur, H., & Aggarwal, M. (2023). Specifying Non-Functional Requirements improves Software Vulnerability. *International Conference on Computing Communication and Networking Technologies*. https://doi.org/10.1109/ICCCNT56998.2023.10306987

Klüter, A., Ndiaye, A., & Kirchmann, H. (2000). *Verbmobil From a Software Engineering Point of View: System Design and Software Integration*. 635–658. https://doi.org/10.1007/978-3-662-04230-4_46

Ko, D. G. (1999). Information requirements analysis and multiple knowledge elicitation techniques: experience with the pricing scenario system. *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers*, 235. https://doi.org/10.1109/HICSS.1999.772743

Kole, V., & Sugeng, S. (2021). Implementasi Penjualan Makanan Secara Online dengan Metode DevOps pada Restaurant Zenbu House Jakarta Barat. *Jurnal Sosial Teknologi*, *1*(8), 867–874.

Kun-wu, X. (2013). A Survey of Software Requirements Analysis. *Journal of Hubei University for Nationalities*.

Lamsweerde, A. (2018). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. W. Ross MacDonald School Resource Services Library.

Leloudas, P. (2023). Software Testing Types and Techniques. In *Introduction to Software Testing* (pp. 5–34). Apress. https://doi.org/10.1007/978-1-4842-9514-4_2

Loeliger, J., & McCullough, M. (2012). *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media.

Marbun, R. R., Al Mufied, F., & Fauzi, R. (2022). PERANCANGAN USER INTERFACE/USER EXPERIENCE (UI/UX) WEBSITE HELPMEONG UNTUK SHELTER MENGGUNAKAN METODE GOAL-DIRECTED DESIGN. *JIPI (Jurnal Ilmiah Penelitian Dan Pembelajaran Informatika)*, *7*(4), 1096–1109. https://doi.org/10.29100/JIPI.V7I4.3190

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education.

Meilinda, E., Sabaruddin, R., & Fitriani, D. (2021). Model Prototype Sebagai Metode Pengembangan Perangkat Lunak Pada Sistem Informasi Pengaduan Umum (Studi Kasus: Dinas Perhubungan Provinsi Kalimantan Barat). *Jurnal Khatulistiwa Informatika*, *9*(2).

Mohialden, Y. M., Hussien, N. M., & Hameed, S. A. (2022). Review of Software Testing Methods. *Journal La Multiapp*, *3*(3), 104–112. https://doi.org/10.37899/JOURNALLAMULTIAPP.V3I3.648

Moore, J. M., & Shipman, F. M. (2001). Requirements elicitation using visual and textual information. *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 308–309. https://doi.org/10.1109/ISRE.2001.948598

Morales, J., Botella, F., Rusu, C., & Quiñones, D. (2019). How "friendly" integrated development environments are? *International Conference on Human-Computer Interaction*, 80–91.

Muqsith, M. A., & Sarjoughian, H. S. (2010). A simulator for service-based software system co-design. *International ICST Conference on Simulation Tools and Techniques*. https://doi.org/10.4108/ICST.SIMUTOOLS2010.8735

Mustafa, K. M., Al-Qutaish, R. E., & Muhairat, M. I. (2009). Classification of Software Testing Tools Based on the Software Testing Methods. *2009 Second International Conference on Computer and Electrical Engineering*, *1*, 229–233. https://doi.org/10.1109/ICCEE.2009.9

Nugroho, A. S., & Daniamiseno, A. G. (2022). Pengembangan E-Book Mitigasi Bencana Gunung Api Berbasis Prinsip-Prinsip Desain Pesan Pembelajaran untuk Siswa Sekolah Menengah Pertama. *Jurnal Inovasi Teknologi Pendidikan*, *9*(1), 114–122. https://doi.org/10.21831/jitp.v9i1.21690

Poo, D. C. C. (1992). An Object-Oriented Software Requirements Analysis Method. *International Journal of Software Engineering and Knowledge Engineering*, *02*(01), 145–168. https://doi.org/10.1142/S0218194092000087

Pressman, R. S. (2000). *Software Engineering: A Practitioner's Approach* (5th ed). McGraw Hill.

Pressman, R. S., & Bruce R. Maxim, D. (2014). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.

Pricillia, T. (2021). Perbandingan metode pengembangan perangkat lunak (waterfall, prototype, RAD). *Jurnal Bangkit Indonesia*, *10*(1), 6–12.

Purba, R., Pardosi, I., Darmawan, H., & Sitorus, A. A. (2019). Pengamanan Data Teks Dengan NTRU dan Modulus Function pada Koefisien IHWT Citra Warna. *Jurnal SIFO Mikroskil*, *20*(1), 59–70. https://doi.org/10.55601/jsm.v20i1.649

Ramadhan, R., Wardani, S. D. K., & Amrozi, Y. (2020). Quo Vadis Pengembangan Rekayasa Perangkat Lunak. *Jurnal Teknologi Terapan: G-Tech*, *3*(2), 237–244. https://doi.org/10.33379/gtech.v3i2.427

Rashwan, A. (2012). Semantic Analysis of Functional and Non-Functional Requirements in Software Requirements Specifications. *Canadian Conference on AI*, *7310 LNAI*, 388–391. https://doi.org/10.1007/978-3-642-30353-1_42

Rizky, M., & Sugiarti, Y. (2022). Pengunaan metode scrum dalam pengembangan perangkat lunak: Literature review. *Journal of Computer Science and Engineering (JCSE)*, *3*(1), 41–48.

Rizqi, M., Darnis, R., & Widiana, S. A. (2024). (2024). Implementation of Lean Software Development on Yarn Production Sample Complaint Application. *Sistemasi: Jurnal Sistem Informasi, 13(1), 16-27.*, *13*(1), 16–27.

Saavedra, R., Ballejos, L. C., & Ale, M. (2013). Quality Properties Evaluation for Software Requirements Specifications: An Exploratory Analysis. *Workshop Em Engenharia de Requisitos*.

Saeeda, H., Dong, J., Wang, Y., & Abid, M. A. (2020). A proposed framework for improved software requirements elicitation process in SCRUM: Implementation by a real-life Norway-based IT project. *Journal of Software: Evolution and Process*, *32*(7). https://doi.org/10.1002/SMR.2247

Sari, A. W., Alfiany, N., Hesti, A. N., & Nisa, A. Z. (2022). the Application of a Balanced Scorecard in Sme: a Case Study of Milanzo Kids. *Keunis*, *10*(1), 56. https://doi.org/10.32497/keunis.v10i1.3093

Smart, J. F. (2011). *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. " O'Reilly Media, Inc."

Sommerville, I. (2011). *Software Engineering* (9th ed). Pearson.

Suryn, W. (2014). *Software Quality Engineering* (W. Suryn, Ed.). Wiley. https://doi.org/10.1002/9781118830208

Syafi'i, M. (2021). Perancangan Desain UI/UX Aplikasi Pemesanan Dekorasi Pernikahan pada UKM MNDecoration Menggunakan Metode LEAN UX. *Doctoral Dissertation, Universitas Dinamika*.

Syakti, F. (2019). Metode Pengembangan Perangkat Lunak Berbasis Mobile: a Review. *Jurnal Bina Komputer*, *1*(2), 82–89. https://doi.org/10.33557/binakomputer.v1i2.440

Utami, Y., Nugroho, A., & Wijaya, A. F. (2018). Perencanaan Strategis Sistem Informasi dan Teknologi Informasi pada Dinas Perindustrian dan Tenaga Kerja Kota Salatiga. *Jurnal Teknologi Informasi Dan Ilmu Komputer*, *5*(3), 253–260. https://doi.org/10.25126/jtiik.201853655

Wu, Y. (2023). *Application of Data Encryption Technology in Computer Software Testing* (pp. 62–70). https://doi.org/10.1007/978-981-19-3632-6_9

Yehdeya, E. F., Primasari, C. H., Purnomo Sidhi, T. A., Wibisono, Y. P., Setyohadi, D. B., & Cininta, M. (2023). Analisis User Interface (UI) Dan User Experience (UX) Sudut Elevasi Pemukul Gamelan Metaverse Virtual Reality Menggunakan User Centered Design (UCD). *JIKO (Jurnal Informatika Dan Komputer)*, *7*(1), 137. https://doi.org/10.26798/JIKO.V7I1.757

Yi, Q., & Alytona, K. (2022). *The Development Direction of Computer Software Testing Methods in the Era of Big Data* (pp. 981–987). https://doi.org/10.1007/978-3-031-05237-8_121

Profil Penulis

Ir. Mursalim Tonggiroh, S.Kom., M.Eng.
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer Universitas Yapis Papua

Penulis yang lahir di Kota Jayapura ini adalah Dosen Tetap di Universitas Yapis Papua. Menyelesaikan pendidikan S1 pada Program Studi Teknik Informatika Universitas Islam Indonesia (UII) dan pendidikan S2 pada Program Studi Teknik Elektro Universitas Gadjah Mada (UGM) serta melanjutkan Profesi pada Program Studi Pendidikan Profesi Insinyur Universitas Hasanuddin (UNHAS). Beberapa penelitian yang ditekuni antara lain berhubungan dengan Teknologi Informasi, Rekayasa Perangkat Lunak, dan e-government. Penulis dapat dihubungi di alamat email mursalim.t@gmail.com.

Victor Benny Alexsius Pardosi, S.Kom., M.Sc.
Dosen Fakultas Ilmu Komputer
Universitas Dharma AUB Surakarta
PT Transformasi Data Digital (HostData.id)

Lahir di Aceh Tengah pada tanggal 31 Januari 1991, penulis merupakan seorang praktisi bidang IT, dengan lebih dari sepuluh tahun pengalaman sebagai Web Developer dan System Administrator. Saat buku ini diterbitkan, penulis bekerja sebagai SysAdmin di HostData.id dan Web Developer di PT Transformasi Data Digital, berbagi pengalaman dengan menjadi dosen di Program Studi Sistem Informasi, Fakultas Ilmu Komputer, Universitas Dharma AUB Surakarta. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi di STMIK Dharmapala Riau lalu melanjutkan S2 jurusan Computer Science di Tomsk Polytechnic University, Rusia. Minat penelitiannya terfokus pada bidang Networking, Cyber Security, dan Implementasi Artificial Intelligence.

Basiroh, S.Kom, M.Kom
Dosen Informatika
Fakultas Teknik Universitas Islam Batik

Penulis merupakan Dosen sekaligus Kepala Program Studi pada Program Studi Informatika Fakultas Teknik, Universitas Islam Batik. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika, Universitas Widya Dharma dan melanjutkan S2 pada Universitas Amikom Yogyakarta dengan Konsentrasi Teknik Informatika.

Penulis menekuni reseach atau penelitian bidang Natural Science(Artificial Intelligence dan Image Processing) serta pengabdian masyarakat berbasis IoT, Digital Technologi.

Fifto Nugroho
Dosen Program Studi Sistem Komputer
Fakultas Ilmu Komputer, Universitas Bung Karno

Fifto Nugroho, S.T., M.Kom., lahir di Kota Jakarta, Bulan Oktober 1982. Alumnus dari Universitas Persada Indonesia, Fakultas Teknologi Industri, Program Studi Teknik Informatika, dengan capaian kelulusan sebagai Sarjana Teknik pada Bulan November 2006. Melanjutkan studi strata dua pada Program Magister Ilmu Komputer di Universitas Bunda Mulia dan lulus pada Bulan Agustus 2013 mencapai gelar Magister Komputer. Pada tahun 2014 diangkat menjadi Dosen Tetap Yayasan Pendidikan Soekarno di Universitas Bung Karno dan ditempatkan di Fakultas Ilmu Komputer pada Program Studi Sistem Komputer. Sampai dengan buku ini diterbitkan, aktif ikut serta dalam keanggotaan di lima organisasi nasional profesi di Indonesia, yaitu, Persatuan Insinyur Indonesia (PII), Asosiasi Pendidikan Tinggi Informatika dan Komputer (APTIKOM), Ikatan Ahli Informatika Indonesia (IAII), Asosiasi Internet of Things Indonesia (ASIOTI), dan Asosiasi Big Data dan AI (ABDI).