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:
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):

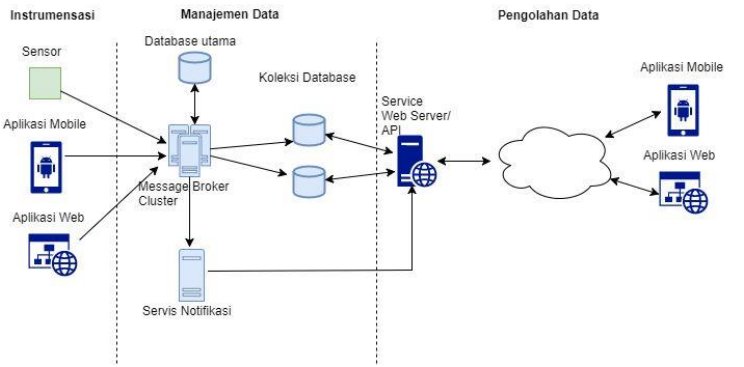
1. Arsitektur Berorientasi Layanan (SOA) adalah pendekatan desain yang menggambarkan sistem perangkat lunak sebagai kumpulan layanan yang digabungkan secara longgar yang masing-masing memiliki fitur tertentu.
2. Alat Simulasi: Perangkat lunak yang digunakan untuk menyimulasikan dan menguji perilaku sistem, 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. 3

METODOLOGI PENGEMBANGAN PERANGKAT LUNAK

3.1 Metodologi Pengembangan Perangkat Lunak

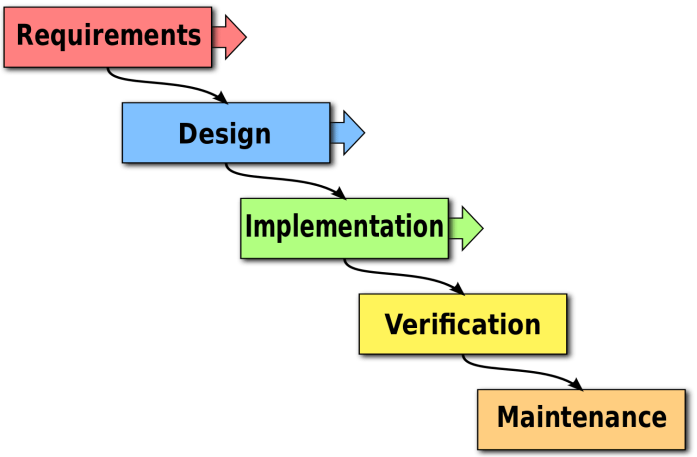
Metodologi pengembangan perangkat lunak atau sistem merujuk pada serangkaian prosedur, prinsip, dan praktik yang digunakan untuk mengatur dan mengendalikan proses pengembangan suatu sistem informasi atau perangkat lunak. Tujuannya adalah untuk memastikan bahwa pengembangan dilakukan dengan efisien, efektif, dan sesuai dengan kebutuhan pengguna serta persyaratan proyek (Budi & Abijono, 2016).

Metodologi ini membantu tim pengembang untuk bekerja secara terkoordinasi, mengurangi risiko kesalahan, meningkatkan prediktabilitas, dan mengoptimalkan penggunaan sumber daya (Pricillia, 2021). Beberapa jenis metodologi pengembangan sistem yang telah dikembangkan meliputi:

3.1.1 Waterfall (Air Terjun)

Waterfall (Air Terjun) merupakan salah satu metodologi pengembangan perangkat lunak yang paling tua dan paling umum digunakan sebelum munculnya pendekatan-pendekatan yang lebih

fleksibel seperti Agile. Metodologi Waterfall mengikuti pendekatan linear dan berurutan, di mana setiap tahapan proses pengembangan harus diselesaikan sebelum memulai tahapan berikutnya. Metodologi ini cocok untuk proyek-proyek yang memiliki persyaratan yang jelas dan tidak mungkin berubah selama siklus pengembangan (Rizky & Sugiarti, 2022).



Gambar 3.1 Metodologi *Waterfall*
(Sumber: https://it.telkomuniversity.ac.id/)

1. Analisis Kebutuhan *(Requirements Analysis)*: identifikasi kebutuhan pengguna dan pemangku kepentingan, serta menganalisis dan mendokumentasikan persyaratan fungsional dan non-fungsional yang harus dipenuhi oleh sistem yang akan dikembangkan.
2. Desain *(Design)*: Setelah persyaratan dikumpulkan, tahap desain dimulai. Ini melibatkan merancang arsitektur sistem dan menguraikan bagaimana sistem akan berfungsi, termasuk desain antarmuka pengguna dan struktur data yang diperlukan.
3. Implementasi *(Implementation)*: kode perangkat lunak sebenarnya ditulis berdasarkan desain yang telah disetujui. Ini adalah tahap di mana seluruh fungsionalitas sistem diterjemahkan menjadi kode komputer yang dapat dieksekusi.

4. Pengujian *(Testing)*: melibatkan pengujian fungsionalitas, kinerja, keamanan, dan kompatibilitas sistem untuk memastikan bahwa perangkat lunak berfungsi sesuai yang diharapkan dan memenuhi standar kualitas yang ditetapkan.
5. Pemeliharaan *(Maintenance)*: Setelah perangkat lunak diluncurkan, tahap pemeliharaan dimulai. Ini melibatkan perbaikan bug, pembaruan perangkat lunak, dan peningkatan fitur sesuai dengan umpan balik pengguna dan perubahan kebutuhan.

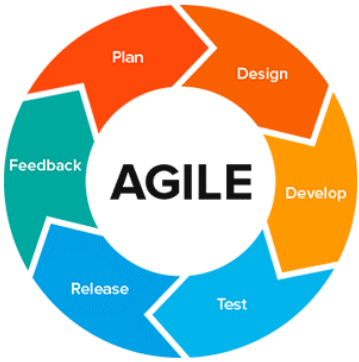
- **Kelebihan Airflow (Waterfall):**
  (a) Sistematis, (b) Mudah dimengerti, (c) Efisiensi: mengurangi waktu dan biaya untuk pengembangan perangkat lunak, (d) Mudah untuk diimplementasikan
- **Kekurangan Airflow (Waterfall):**
  (a) Tidak fleksibel, (b) Tidak cocok untuk proyek yang besar, (c) Tidak cocok untuk proyek yang memerlukan perubahan.

Metodologi Waterfall sering digunakan dalam proyek-proyek dengan persyaratan yang stabil dan tidak berubah sepanjang siklus pengembangan.

3.1.2 Metodologi Agile

Metodologi agiley adalah salah satu pendekatan populer dalam pengembangan perangkat lunak yang menekankan kolaborasi tim, responsivitas terhadap perubahan, pengiriman iteratif, dan fokus pada kepuasan pengguna. Metodologi ini dirancang untuk mengatasi beberapa kelemahan dalam pendekatan pengembangan tradisional yang lebih berorientasi pada perencanaan dan dokumentasi yang terperinci.

Pendekatan adaptif yang menekankan pada kerja kolaboratif, responsif terhadap perubahan, dan pengiriman perangkat lunak secara bertahap dan terus-menerus. Metodologi ini melibatkan pengembangan dalam iterasi pendek yang disebut sprint.



Gambar 3.2 Metode Agile
(Sumber: https://it.telkomuniversity.ac.id/)

Tahapan dalam metodologi Agile tidak selalu secara eksplisit dinamai seperti dalam Waterfall, tetapi terdapat serangkaian aktivitas yang dilakukan secara berulang dalam siklus pengembangan (Haniva et al., 2023). Berikut adalah enam tahapan umum dalam Agile:

1. Plan (Perencanaan): melibatkan identifikasi dan pemetaan kebutuhan pengguna, penentuan prioritas fitur, dan penjadwalan iterasi atau sprint. Tujuan dari tahap ini adalah untuk merencanakan pekerjaan yang akan dilakukan selama iterasi berikutnya dan memastikan pemahaman yang jelas tentang tujuan sprint.
2. Design (Desain): desain lebih fokus pada penentuan arsitektur dan kerangka kerja yang akan digunakan. Tim juga dapat mulai merancang antarmuka pengguna (UI/UX) dan membuat sketsa atau wireframe awal.
3. Develop (Pengembangan): melibatkan pengkodean atau pembangunan perangkat lunak berdasarkan item backlog yang telah diprioritaskan. Pengembang dan tim bekerja untuk mengimplementasikan fitur-fitur yang telah direncanakan dalam iterasi atau sprint.
4. Test (Pengujian): fitur-fitur yang telah dibangun diuji untuk memastikan kualitasnya. Pengujian meliputi pengujian

fungsionalitas, integrasi, kinerja, dan keamanan, sering kali dilakukan secara otomatis atau manual.

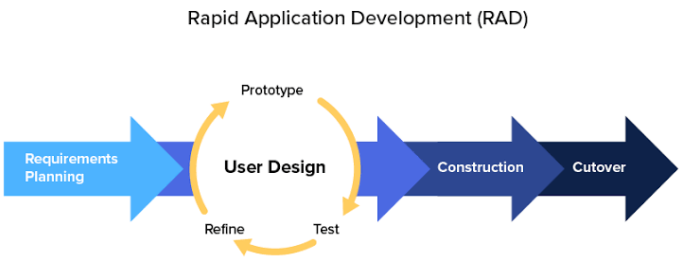
5. Release (Peluncuran): Rilis dapat berupa rilis perangkat lunak secara keseluruhan atau rilis berbasis fitur, tergantung pada kebutuhan dan strategi pengembangan.
6. Feedback (Umpan Balik): Umpan balik ini dapat digunakan untuk memperbaiki atau meningkatkan fitur-fitur yang ada, serta untuk menginformasikan iterasi berikutnya dari siklus pengembangan.

- **Kelebihan Agile:**
  (a) Responsif terhadap Perubahan, (b) Kolaborasi Tim yang Tinggi, (c) Peningkatan Kualitas, (d) Pemenuhan Kebutuhan Pelanggan
- **Kekurangan Agile:**
  (a)Kurangnya Prediksi, (b) Membutuhkan Keterlibatan Pelanggan yang Tinggi, (c) Tidak Cocok untuk Semua Proyek

3.1.3 Metode Rapid Application Development RAD

Rapid Application Development (RAD) adalah suatu pendekatan dalam pengembangan perangkat lunak yang ditekankan pada pembuatan aplikasi secara cepat dengan menggunakan pengulangan dan umpan balik yang berulang-ulang dari para pengguna. Pendekatan ini bertujuan untuk mengatasi tantangan dalam memenuhi permintaan yang cepat dan terus berkembang di dunia teknologi informasi.

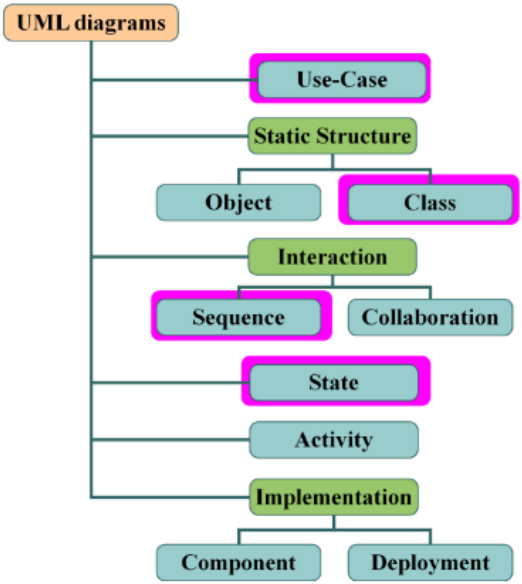
Metode pengembangan *Rapid Application Development* (RAD) adalah pendekatan yang digunakan dalam pengembangan perangkat lunak yang menekankan pada kecepatan dan efisiensi dalam pengembangan aplikasi. RAD dirancang untuk mempercepat proses pengembangan dengan menggunakan teknologi dan metode yang inovatif. Berikut adalah beberapa metode utama dalam pengembangan RAD:



Gambar 3.3 Diagram *Rapid Application Development* (RAD) (Sumber: Kissflow, 2023)

1. Prototyping: Prototipe ini digunakan untuk menguji konsep dan mendapatkan umpan balik dari pengguna sebelum pengembangan aplikasi yang sebenarnya dimulai. Prototyping membantu dalam mengidentifikasi masalah awal dan memperbaiki desain sebelum pengembangan yang lebih lanjut.
2. Iterasi: RAD menggunakan pendekatan iteratif, setiap iterasi melibatkan pengembangan, pengujian, dan penyesuaian. Ini memungkinkan tim untuk membuat perubahan dan penyesuaian yang diperlukan sepanjang proses pengembangan.
3. Penggunaan Teknologi Terkini: RAD memanfaatkan teknologi terkini dan alat pengembangan untuk mempercepat proses pengembangan. Ini termasuk penggunaan IDE (Integrated Development Environment) yang memudahkan pengembangan, pengujian, dan debugging.
4. Pendekatan Berorientasi Objek: RAD sering menggunakan pendekatan berorientasi objek atau UML (*Unified Modeling Language*) untuk mengorganisir kode dan data. Bagaimana UML digunakan dalam model RAD (Armano & Marchesi, 2000), sebagai berikut:
   a) Visualisasi dan Dokumentasi: UML terbagi menjadi tiga kategori utama, yaitu diagram struktur, diagram perilaku, dan diagram interaksi. Setiap kategori ini memiliki berbagai diagram yang merinci arsitektur sistem secara terintegrasi (Haviluddin, 2011). Diagram UML membantu

dalam memahami, merancang, dan mendokumentasikan sistem perangkat lunak dengan cara yang efektif dan konsisten. Berikut diagram UML:



Gambar 3.4 UMl Diagram
(Sumber: https://www.uml.org/)

b) Komunikasi: UML memfasilitasi komunikasi antara berbagai pemangku kepentingan dalam proyek pengembangan perangkat lunak, termasuk pengembang, manajer proyek, stakeholder bisnis, dan pengguna akhir.
c) Perencanaan: UML digunakan sebagai alat perencanaan dalam pengembangan perangkat lunak. Diagram UML, seperti diagram kasus penggunaan dan diagram sekuens, dapat membantu dalam merancang dan melaksanakan pengujian sistem. I
d) Pengujian: Diagram UML, seperti diagram kasus penggunaan dan diagram sekuens, dapat membantu dalam merancang dan melaksanakan pengujian sistem. Ini memungkinkan tim pengujian untuk memahami dan menguji perilaku sistem dengan lebih baik

e) Pemeliharaan: UML juga dapat digunakan dalam proses pemeliharaan perangkat lunak. Dengan mendokumentasikan desain sistem menggunakan UML.

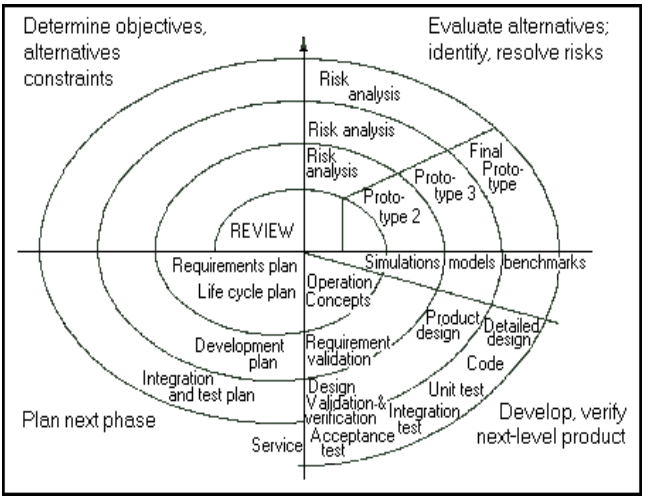
5. Penggunaan Database: RAD sering menggunakan database untuk menyimpan data aplikasi. Penggunaan database memungkinkan pengembang untuk mengelola data dengan lebih efisien dan memudahkan pembaruan dan perubahan data.
6. Pengujian dan Evaluasi: RAD menekankan pada pengujian dan evaluasi yang berkelanjutan. Ini memungkinkan tim untuk mengidentifikasi dan memperbaiki masalah secepat mungkin, sehingga mengurangi risiko kesalahan dan penundaan dalam pengembangan.

- **Kelebihan *Rapid Application Development* (RAD)**
  (a) adaptabilitas terhadap Perubahan, (b) Pengembangan Berdasarkan Kebutuhan Pengguna, (c) Minimalkan Kesalahan: dengan fokus pada umpan balik dan pengujian prototipe, (d) Waktu Pengembangan yang Cepat dan Efisien, (e Mempermudah Proses Integrasi
- **Kekurangan *Rapid Application Development* (RAD)**
  (a) Kebutuhan Tim yang Kuat, (b) Modal Besar, (c) waktu yang Sangat Sedikit, (d) Ketergantungan pada Umpan Balik dan Pengujian (e) Kompleksitas dalam Manajemen Proyek

Metode pengembangan RAD dirancang untuk memungkinkan tim pengembangan untuk merespon dengan cepat terhadap perubahan dan kebutuhan pengguna, serta mempercepat proses pengembangan aplikasi.

3.1.4 Metode Spiral

Metode pengembangan Spiral adalah model pengembangan perangkat lunak yang dirancang untuk proyek-proyek besar yang memiliki risiko tinggi. Model ini menggabungkan aspek dari model pengembangan perangkat lunak incremental, waterfall, dan prototyping evolusi. Model spiral menekankan pada Analisa resiko setiap tahapannya. Berikut tahapan-tahapan metode spiral



Gambar 3.5 Metode Spiral
(Sumber: https://sis.binus.ac.id/)

1. Tahap Koordinasi: komunikasi antara pihak-pihak terlibat dalam pengembangan perangkat lunak, seperti analis sistem, dengan pengguna.
2. Tahap Perencanaan: mencakup estimasi biaya, batas waktu, penjadwalan, identifikasi lingkungan kerja, dan sumber-sumber informasi untuk melakukan iterasi. Hasil dari tahap ini adalah dokumen spesifikasi kebutuhan sistem dan bisnis.
3. Tahap Analisis Risiko: mengidentifikasi risiko yang mungkin timbul dan mencari solusi alternatif secara teknis dan manajerial. Pada tahap ini, strategi mitigasi juga direncanakan dan dilaksanakan untuk mengurangi risiko bencana.
4. Tahap Rekayasa: beberapa kegiatan dilakukan, seperti pengujian, pemrograman, pengembangan perangkat lunak, instalasi perangkat lunak, pembuatan prototipe, perancangan dokumen, penyusunan ringkasan pengujian perangkat lunak, serta pembuatan laporan tentang kekurangan perangkat lunak untuk segera diperbaiki.
5. Tahap Evaluasi: analis sistem memperoleh masukan dan tanggapan dari pengguna untuk mengevaluasi produk yang diuji. Mereka memastikan bahwa produk sesuai dengan

kebutuhan awal yang telah dibicarakan dengan pengguna. Selain itu, mereka juga memantau risiko potensial seperti biaya yang melampaui anggaran yang telah ditetapkan.

- **Kelebihan Model Spiral:**
  (a) Adaptabilitas terhadap Perubahan, (b) Penggunaan Prototype (c) Manajemen Risiko yang Efektif (d) Pemantauan dan Kontrol yang Efektif, (e) Fokus pada Kontrol Dokumentasi, (f) Penekanan pada Persetujuan Klien
- **Kekurangan Model Spiral**
  (a) Manajemen yang Lebih Kompleks, (b) Biaya yang Tinggi, (c) Tidak Cocok untuk Proyek Kecil atau Berisiko Rendah

Model Spiral dirancang untuk menangani proyek-proyek yang kompleks dengan risiko tinggi, dengan fokus pada perencanaan yang baik, dokumentasi yang rinci, dan pengujian yang sistematis. Ini membuatnya cocok untuk proyek-proyek besar di mana manajemen risiko dan kualitas adalah prioritas utama.

3.2 Studi Case Metode Pengembangan Perangkat Lunak

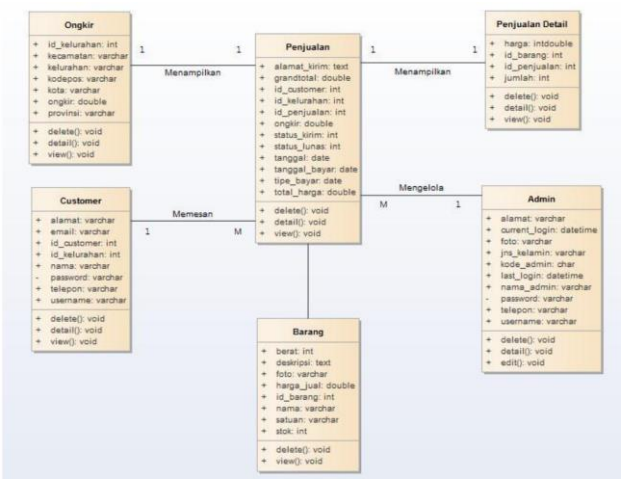
3.2.1 Perancangan Aplikasi Pemesanan Makanan Ringan Berbasis Object Oriented dengan Metode Waterfall (FarisRosyid & R.Soelistijadi, 2019)

Metode Waterfall:

1. Requirements analysis and definition: a) konsumen dapat melihat produk dan mendapatkan informasi tentang stok, b) konsumen harus bisa login/registrasi terlebih dahulu c) masuk ke menu keranjang untuk melanjutkan pembayaran d) melakukan konfirmasi pembayaran
2. System and software design: menggunakan tool-tool grafis dari UML.

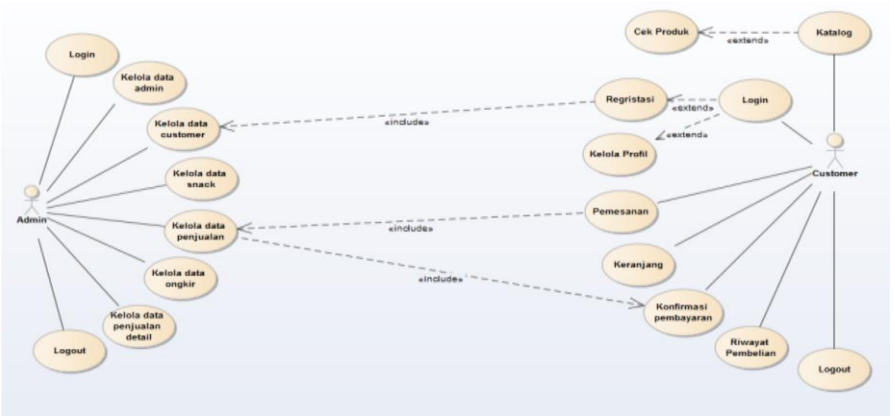
a. Class Diagram

1) Class Admin dengan atribut : kode_admin*, nama_admin, jns_kelamin, alamat, telepon, username, password, current_login, last_login, foto.
2) Class Customer dengan atribut :id_customer*, nama, username, password, email, telepon, alamat, id_kelurahan
3. Class Barang dengan atribut :id_barang*, nama, satuan, harga_jual, deskripsi, foto, stok, berat.
4. Class Penjualan dengan atribut :id_penjualan*, tanggal, id_customer, status_lunas, total_harga, ongkir, grandtotal, tipe_bayar, tanggal_bayar, status_kirim, alamat_kirim, id_kelurahan
5. Class Penjualan Detail dengan atribut : id_penjualan, id_barang, jumlah, harga
6. Class Ongkir dengan atribut :id_kelurahan, provinsi, kabupaten, kecamatan, kelurahan, kodepos, ongkir.



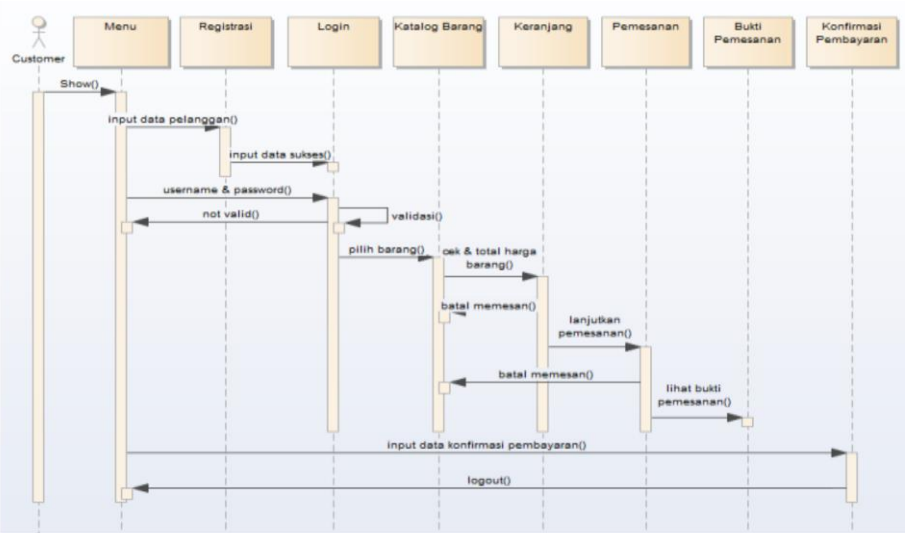
Gambar 3.6 Class Diagram pada Kasus
Sumber: (FarisRosyid & R.Soelistijadi, 2019)

b. Usecase Diagram



Gambar 3.7 Usecase Diagram pada Kasus
Sumber: (FarisRosyid & R.Soelistijadi, 2019)

c. Sequence Diagram



Gambar 3.9 *Sequence Diagram*
Sumber: (FarisRosyid & R.Soelistijadi, 2019)

3. Implementation and unit testing: diimplementasikan dalam bentuk tampilan-tampilan. disesuaikan dengan kebutuhan dan *user oriented*

   a) Tampilan Home, b) Menu Login, c) Menu Register, d) Katalog, e) Detail Produk, f) Keranjang, g) Kategori, h) Riwayat pembelian i) Konfirmasi pembayaran, j) pesanan terkirim
4. Integration and system testing Unit-unit individu program digabung dan diuji. Penggabungan dan Testing Sistem dilakukan dengan metode pengujian black box. Setelah pengujian, perangkat lunak dapat dikirimkan ke customer

Data Masukan	Yang diharapkan	Yang Didapatkan	Kesimpulan
Usernamedan password sudah diisi dengan benar	Akan masuk kedalam halaman web admin	Sistem menerima akses login	Berhasil
Username dan password tidak diisi.	Akan menampilkan pesan bahwa login gagal	Sistem menolak akses login	Berhasil
Admin lupa password akun dan tidak bisa login.	Akan menampilkan pesan bahwa login gagal	Sistem menolak akses login	Berhasil
Menambah data makanan untuk stok yang baru	Sistem akan menyimpan data makanan yang ditambahkan	Sistem sukses memproses	Berhasil
Menghapus data makanan yang stok nya telah habis	Sistem akan melakukan proses penghapusan data	Sistem sukses memproses	Berhasil
Mengganti status bayar dari belum lunas menjadi sudah lunas	Sistem akan mengupdate status bayar	Sistem sukses memproses	Berhasil
Mengganti status kirim dari belum dikirim menjadi sudah dikirim	Sistem akan mengupdate status kirim	Sistem sukses memproses	Berhasil
Mengisi seluruh data yang diperlukan sistem	Sistem akan menerima dan customer akan melakukan login	Sistem suskses memproses	Berhasil
Tidak mengisi seluruh data yang diperlukan sistem	Akan menampilkan pesan bahwa regristasi gagal	Sistem menolak akses	Berhasil
Username dan password sudah diisi dengan benar	Akan masuk kedalam halaman utama.	Sistem menerima akses login	Berhasil
Username dan password tidak diisi.	Akan menampilkan pesan bahwa login gagal	Sistem menolak akses login	Berhasil
Costumer lupa password akun dan tidak bisa login.	Akan menampilkan pesan bahwa login gagal	Sistem menolak akses login	Berhasil
Customer memilih barang yang akan di beli	Akan masuk kedalam halaman keranjang.	Sistem sukses memproses	Berhasil
Customer memilih barang lalu memberitahu jumlah barang	Akan masuk kedalam halaman keranjang	Sistem sukses memproses dan harga bertambah	Berhasil
Mengisi no order, tanggal bayar, jumlah bayar, bukti transfer	Proses konfirmasi pembayaran akan sukes	Sistem sukses memproses	Berhasil
Mengisi no order, tanggal bayar, jumlah bayar, namun tidak memberi bukti transfer	Proses konfirmasi pembayaran gagal	Sistem menolak akses	Berhasil

Gambar 3.9 *Implemetation and Testing* pada Kasus
Sumber: (FarisRosyid & R.Soelistijadi, 2019)

5. Operation and maintenance: pemasangan program yang sudah jadi dan digunakan secara nyata.

BAB. 4

KEBUTUHAN PERANGKAT LUNAK

4.1. Definisi Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak merujuk pada spesifikasi fungsional dan non-fungsional yang harus dipenuhi oleh sebuah sistem perangkat lunak agar dapat memenuhi kebutuhan pengguna, pemangku kepentingan, dan tujuan bisnis yang ditetapkan (Sommerville, 2011). Dalam konteks rekayasa perangkat lunak, definisi yang jelas dan komprehensif mengenai kebutuhan perangkat lunak menjadi landasan utama dalam seluruh siklus pengembangan perangkat lunak.

Kebutuhan perangkat lunak terbagi menjadi dua kategori utama: kebutuhan fungsional dan non-fungsional. Kebutuhan fungsional mencakup spesifikasi tentang apa yang seharusnya dilakukan oleh sistem, seperti fitur-fitur yang harus dimiliki atau perilaku yang diharapkan dari perangkat lunak. Sedangkan kebutuhan non-fungsional meliputi aspek-aspek tambahan seperti kinerja, keamanan, dan keandalan sistem.

Definisi yang jelas tentang kebutuhan perangkat lunak memungkinkan para pengembang untuk memahami secara menyeluruh tentang apa yang harus dicapai oleh sistem yang

sedang dikembangkan. Selain itu, definisi yang tepat juga membantu dalam memfasilitasi komunikasi yang efektif antara tim pengembang, pemangku kepentingan, dan pengguna akhir.

Dalam prakteknya, proses pengembangan perangkat lunak sering dimulai dengan fase identifikasi dan dokumentasi kebutuhan perangkat lunak. Pada tahap ini, berbagai teknik dan metode digunakan untuk menggali, menganalisis, dan mendokumentasikan kebutuhan perangkat lunak secara sistematis.

Memahami dan mendefinisikan kebutuhan perangkat lunak dengan baik bertujuan supaya pengembang dapat mengurangi risiko kesalahan dalam pengembangan, meningkatkan kualitas produk, dan memastikan bahwa perangkat lunak yang dihasilkan dapat memenuhi ekspektasi pengguna dan tujuan bisnis yang diinginkan.

4.2. Jenis-jenis Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak dapat dibagi menjadi beberapa kategori utama, yang masing-masing memberikan pandangan yang berbeda terhadap spesifikasi dan karakteristik yang harus dimiliki oleh sistem perangkat lunak yang dikembangkan(Pressman, 2000).

4.2.1. Kebutuhan Fungsional

Kebutuhan fungsional adalah aspek krusial dalam rekayasa perangkat lunak yang menggambarkan apa yang sistem seharusnya lakukan. Ini merinci fungsi-fungsi yang harus dilakukan oleh perangkat lunak, serta perilaku yang diantisipasi dari sistem tersebut. Kebutuhan fungsional menetapkan kerangka kerja dasar yang menjadi dasar bagi pengembang untuk merancang, mengimplementasikan, dan menguji fungsionalitas sistem.

Kebutuhan fungsional mencakup beragam fitur yang diinginkan oleh pengguna atau *stakeholder*, seperti kemampuan untuk melakukan login pengguna, mencari dan menampilkan data, mengelola proses transaksi, dan banyak lagi. Fitur-fitur ini membentuk landasan sistem yang memungkinkan pengguna untuk

mencapai tujuan mereka secara efektif melalui penggunaan perangkat lunak.

Dalam proses identifikasi kebutuhan fungsional, pengembang perangkat lunak berinteraksi dengan pemangku kepentingan untuk memahami secara mendalam kebutuhan fungsional yang spesifik. Hal ini melibatkan analisis kebutuhan secara rinci, yang dapat dilakukan melalui teknik seperti wawancara, sesi brainstorming, atau penggunaan studi kasus. Dengan memahami kebutuhan fungsional dengan baik, tim pengembang dapat menggambarkan dengan jelas apa yang diharapkan dari sistem yang sedang dikembangkan.

Kebutuhan fungsional sering diungkapkan melalui dokumen spesifikasi kebutuhan fungsional (*Functional Requirements Specification*), yang mencantumkan setiap fitur yang diinginkan bersama dengan deskripsi fungsionalnya. Dokumen ini menjadi panduan utama dalam pengembangan perangkat lunak, memandu tim pengembang selama seluruh siklus pengembangan, dari perencanaan hingga implementasi dan pengujian.

4.2.2. Kebutuhan Non-Fungsional

Kebutuhan non-fungsional merupakan dimensi kritis dalam rekayasa perangkat lunak yang mengarahkan aspek-aspek tambahan yang memengaruhi performa, kinerja, dan kualitas keseluruhan dari sistem perangkat lunak. Sementara kebutuhan fungsional menggambarkan apa yang sistem harus lakukan, kebutuhan non-fungsional memperinci bagaimana sistem harus berperilaku dalam kondisi tertentu. Aspek-aspek ini mencakup berbagai persyaratan tambahan yang tidak berkaitan langsung dengan fungsi-fungsi inti sistem, tetapi penting untuk memastikan bahwa sistem beroperasi secara efisien, aman, dan memuaskan pengguna.

Salah satu contoh kebutuhan non-fungsional adalah waktu respons, yang mengacu pada waktu yang dibutuhkan oleh sistem untuk memberikan respons terhadap permintaan pengguna. Kebutuhan ini sering kali menjadi kritis dalam aplikasi yang

membutuhkan kinerja yang cepat, seperti sistem perbankan *online* atau platform *e-commerce*. Selain itu, keandalan sistem adalah aspek penting yang menentukan seberapa baik sistem dapat berfungsi tanpa mengalami gangguan atau kegagalan, yang sering kali diukur dalam waktu operasional tanpa gangguan.

Aspek keamanan juga merupakan bagian integral dari kebutuhan non-fungsional, yang menetapkan standar dan mekanisme yang diperlukan untuk melindungi data dan sistem dari ancaman luar dan penyalahgunaan. Ini mencakup persyaratan terkait dengan enkripsi data, autentikasi pengguna, dan kontrol akses yang ketat. Selain itu, skalabilitas adalah kebutuhan non-fungsional lainnya yang menentukan kemampuan sistem untuk menangani peningkatan beban kerja tanpa mengalami penurunan kinerja atau kegagalan.

Selain aspek-aspek tersebut, kegunaan (*usability*) dan aksesibilitas juga merupakan aspek penting dari kebutuhan non-fungsional. Kebutuhan kegunaan menentukan seberapa mudah sistem dapat digunakan oleh pengguna akhir, sementara kebutuhan aksesibilitas menetapkan standar untuk memastikan bahwa sistem dapat diakses oleh semua pengguna, termasuk mereka yang memiliki keterbatasan fisik atau sensorik.

4.2.3. Kebutuhan *Domain*

Kebutuhan *domain* menandakan kategori kebutuhan yang secara khusus berasal dari konteks atau lingkungan di mana perangkat lunak akan diimplementasikan. Ini melibatkan persyaratan yang disesuaikan dengan kebutuhan unik dari suatu industri, bisnis, atau bidang spesifik. Dalam dunia rekayasa perangkat lunak, memahami dan memenuhi kebutuhan domain menjadi krusial karena membantu memastikan bahwa perangkat lunak yang dikembangkan akan efektif beroperasi dan menyokong kebutuhan yang muncul dalam lingkungan yang spesifik.

Misalnya, dalam industri medis, perangkat lunak yang digunakan untuk manajemen catatan pasien akan memiliki kebutuhan *domain* yang berbeda dengan perangkat lunak yang

digunakan untuk sistem manajemen keuangan. Kebutuhan perangkat lunak medis mungkin termasuk kepatuhan terhadap standar pengobatan, keamanan data pasien, dan pelaporan klinis yang akurat. Di sisi lain, perangkat lunak keuangan mungkin memerlukan integrasi dengan sistem perbankan dan kepatuhan terhadap regulasi keuangan yang berlaku.

Dalam konteks pendidikan, perangkat lunak yang digunakan untuk pembelajaran jarak jauh atau manajemen akademik akan memiliki kebutuhan domain yang berbeda dengan perangkat lunak yang digunakan dalam industri lain. Kebutuhan perangkat lunak pendidikan mungkin mencakup integrasi dengan sistem manajemen pembelajaran, fitur-fitur untuk mengevaluasi kemajuan siswa, dan alat kolaborasi antara siswa dan pengajar.

4.2.4. Kebutuhan Pengguna

Kebutuhan pengguna merupakan elemen kunci dalam rekayasa perangkat lunak yang berfokus pada kebutuhan, harapan, dan preferensi yang diajukan oleh pengguna akhir atau pemangku kepentingan yang akan menggunakan sistem perangkat lunak. Memahami dengan jelas kebutuhan pengguna adalah langkah penting dalam merancang sistem yang dapat memenuhi ekspektasi pengguna, meningkatkan kepuasan pengguna, dan mencapai tujuan bisnis yang diinginkan.

Kebutuhan pengguna mencakup berbagai aspek, baik fungsional maupun non-fungsional, yang dinyatakan atau diungkapkan oleh pengguna. Contoh dari kebutuhan fungsional pengguna bisa mencakup fitur-fitur tertentu yang diinginkan oleh pengguna, seperti antarmuka pengguna yang ramah pengguna (*user-friendly interface*), kemampuan untuk menyimpan dan membagikan data, atau algoritma pencarian yang efisien. Sementara itu, kebutuhan non-fungsional pengguna mungkin termasuk preferensi terhadap kecepatan respons aplikasi, tampilan yang menarik, keamanan data yang tinggi, atau dukungan aksesibilitas bagi pengguna dengan kebutuhan khusus.

Untuk mengidentifikasi kebutuhan pengguna dengan tepat, pengembang perangkat lunak sering menggunakan berbagai teknik seperti wawancara dengan pengguna, pengamatan langsung, survei, atau analisis dokumen kebutuhan yang telah ada. Dengan memahami kebutuhan pengguna secara menyeluruh, pengembang dapat merancang sistem yang dapat memenuhi kebutuhan dan ekspektasi pengguna dengan baik.

Kebutuhan pengguna juga dapat berubah seiring waktu, karena perkembangan teknologi, perubahan kebutuhan bisnis, atau perubahan preferensi pengguna. Oleh karena itu, penting untuk terus menganalisis, memantau, dan memperbarui kebutuhan pengguna selama siklus hidup pengembangan perangkat lunak untuk memastikan bahwa sistem tetap relevan dan memenuhi kebutuhan pengguna dengan baik.

4.3. Proses Identifikasi Kebutuhan Perangkat Lunak

Proses identifikasi kebutuhan perangkat lunak merupakan tahap awal yang kritis dalam pengembangan perangkat lunak di mana kebutuhan fungsional dan non-fungsional dari sistem yang akan dikembangkan diidentifikasi, dianalisis, dan didokumentasikan secara sistematis (Ameller et al., 2016). Proses ini melibatkan berbagai langkah dan teknik untuk memastikan bahwa kebutuhan yang terkumpul mencerminkan kebutuhan sebenarnya dari pengguna, pemangku kepentingan, dan tujuan bisnis yang ditetapkan. Berikut adalah Langkah untuk proses identifikasi kebutuhan perangkat lunak, yaitu:

4.3.1 Eksplorasi Kebutuhan

Langkah pertama dalam proses identifikasi kebutuhan perangkat lunak adalah eksplorasi kebutuhan, di mana tim pengembang melakukan interaksi dengan pengguna potensial dan pemangku kepentingan lainnya untuk memahami konteks penggunaan sistem dan mengeksplorasi berbagai kebutuhan yang mungkin ada. Ini dapat melibatkan wawancara, diskusi kelompok, atau teknik observasi untuk memahami secara mendalam tantangan dan kebutuhan yang dihadapi oleh pengguna.

4.3.2 Analisis Stakeholder

Setelah kebutuhan awal teridentifikasi, langkah berikutnya adalah analisis *stakeholder*, di mana pemangku kepentingan yang berbeda yang terlibat dalam pengembangan sistem diidentifikasi dan kebutuhan mereka dipahami secara lebih rinci. Hal ini membantu dalam memastikan bahwa semua kepentingan dan persyaratan yang relevan dipertimbangkan dalam proses pengembangan perangkat lunak.

4.3.3 Dokumentasi Kebutuhan

Setelah kebutuhan perangkat lunak teridentifikasi dengan baik, langkah terakhir dalam proses ini adalah mendokumentasikan kebutuhan dengan jelas dan terstruktur. Dokumen kebutuhan perangkat lunak yang baik harus mencakup deskripsi yang rinci tentang setiap kebutuhan, termasuk deskripsi fungsionalitas, kriteria penerimaan, prioritas, dan aspek-aspek non-fungsional yang terkait.

4.4 Alat dan Metode untuk Pengelolaan Kebutuhan

Pengelolaan kebutuhan perangkat lunak adalah aspek penting dalam pengembangan perangkat lunak yang memerlukan pendekatan sistematis untuk mengidentifikasi, mendokumentasikan, memvalidasi, dan melacak perubahan terhadap kebutuhan selama seluruh siklus pengembangan (Sommerville, 2011). Terdapat berbagai alat dan metode yang tersedia untuk membantu tim pengembang dalam pengelolaan kebutuhan perangkat lunak secara efektif.

4.4.1. Model Pengembangan Kebutuhan

Model pengembangan kebutuhan adalah landasan metodologis yang mengatur proses identifikasi, manajemen, dan evolusi kebutuhan perangkat lunak selama siklus hidup pengembangan. Setiap model memberikan pandangan unik tentang bagaimana kebutuhan perangkat lunak harus diidentifikasi, diprioritaskan, dan disesuaikan dengan perubahan yang terjadi dalam lingkungan proyek.

Salah satu contoh model pengembangan kebutuhan yang umum adalah model *waterfall*, yang menggambarkan proses pengembangan sebagai serangkaian tahapan yang berurutan, mulai dari analisis kebutuhan hingga pengujian dan pemeliharaan. Model ini menekankan pada pemetaan kebutuhan secara komprehensif pada tahap awal proyek, sebelum melanjutkan ke tahapan selanjutnya. Namun, kekurangan utama dari model ini adalah kurangnya fleksibilitas dalam menanggapi perubahan kebutuhan yang mungkin terjadi selama siklus pengembangan.

Selain model *waterfall*, terdapat juga model spiral yang menekankan pada iterasi dan peningkatan berkelanjutan. Model ini memungkinkan pengembang untuk menggali kebutuhan secara mendalam melalui serangkaian siklus iteratif, sambil terus menambahkan, mengubah, atau menghapus kebutuhan sebagai-mana diperlukan. Pendekatan ini memberikan fleksibilitas yang lebih besar dalam menanggapi perubahan kebutuhan dan menyesuaikan strategi pengembangan sesuai dengan temuan baru selama iterasi.

Sementara itu, model *agile* menjadi pendekatan yang semakin populer dalam pengembangan perangkat lunak yang menekankan pada fleksibilitas, adaptabilitas, dan kolaborasi tim. Dalam model ini, kebutuhan dikembangkan secara inkremental melalui serangkaian sprint atau iterasi singkat, dengan fokus pada memberikan nilai tambah kepada pengguna pada setiap langkahnya. Pendekatan ini memungkinkan pengembang untuk terus memperbaiki dan menyesuaikan kebutuhan seiring evolusi produk dan feedback dari pengguna.

4.4.2. Perangkat Lunak Pengelola Kebutuhan

Perangkat lunak pengelola kebutuhan menjadi inti dari proses pengelolaan kebutuhan perangkat lunak yang efektif. Dirancang khusus untuk menyediakan kerangka kerja yang terstruktur dan terorganisir, perangkat lunak ini memfasilitasi pengumpulan, dokumentasi, pengelolaan, dan pelacakan perubahan terhadap kebutuhan selama seluruh siklus hidup pengembangan perangkat lunak. Salah satu fitur utama dari

perangkat lunak pengelola kebutuhan adalah kemampuannya untuk mencatat dan mendokumentasikan setiap kebutuhan dengan rinci. Ini termasuk deskripsi lengkap tentang setiap kebutuhan fungsional dan non-fungsional, serta atribut-atribut tambahan seperti prioritas, sumber, dan status. Dengan adanya pencatatan yang terperinci, tim pengembang dapat memahami dengan jelas setiap aspek dari kebutuhan dan bagaimana itu berkontribusi pada tujuan keseluruhan proyek.

Selain itu, perangkat lunak pengelola kebutuhan juga memungkinkan pelacakan perubahan terhadap kebutuhan. Ini memungkinkan pengembang untuk melacak setiap perubahan yang terjadi pada kebutuhan selama proses pengembangan, termasuk revisi, penambahan, atau penghapusan kebutuhan. Pelacakan ini penting untuk memastikan bahwa setiap perubahan terhadap kebutuhan dipertimbangkan dengan baik dan diimplementasikan dengan benar. Kolaborasi tim juga menjadi fokus penting dari perangkat lunak pengelola kebutuhan. Dengan memfasilitasi komunikasi dan kerja sama antara anggota tim yang terlibat dalam pengembangan perangkat lunak, perangkat lunak ini memungkinkan pemangku kepentingan untuk berbagi informasi, memberikan umpan balik, dan bekerja sama dalam memahami dan memenuhi kebutuhan yang ada.

4.4.3. Metode Validasi Kebutuhan

Validasi kebutuhan adalah tahap penting dalam proses rekayasa kebutuhan yang bertujuan untuk memastikan bahwa kebutuhan yang diidentifikasi memenuhi standar kualitas yang diharapkan dan sesuai dengan kebutuhan sebenarnya dari sistem yang akan dikembangkan. Metode validasi kebutuhan melibatkan serangkaian teknik dan pendekatan yang digunakan untuk memeriksa, mengevaluasi, dan mengonfirmasi keakuratan, konsistensi, dan kelengkapan dari kebutuhan yang telah teridentifikasi.

Salah satu metode validasi kebutuhan yang umum adalah teknik pengujian kebutuhan, di mana kebutuhan dipertimbangkan sebagai tes yang harus dilalui oleh sistem. Teknik ini melibatkan

pembuatan skenario pengujian yang mencakup semua kebutuhan yang telah diidentifikasi, dan sistem kemudian diuji untuk memastikan bahwa setiap kebutuhan terpenuhi dengan benar. Hasil dari pengujian ini digunakan untuk memvalidasi keakuratan dan keberhasilan dari setiap kebutuhan.

Selain teknik pengujian, analisis prototipe juga sering digunakan sebagai metode validasi kebutuhan. Dalam hal ini, prototipe sistem atau fitur tertentu dikembangkan dan diperlihatkan kepada pemangku kepentingan untuk mendapatkan umpan balik tentang kelayakan dan keefektifan kebutuhan yang diusulkan. Analisis prototipe memungkinkan pengembang untuk mengevaluasi respons pengguna terhadap desain sistem dan mengidentifikasi potensi masalah atau kebutuhan tambahan yang mungkin timbul.

Selain itu, tinjauan dan verifikasi oleh pemangku kepentingan yang relevan juga merupakan bagian integral dari metode validasi kebutuhan. Proses ini melibatkan penyelidikan menyeluruh atas kebutuhan yang diajukan oleh tim pengembang, serta tinjauan dan persetujuan oleh pemangku kepentingan terkait, seperti pengguna akhir, manajemen proyek, atau tim pemasaran. Dengan melibatkan pemangku kepentingan secara aktif dalam proses validasi, tim pengembang dapat memastikan bahwa kebutuhan yang diajukan mencerminkan kebutuhan yang sebenarnya dari sistem yang akan dikembangkan.

BAB. 5

TOOLS REKAYASA PERANGKAT LUNAK

5.1 Alat Pengembangan *(Development Tools)*

Alat pengembangan khususnya *Integrated Development Environments* (IDEs) dan *Text Editors*, memainkan peran penting dalam siklus hidup pengembangan perangkat lunak. Keduanya menawarkan lingkungan yang efisien dan fleksibel untuk pengkodean, dengan masing-masing memiliki kelebihan tersendiri yang sesuai dengan kebutuhan dan preferensi pengembang yang berbeda.

5.1.1 Integrated Development Environments (IDEs)

Integrated Development Environment menyediakan lingkungan pengembangan terpadu yang komprehensif termasuk editor kode, kompiler, debugger, dan simulator. *Integrated Development Environment* mendukung pengembangan end to end dari menulis kode hingga debugging dan pengujian (Morales et al., 2019). Contoh:

- Visual Studio
  Populer di kalangan pengembang yang menggunakan bahasa pemrograman seperti C#, VB.NET, dan C++. Menawarkan dukungan luas untuk pengembangan aplikasi dekstop, web, dan mobile.

- IntelliJ IDEA
  Dikenal luas di kalangan pengembang Java, kotlin, dan berbagai bahasa pemrograman lainnya. Menyediakan fitur seperti analisis kode, *refactoring*, dan integrasi sistem *version control* yang kuat.
- Eclipse
  Sebuah IDE yang serbaguna dan dapat diperluas, populer di kalangan pengembang Java. Mendukung berbagai bahasa pemrograman lain melalui plugin.

5.1.2 Text Editors

Text editors menawarkan lingkungan yang lebih ringan dna cepat untuk menulis dan mengedit kode. Walaupun tidak sekomprehensif IDE, banyak *text editor* yang menawarkan fitur seperti *syntax highlighting*, pengecekan *syntax*, dan dukungan sederhana untuk *version control*. Contoh:

- *Sublime Text*
  Dikenal dengan antarmukanya yang bersih dan kecepatannya. Menyediakan fitur pencarian yang kuat, banyak shortcuts keyboard, dan kemampuan untuk menyesuaikan melalui plugin.
- Atom
  Dikembangkan oleh GitHub, Atom adalah *text editor open source* yang dapat diperluas dan sangat disesuaikan. Mendukung integrasi langsung dengan Git dan GitHub.
- *Visual Studio Code* (VS Code)
  Menawarkan kombinasi kecepatan text editor dengan beberapa kemampuan IDE, seperti debugging dan git integration. Sangat populer dan mendukung berbagai bahasa pemrograman melalui ekstensi.

Kedua alat ini merupakan pilihan utama bagi pengembang perangkat lunak dan seringkali dipilih berdasarkan kebutuhan proyek, preferensi pribadi atau lingkungan pengembangan. Penggunaan IDE umumnya lebih disukai untuk proek yang kompleks yang memerlukan banyak debugging dan pengujian,

sementara text editor lebih disukai untuk pengeditan cepat, kobfigurasi atau proyek yang lebih ringan.

5.2 Alat Manajemen Kode Sumber (*Source Control Management Tools*)

Berikut tabel yang menjelaskan tentang alat manajemen kode sumber, khususnya sistem kontrol versi yang merupakan komponen penting dalam manajemen proyek pengembangan perangkat lunak untuk mengelola perubahan pada dokumen, kode, dan berbagai jenis file.

Tabel 5.1 Alat Manajemen Kode Sumber *(Source Control Management Tools)*

Alat	Deskripsi	Keunggulan	Kekurangan
Git	Sistem kontrol versi terdistribusi yang memungkinkan pengembang bekerja bersama dengan efisien pada proyek yang sama.	- Fleksibel dan kuat - Mendukung *workflow* terdistribusi - Banyak digunakan dna memiliki komunitas yang besar	- Kurva belajar yang cukup tinggi bagi pemula - Manajemen besar *binary file* dapat menjadi rumit
Subversion (SVN)	Sistem kontrol versi terpusat yang menyimpan versi berbeda dari file dalam sebuah reporitori pusat.	- Mudah digunakan bagi pemula - Manajemen akses yang baik - Integrasi dengan berbagai *tools*	- Tidak seefisien Git dalam manajemen *branch* dan *merge* - Kurang mendukung *workflow* terdistribusi

Mercurial	Sistem kontrol versi terdistribusi yang mirip dengan Git, dirancang untuk mudah digunakan.	- Mudah digunakan dan dipelajari - Performa yang baik - Fleksibel dalam manajemen proyek	- Tidak sepopuler Git - Komunitas yang lebih kecil dibandingkan Git

Sistem kontrol versi memainkan peran kunci dalam pengembangan perangkat lunak, memungkinkan tim untuk bekerja secara kolaboratif dengan mengelola perubahan pada kode sumber dan menjaga integritas proyek. Pilihan antara Git, SVN, dan Mercurial seringkali bergantung pada kebutuhan spesifik proyek, preferensi tim pengembangan, dan lingkungan kerja yang ada (Kole & Sugeng, 2021).

5.3 Alat Pengujian *(Testing Tools)*

Dalam pengembangan perangkat lunak, alat pengujian memainkan peran krusial dalam memastikan kualitas dan keandalan kode sebelum diluncurkan ke produksi. Alat ini dibagi menjadi dua kategori utama yakni *Automated Testing Tools* dan *Static Code Analysis Tools*.

5.3.1 *Automated Testing Tools*

Alat pengujian otomatis memungkinkan pengembang dan QA engineer untuk menjalankan pengujian fungsional dan non-fungsional secara otomatis, tanpa perlu intervensi manual. Hal ini meningkatkan efisiensi pengujian dan memungkinkan tim untuk menemukan dan memperbaiki bug lebih awal dalam siklus pengembangan.

- Selenium
  Selenium adalah *framework* pengujian yang sangat populer untuk aplikasi web. Hal ini memungkinkan pengujian otomatis

di berbagai *browser* dan sistem operasi. Selenium mendukung pengujian melalui berbagai bahasa pemrograman termasuk Java, C#, dan Phyton, memberikan fleksibilitas besar dalam pengembangan skrips pengujian.

- JUnit
  JUnit adalah framework pengujian untuk Java yang memfasilitasi pembuatan dan eksekusi tes unit. Dengan memanfaatkan anotasi dan asersi, JUnit memugkinkan pengembang untuk dengan mudah menentukan kondisi tes dan mengevaluasi hasil tes, sehingga memudahkan pengujian kode secara terperinci.
- TestNG
  Mirip dengan JUnit, TestNG adalah framework pengujian untuk Java yang menyediakan dukungan yang lebih luas untuk berbagai jenis tes termasuk tes unit, tes fungsional, dan tes integrasi. TestNG menawarkan fitur yang lebih canggih seperti pengelompokan tes, eksekusi paralel, dan parameterisasi tes.

5.3.2 *Static Code Analysis Tools*

Alat analisis kode statis mengevaluasi kode sumber tanpa menjalankannya, memungkinkan deteksi dini kesalahan pemrograman, kelemahan keamanan, dan ketidaksesuaian dengan standar kode.

- SonarQube
  SonarQube adalah platform yang komprehensif untuk inspeksi kualitas kode secara terus-menerus. Mengidentifikasi bug, kerentanan keamanan, dan "*code smells*" sambil juga memberikan umpan balik untuk meningkatkan kualitas kode. SonarQube mendukung berbagai bahasa pemrograman dan terintegrasi dengan alur kerja CI/CD.
- ESLint
  ESLint adalah alat analisis kode statis untuk mengidentifikasi pola bermasalah dalam kode JavaScript. ESLint sangat dapat dikonfigurasi, memungkinkan pengembang untuk menentukan aturan sendiri atau menggunakan aturan yang telah

ditetapkan untuk memastikan konsistensi kode dan menghindari kesalahan umum.

- Checkstyle
  Checkstyle digunakan untuk memeriksa kesesuaian kode Java dengan standar kode tertentu. Hal ini membantu memastikan bahwa kode mengikuti praktik penulisan kode yang baik dan konsisten di seluruh proyek. Checkstyle dapat dikustomisasi untuk mengikuti aturan spesifik tim atau organisasi.

5.4 Alat Pembangunan *(Build Tools)*

Alat pembangunan atau *build automation tools* memainkan peran penting dalam proses pengembangan perangkat lunak dengan mengotomatisasi pembuatan kode dari sumber *(source code)* menjadi program yang siap dijalankan *(executable)* atau pustaka *(library)*. Proses ini termasuk kompilasi kode, pengemasan, pengujian dan deployment.

- Gradle
  Gradel adalah sistem otomatisasi *build* yang menggunakan pendekatan berbasis konfigurasi yang dinyatakan *declarative* dan *scripting* yang kuat, untuk merancang proses build yang sangat disesuaikan dan efisien. Gradle adalah alat pilihan untuk proyek-proyek Android karena fleksibilitas dan performanya yang tinggi. Hal ini mendukung bahasa Groovy atau Kotlin untuk *scripting buildnya*, memungkinkan pengembang untuk menulis *build script* yang ringkas namun kuat.
- Maven
  Maven adalah alat manajemen proyek dan pemahaman yang menyediakan cara yang seragam untuk proyek perangkat lunak dibangun, didokumentasikan, dan dilaporkan. Pendekatan konvensi daripada konfigurasi Maven mengurangi kebutuhan untuk menulis script dari awal, sebaliknya mengandalkan model *Project Object Model* (POM) XML untuk mendefinisikan proyek dan ketergantungannya. Maven sangat

cocok untuk proyek Java dan digunakan luas di industri untuk mengelola siklus hidup pembangunan.

- Ant
  Apache Ant adalah alat automasi build yang mirip dengan Make tetapi ditulis dalam bahasa Java, membuatnya ideal untuk proyek berbasis Java. Ant menggunakan file build.xml untuk mendefinisikan tugas dan target. Walaupun Ant kurang modern dibandingkan Gradle dan Maven, masih digunakan dalam beberapa proyek yang membutuhkan tingkat kustomisasi build yang tinggi atau memiliki dependensi pada proses *build legacy*.

Fitur utama dari alat pembangunan ini termasuk kemampuan untuk mengotomatisasi tugas-tugas yang berulang dan kompleks, memastikan konsistensi di lingkungan pengembangan yang berbeda, dan meningkatkan produktivitas pengembang dengan mengurangi waktu yang dihabiskan untuk tugas-tugas rutin. Selain itu, dengan menyediakan dependensi dan manajemen *library* yang terpusat, alat-alat ini membantu mengurangi konflik dan memastikan bahwa semua anggota tim menggunakan versi *library* yang sama untuk proyek.

5.5 Alat *Continuous Integration* dan *Continuous Development* (CI/DP Tools)

Alat *Continuous Integration* dan *Continuous Deployment* (CI/CD) sangat penting dalam praktik DevOps dan pengembangan perangkat lunak agile. Memungkinkan tim untuk mengotomatisasi pengujian dan deployment kode, mempercepat siklus rilis, dan meningkatkan kualitas serta keandalan aplikasi (Duvall et al., 2007).

1. Jenkins
   Jenkins adalah alat otomatisasi *open source* yang ditulis dalam Java. Menyediakan ratusan plugin untuk mendukung pembangunan, *deployment*, dan otomatisasi untuk semua jenis proyek (Smart, 2011). Jenkins sangat fleksibel dan dapat dikonfigurasi untuk hampir semua jenis tugas *Continuous*

Integration dan *Continuous Deployment* (CI/CD), menjadikannya pilihan populer di kalangan pengembang. Cocok untuk proyek-proyek yang memerlukan konfigurasi pipeline yang sangat kustomisasi dan untuk organisasi yang membutuhkan server *Continuous Integration* dan *Continuous Deployment* (CI/CD) on premise. Fitur utamanya, meliputi:

- Otomatisasi proses *build*, *test*, dan *deployment* menggunakan *pipeline* yang dapat dikustomisasi.
- Mendukung integrasi dengan hampir semua alat pengembangan dan sistem manajemen kode sumber.
- Kemampuan untuk dijalankan pada *server* terdistribusi, memungkinkan eksekusi pekerjaan secara paralel.

2. GitLab CI

   GitLab CI adalah bagian dari GitLab, platform DevOps yang menyediakan manajemen repositori git serta fitur *Continuous Integration* dan *Continuous Deployment* (CI/CD) dalam satu aplikasi. Dengan GitLab CI, *pipeline Continuous Integration* dan *Continuous Deployment* (CI/CD) diatur dan dijalankan langsung dari repositori GitLab, memudahkan pengaturan dan penggunaan. Ideal untuk tim yang sudah menggunakan GitLab sebagai manajemen kode sumber dan mencari solusi *Continuous Integration* dan *Continuous Deployment* (CI/CD) yang terintegrasi dengan baik. Berikut fitur utamanya:

   - Integrasi dalam dengan GitLab, memudahkan pengaturan dan pengelolaan pipeline *Continuous Integration* dan *Continuous Deployment* (CI/CD).
   - Konfigurasi pipeline menggunakan file '.gitlab-ci.yml' di repositori, mempermudah pengelolaan dan versioning konfigurasi.
   - Dukungan untuk runner yang terdistribusi, penyebaran pekerjaan pada berbagai mesin.

3. CircleCI

   CircleCI adalah layanan *Continuous Integration* dan *Continuous Deployment* (CI/CD) berbasis cloud yang menawarkan integrasi cepat dan efisien untuk aplikasi web (Gallaba et al., 2022). Dengan CircleCI, tim dapat mengotomatisasi proses

build, *test*, dan *deployment* dengan cepat dan mudah. Cocok untuk startup dan tim pengembangan yang memerlukan solusi *Continuous Integration* dan *Continuous Deployment* (CI/CD) cepat dan mudah tanpa perlu pengaturan infrastruktur yang kompleks. Fitur utama meliputi:

- Konfigurasi pipeline yang mudah dengan menggunakan file 'config.yml' di repositori.
- Integrasi langsung dengan GitHub dan Bitbucket, memudahkan pengaturan dan sinkronisasi proyek.
- Dukungan untuk containerisasi dan orkestrasi, memungkinkan penggunaan Docker dan Kubernetes dalam pipeline.

Alat-alat *Continuous Integration* dan *Continuous Deployment* (CI/CD) membantu tim mengurangi risiko kesalahan manusia, meningkatkan transparansi proses pengembangan, dan memastikan bahwa setiap perubahan kode diuji secara otomatis dan dapat dideploy ke produksi dengan cepat dan aman (Humble & Farley, 2010). Pilihan alat yang tepat bergantung pada kebutuhan spesifik proyek, infrastruktur yang ada, dan pereferensi tim pengembangan.

BAB. 6

DESAIN PERANGKAT LUNAK

6.1 Pengertian Desain Perangkat Lunak

Desain perangkat lunak merupakan rangkaian proses yang melibatkan beberapa tahap, yang bertujuan untuk merancang program perangkat lunak dengan memperhatikan aspek-aspek seperti struktur data, arsitektur perangkat lunak, antarmuka pengguna, dan langkah-langkah dalam penulisan kode (Jayanti & Hendini, 2021). Ini melibatkan pembuatan model konseptual dan teknis yang mendefinisikan cara sistem akan beroperasi, bagaimana komponen akan berinteraksi, dan bagaimana sistem akan memenuhi kebutuhan fungsional dan non-fungsionalnya. Dimulai dari pemahaman masalah (*requirement elicitation*), analisis, desain, implementasi, dan diakhiri dengan pengujian. Selanjutnya, perangkat lunak ditempatkan (*deploy*) pada pelanggan dan dilakukan pemeliharaan terhadapnya.

Pada tingkat konseptual, desain perangkat lunak mencakup pemahaman masalah yang akan dipecahkan, kebutuhan pengguna, serta solusi tingkat tinggi. Ini melibatkan penyusunan skenario penggunaan dan pemetaan aliran data. Pada tingkat teknis, fokusnya adalah pada implementasi detail, termasuk identifikasi

komponen sistem, spesifikasi antarmuka, pemilihan struktur data dan algoritma, serta teknologi yang digunakan. Desain juga mempertimbangkan aspek keamanan, kinerja, ketersediaan, dan skalabilitas.

Dikarenakan desain merupakan proses di mana kebutuhan-kebutuhan diubah menjadi atribut-atribut yang dapat dipahami oleh perangkat lunak maka, tujuan dari desain perangkat lunak adalah untuk menciptakan rencana yang jelas dan terstruktur untuk implementasi sistem perangkat lunak itu sendiri (Budi et al., 2015). Desain yang baik memungkinkan pengembang untuk memahami dengan jelas apa yang harus dibangun, bagaimana membangunnya, dan bagaimana mengukur keberhasilan implementasi. Selain itu, desain perangkat lunak yang baik juga memungkinkan untuk pemeliharaan yang lebih mudah di masa depan, serta penyesuaian terhadap perubahan kebutuhan atau teknologi.

Keluaran dari tahap *requirement elicitation* menjadi input penting pada tahapan analisis. Hal ini krusial karena tahapan ini merupakan langkah awal dalam memahami domain masalah yang akan diselesaikan oleh perangkat lunak. Selanjutnya, hasil dari analisis digunakan sebagai dasar pada tahap desain. Proses ini sama pentingnya karena hasilnya menjadi panduan dalam pembuatan implementasi perangkat lunak.

6.2 Prinsip-Prinsip Desain

6.2.1 Prinsip-Prinsip Umum

Prinsip-prinsip umum dalam desain perangkat lunak adalah aturan atau pedoman yang membimbing pengembang dalam menciptakan sistem yang berkualitas. Selain itu penggunaan prinsip desain bertujuan untuk mengatur elemen-elemen tersebut agar berinteraksi secara sinergis, sehingga dapat meningkatkan aksesibilitas, kegunaan, dan proses pembelajaran (Nugroho & Daniamiseno, 2022). Beberapa prinsip tersebut antara lain:

1. Keterbacaan: Kode harus mudah dipahami dan dipelihara oleh pengembang lain.

2. Keterpisahan Kepentingan: Kode harus dipisahkan menjadi bagian-bagian yang bertanggung jawab atas tugas-tugas tertentu.
3. Keterbukaan: Desain harus memungkinkan untuk perluasan atau modifikasi tanpa harus mengubah bagian-bagian yang sudah ada.
4. Prinsip Kebasahan: Fungsi dan perilaku perangkat lunak harus konsisten dengan harapan pengguna.
5. Kesederhanaan: Desain harus sederhana dan tidak lebih rumit dari yang diperlukan.
6. Kohesi: Kelompokkan kode yang berkaitan erat bersama.
7. Kopling: Minimalkan ketergantungan antara komponen-komponen dalam sistem.
8. Konsistensi: Kode harus konsisten dalam gaya, nomenklatur, dan penggunaan pola desain.
9. Kelasikasi: Abstraksi yang tepat membantu menyembunyikan detail kompleks dan tidak relevan.
10. Kepentingan Pengguna: Desain harus dipahami dari perspektif pengguna akhir.
11. Kualitas: Desain harus mengutamakan aspek-aspek seperti keandalan, kinerja, keamanan, dan skalabilitas. Dengan mengikuti prinsip-prinsip ini, pengembang dapat menciptakan desain perangkat lunak yang efisien, mudah dipahami, dan mudah dipelihara.

6.2.2 Prinsip Desain Berorientasi Objek

Prinsip-prinsip desain berorientasi objek (*Object-Oriented Design Principles*) adalah seperangkat pedoman yang diikuti oleh pengembang perangkat lunak yang menitik beratkan pada fungsionalitas sistem untuk merancang sistem yang modular, fleksibel, dan mudah dimengerti (Aman & Suroso, 2021). Prinsip-prinsip ini membantu memastikan bahwa desain yang dihasilkan memenuhi standar keberlanjutan dan dapat dipelihara dengan baik dalam jangka panjang.

Pertama, *Single Responsibility Principle* (SRP) mengamanatkan bahwa sebuah kelas harus memiliki satu dan hanya satu alasan untuk berubah. Dengan kata lain, kelas tersebut harus memiliki tanggung jawab tunggal atau fokus tunggal dalam konteks fungsionalitas sistem. Prinsip ini mendorong pembuatan kelas-kelas yang memiliki kohesi yang tinggi, sehingga setiap perubahan hanya memengaruhi satu aspek tertentu dari sistem. Kemudian, *Open/Closed Principle* (OCP) menegaskan bahwa entitas perangkat lunak, seperti kelas atau modul, harus terbuka untuk perluasan tetapi tertutup untuk modifikasi. Artinya, fungsionalitas sistem harus dapat diperluas dengan menambahkan kode baru tanpa mengubah kode yang sudah ada. Hal ini mempromosikan desain yang dapat dengan mudah beradaptasi dengan perubahan dan meminimalkan risiko untuk memecah fungsi yang sudah ada.

Liskov Substitution Principle (LSP) berfokus pada hierarki kelas dalam desain berorientasi objek. Prinsip ini menyatakan bahwa objek dari kelas turunan harus dapat digunakan sebagai pengganti objek dari kelas dasarnya tanpa memengaruhi fungsionalitas program. Ini memastikan konsistensi dan keterpercayaan dalam perilaku objek dalam konteks polimorfisme. Selanjutnya, *Interface Segregation Principle* (ISP) menekankan bahwa klien tidak harus bergantung pada metode yang tidak mereka gunakan. Prinsip ini mendorong pembuatan antarmuka yang lebih spesifik untuk setiap klien, sehingga setiap klien hanya bergantung pada metode yang diperlukan untuk fungsionalitasnya.

Terakhir, *Dependency Inversion Principle* (DIP) merujuk pada hubungan antara modul tingkat tinggi dan rendah dalam sistem. Prinsip ini menyatakan bahwa modul tingkat tinggi tidak boleh bergantung langsung pada modul tingkat rendah, melainkan keduanya harus bergantung pada abstraksi. Ini mempromosikan fleksibilitas dan penurunan ketergantungan antar komponen dalam sistem perangkat lunak. Dengan menerapkan prinsip-prinsip ini, pengembang dapat menciptakan desain yang kuat dan adaptif untuk memenuhi kebutuhan kompleksitas dan perubahan dalam pengembangan perangkat lunak.

6.3 Proses Desain

Proses desain dalam desain perangkat lunak adalah tahapan di mana konsep-konsep dan persyaratan yang dikumpulkan selama tahap analisis digunakan untuk merancang arsitektur dan struktur dari sistem perangkat lunak yang akan dikembangkan. Desain perangkat lunak melibatkan rangkaian tahap yang berusaha menciptakan program perangkat lunak dengan memperhatikan berbagai faktor penting seperti organisasi data, struktur global perangkat lunak, antarmuka pengguna, dan proses penulisan kode yang diperlukan (Hartadi et al., 2020). Proses desain ini melibatkan beberapa langkah, yang mencakup:

Analisis Persyaratan (*Requirement Analysis*): Tahap pertama dari proses desain melibatkan pemahaman yang mendalam tentang persyaratan fungsional dan non-fungsional dari sistem yang akan dikembangkan. Ini melibatkan identifikasi kebutuhan pengguna, pemahaman terhadap alur kerja sistem, dan pengumpulan informasi yang diperlukan untuk memahami masalah yang akan diselesaikan.

Perancangan Konseptual (*Conceptual Design*): Pada tahap ini, konsep-konsep tingkat tinggi untuk sistem perangkat lunak dirancang. Ini mungkin melibatkan pembuatan model konseptual, diagram aliran data, atau sketsa arsitektur umum dari sistem. Tujuannya adalah untuk mendefinisikan cara sistem akan beroperasi secara keseluruhan.

Perancangan Arsitektur (*Architectural Design*): Langkah ini melibatkan pengembangan struktur dan arsitektur dari sistem perangkat lunak. Ini termasuk identifikasi komponen-komponen utama sistem, hubungan antara komponen-komponen tersebut, serta pola komunikasi dan aliran data antara komponen-komponen tersebut. Arsitektur harus memenuhi kebutuhan sistem secara keseluruhan dan memungkinkan untuk skalabilitas, keandalan, dan kinerja yang baik.

Perancangan Detail (*Detailed Design*): Setelah arsitektur sistem ditentukan, tahap selanjutnya adalah merancang detail

implementasi dari komponen-komponen sistem. Ini melibatkan spesifikasi antarmuka untuk setiap komponen, pemilihan struktur data dan algoritma yang tepat, serta pemodelan objek dan kelas jika menggunakan pendekatan berorientasi objek.

Perancangan Antarmuka Pengguna (*User Interface Design*): Jika sistem memiliki antarmuka pengguna, langkah ini melibatkan perancangan antarmuka yang intuitif, estetis, dan mudah digunakan. Ini melibatkan pembuatan wireframe, mockup, atau prototipe antarmuka pengguna, serta pemikiran tentang navigasi, tata letak, dan gaya visual.

Validasi Desain (*Design Validation*): Langkah terakhir dari proses desain adalah memvalidasi desain yang telah dibuat. Ini melibatkan pengujian desain terhadap persyaratan fungsional dan non-fungsional yang telah ditentukan sebelumnya, serta mendapatkan umpan balik dari pemangku kepentingan untuk memastikan bahwa desain memenuhi harapan mereka.

Proses desain dalam desain perangkat lunak merupakan tahapan kritis yang memungkinkan untuk mengonseptualisasikan dan merencanakan implementasi sistem perangkat lunak dengan cara yang sistematis dan terstruktur. Desain yang baik memastikan bahwa sistem dapat memenuhi kebutuhan pengguna, mudah dipelihara, dan dapat berkembang seiring waktu.

6.3.1 Desain Arsitektur

Desain arsitektur dalam rekayasa perangkat lunak adalah proses pengembangan struktur global sistem perangkat lunak yang melibatkan pengidentifikasi, pemodelan, dan organisasi elemen-elemen utama serta interaksi di antara mereka. Dlam tahap ini, para pengembang melibatkan pemikiran strategis untuk merancang struktur perangkat lunak yang akan memenuhi kebutuhan fungsional dan non-fungsional dari sistem tersebut (Ramadhan et al., 2020). Tahap ini juga berfokus pada pengembangan struktur keseluruhan atau arsitektur perangkat lunak, yang mencakup tidak hanya cara pengaksesan data dan fungsionalitas sistem, tetapi juga perancangan antarmuka yang

digunakan oleh pengguna untuk berinteraksi dengan perangkat lunak tersebut (Syakti, 2019). Langkah-langkah dalam desain arsitektur mencakup analisis kebutuhan sistem untuk memahami persyaratan fungsional dan non-fungsionalnya secara menyeluruh. Selanjutnya, berdasarkan analisis tersebut, pemilihan pola desain yang sesuai, seperti *Model-View-Controller* (MVC) atau *Microservices*, dilakukan untuk mencapai tujuan desain yang diinginkan.

Selama proses desain arsitektur, keputusan arsitektural diambil untuk memilih struktur dan teknologi yang paling sesuai dengan kebutuhan sistem, mempertimbangkan faktor-faktor seperti kompleksitas sistem, skala proyek, dan ketersediaan sumber daya. Dokumentasi yang baik dan komunikasi yang efektif antara anggota tim pengembangan sangat penting untuk memastikan pemahaman yang konsisten tentang arsitektur sistem dan sinergi dalam implementasi.

Validasi arsitektur dilakukan melalui teknik analisis statis dan dinamis, simulasi, dan prototyping untuk memastikan bahwa desain tersebut memenuhi persyaratan yang ditetapkan serta mengatasi masalah yang diidentifikasi selama tahap desain. Dalam keseluruhan, desain arsitektur merupakan langkah kritis dalam siklus pengembangan perangkat lunak yang memastikan sistem dirancang dengan struktur yang kokoh, fleksibel, dan dapat dipelihara, serta memenuhi kebutuhan dan tujuan proyek secara efektif.

6.3.2 Desain Mendetail

Desain mendetail dalam pengembangan perangkat lunak adalah tahap yang krusial setelah desain arsitektur, di mana fokus utama adalah pada implementasi teknis solusi yang telah direncanakan sebelumnya. Proses ini melibatkan analisis yang mendalam terhadap komponen-komponen sistem yang telah diidentifikasi selama tahap desain arsitektur, dengan tujuan memperinci fungsionalitas dan spesifikasi teknis masing-masing komponen. Spesifikasi ini mencakup penjelasan rinci tentang tanggung jawab dan fungsi setiap komponen dalam konteks

keseluruhan sistem, serta definisi yang jelas mengenai antarmuka antar-komponen untuk memastikan integrasi yang lancar.

Selanjutnya, dalam desain mendetail, proses pemodelan objek dan kelas menjadi aspek penting, terutama dalam konteks pendekatan berorientasi objek. Tahap ini melibatkan penentuan atribut, metode, dan hubungan antar objek atau kelas dengan tujuan memahami aliran informasi di antara mereka. Selain itu, desain mendetail juga melibatkan spesifikasi yang detail tentang struktur data yang digunakan dalam sistem, seperti skema basis data atau struktur koleksi data, untuk memastikan efisiensi penyimpanan dan pengelolaan informasi. Pemilihan algoritma dan teknologi yang tepat menjadi pertimbangan penting dalam desain mendetail, karena akan memengaruhi kinerja, keandalan, dan skalabilitas sistem secara keseluruhan. Selain itu, desain antarmuka pengguna (UI/UX) berfokus pada pengembangan antarmuka yang intuitif dan menarik bagi pengguna akhir, sementara strategi manajemen kesalahan dan pemulihan dirancang untuk mengantisipasi dan mengatasi masalah yang mungkin terjadi selama operasi sistem.

Terakhir, pemantauan kinerja sistem dan pelaporan status sistem menjadi bagian integral dalam desain mendetail. Proses ini melibatkan pemilihan metrik kinerja yang relevan, alat pemantauan yang sesuai, dan penyusunan laporan status yang informatif bagi pemangku kepentingan. Dengan memperhatikan semua aspek ini, desain mendetail memastikan implementasi sistem perangkat lunak sesuai dengan persyaratan dan standar yang telah ditetapkan.

6.4 Model Desain

Model desain dalam desain perangkat lunak adalah representasi struktur, interaksi, dan perilaku sistem perangkat lunak yang akan dikembangkan. Model desain ini menyediakan pandangan abstrak tentang bagaimana komponen-komponen sistem akan berinteraksi satu sama lain untuk mencapai fungsi-fungsi yang diinginkan. Dalam konteks ini, model desain berfungsi

sebagai pedoman bagi para pengembang dalam merancang dan mengimplementasikan sistem perangkat lunak.

Ada berbagai jenis model desain yang digunakan dalam rekayasa perangkat lunak, yang masing-masing menekankan aspek-aspek tertentu dari sistem. Beberapa contoh model desain termasuk Model Konseptual, Model Arsitektur, dan Model Detail. Model Konseptual bertujuan untuk memberikan gambaran umum tentang tujuan dan fungsi sistem, sementara Model Arsitektur fokus pada struktur dan hubungan antara komponen-komponen utama. Model Detail lebih mendalam, menetapkan spesifikasi teknis dari implementasi sistem, termasuk antarmuka, algoritma, dan struktur data yang digunakan. Selain itu, model desain juga dapat menggunakan notasi formal, seperti diagram alur data, diagram kelas UML, atau diagram aktivitas, untuk menggambarkan aspek-aspek sistem dengan lebih jelas dan terstruktur. Penggunaan notasi formal memungkinkan para pengembang untuk berkomunikasi dengan lebih efektif dan menghindari ambiguitas dalam desain sistem.

Dalam keseluruhan, model desain berperan penting dalam pengembangan perangkat lunak karena membantu dalam merencanakan, merancang, dan mengkomunikasikan struktur dan perilaku sistem secara sistematis dan terstruktur. Dengan menggunakan model desain yang sesuai, pengembang dapat menghasilkan sistem perangkat lunak yang efisien, andal, dan mudah dipelihara.

6.5 Tantangan Desain Perangkat Lunak

Desain perangkat lunak merupakan tahap yang penuh dengan tantangan dan kompleksitas. Salah satu tantangan utama dalam desain perangkat lunak adalah memahami dengan tepat kebutuhan pengguna yang bervariasi dan kompleks. Tanpa pemahaman yang mendalam tentang kebutuhan pengguna, risiko kesenjangan antara ekspektasi pengguna dan fungsionalitas yang disediakan oleh perangkat lunak akan meningkat.

Selain itu, dalam desain perangkat lunak, pengembang juga harus mengatasi keterbatasan sumber daya yang tersedia. Hal ini mencakup keterbatasan dalam hal daya komputasi, memori, dan ruang penyimpanan. Khususnya dalam pengembangan perangkat lunak untuk sistem terbenam, di mana sumber daya perangkat keras seringkali terbatas, merancang solusi yang efisien dalam penggunaan sumber daya menjadi tantangan tersendiri.

Kompleksitas teknis juga menjadi tantangan dalam desain perangkat lunak, terutama dalam pengembangan sistem yang besar dan terdistribusi. Memahami dan merancang arsitektur yang sesuai serta memilih teknologi yang tepat untuk mengimplementasikan solusi menjadi hal yang penting. Selain itu, pengembang juga harus mempertimbangkan kinerja dan skalabilitas sistem, yang memerlukan pemilihan algoritma dan struktur data yang efisien. Aspek keamanan dan privasi data juga menjadi fokus penting dalam desain perangkat lunak. Dengan semakin meningkatnya ancaman siber, pengembang perangkat lunak harus memastikan bahwa solusi yang mereka rancang memiliki lapisan keamanan yang kuat dan melindungi data pengguna dengan baik.

Terakhir, manajemen perubahan kebutuhan selama siklus pengembangan juga menjadi tantangan yang tidak bisa diabaikan. Kebutuhan proyek dapat berubah seiring waktu, dan pengembang harus responsif terhadap perubahan tersebut tanpa mengorbankan kualitas atau kestabilan solusi yang sudah ada. Dengan mengatasi tantangan-tantangan ini dengan bijaksana, pengembang dapat menghasilkan desain perangkat lunak yang efektif, sesuai dengan kebutuhan pengguna, dan mampu berkembang seiring waktu.

BAB. 7

PENGUJIAN PERANGKAT LUNAK

7.1 Pengenalan Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses penting dalam siklus pengembangan perangkat lunak yang bertujuan untuk memastikan bahwa perangkat lunak yang dihasilkan memenuhi standar kualitas yang ditetapkan serta memenuhi kebutuhan dan harapan pengguna. Definisi ini mencerminkan peran krusial yang dimainkan oleh pengujian dalam memastikan bahwa perangkat lunak beroperasi sebagaimana mestinya, meminimalkan kesalahan atau cacat yang dapat terjadi, serta memastikan bahwa pengguna dapat menggunakan perangkat lunak dengan nyaman dan efisien. Dalam konteks siklus pengembangan perangkat lunak, pengujian bukanlah sekadar langkah terpisah yang dijalankan setelah pengembangan selesai, melainkan merupakan elemen yang terintegrasi secara erat sepanjang seluruh proses pengembangan.

Pentingnya pengujian perangkat lunak terlihat dari dampak yang signifikan yang dimilikinya terhadap kualitas, keandalan, dan kepuasan pengguna terhadap produk perangkat lunak. Tanpa pengujian yang menyeluruh, perangkat lunak rentan terhadap

cacat atau kesalahan yang dapat mengganggu kinerja, menyebabkan kerugian finansial, atau bahkan membahayakan pengguna. Oleh karena itu, pengujian perangkat lunak bukan hanya merupakan langkah tambahan, melainkan merupakan bagian integral dari proses pengembangan yang harus dilakukan secara terencana dan terstruktur. Selain itu, pengujian perangkat lunak juga memainkan peran penting dalam mengelola risiko yang terkait dengan pengembangan perangkat lunak. Dengan mengidentifikasi cacat atau kelemahan perangkat lunak sedini mungkin, pengujian memungkinkan tim pengembangan untuk melakukan perbaikan dan penyesuaian yang diperlukan sebelum produk diperkenalkan ke pasar atau digunakan dalam lingkungan produksi. (Leloudas, 2023) Ini membantu mengurangi kemungkinan terjadinya kegagalan atau kerusakan serius di masa mendatang, yang dapat memiliki konsekuensi yang merugikan bagi organisasi yang terlibat dalam pengembangan perangkat lunak.

Selain itu, pengujian perangkat lunak juga memberikan keyakinan kepada para pemangku kepentingan bahwa produk perangkat lunak telah melalui proses validasi dan verifikasi yang komprehensif, sehingga dapat dipercaya untuk digunakan dalam berbagai konteks. Ini membangun reputasi perusahaan pengembang, meningkatkan kepercayaan pengguna, dan memperluas peluang pasar. Dengan demikian, pengujian perangkat lunak tidak hanya berkaitan dengan aspek teknis pengembangan, tetapi juga memiliki dampak yang signifikan pada aspek bisnis dan keuangan dari suatu organisasi. Secara keseluruhan, pengujian perangkat lunak adalah langkah yang krusial dalam siklus pengembangan perangkat lunak yang tidak boleh diabaikan. Dengan memastikan bahwa perangkat lunak telah diuji secara menyeluruh, tim pengembangan dapat meminimalkan risiko, meningkatkan kualitas produk, dan memastikan kepuasan pengguna. Oleh karena itu, pengujian perangkat lunak harus dianggap sebagai investasi yang penting dan diberikan perhatian yang memadai dalam setiap proyek pengembangan perangkat lunak.

7.2 Metode Pengujian

Metodologi pengujian adalah pendekatan sistematis untuk merencanakan, mendesain, dan melaksanakan proses pengujian perangkat lunak. Tinjauan berbagai metode pengujian perangkat lunak mencakup berbagai pendekatan yang digunakan untuk menguji berbagai aspek perangkat lunak, seperti fungsionalitas, kinerja, keamanan, dan lainnya (Izzat & Saleem, 2023). Berikut adalah penjelasan rinci tentang beberapa metode pengujian utama:

1. *Blackbox Testing*: *Blackbox testing* merupakan salah satu metode pengujian perangkat lunak yang fokus pada menguji fungsionalitas perangkat lunak tanpa memperhatikan struktur internal atau kode sumbernya. Dalam *blackbox testing*, pengujian dilakukan dari perspektif pengguna eksternal yang tidak memiliki pengetahuan tentang implementasi internal dari perangkat lunak tersebut. Metode-metode *blackbox testing* meliputi pengujian pengguna, pengujian kecocokan, pengujian batas, dan lain-lain.

2. *Whitebox Testing*: *Whitebox testing* adalah metode pengujian perangkat lunak yang melibatkan pemeriksaan struktur internal dan kode sumber dari perangkat lunak yang sedang diuji. Dalam *whitebox testing*, tester memiliki pengetahuan yang mendalam tentang bagaimana perangkat lunak diimplementasikan, termasuk alur kontrol, struktur data, dan logika bisnisnya. Metode-metode *whitebox testing* meliputi pengujian jalur, pengujian kondisi, dan pengujian integrasi.

3. *Greybox Testing*: *Greybox testing* merupakan metode pengujian perangkat lunak yang menggabungkan elemen-elemen dari *blackbox testing* dan *whitebox testing*. Dalam *greybox testing*, tester memiliki akses terbatas terhadap struktur internal atau kode sumber perangkat lunak, namun tidak sepenuhnya terbuka seperti dalam *whitebox testing*. Metode-metode *greybox testing* sering kali melibatkan analisis kode sumber yang terbatas, pengujian integrasi, pengujian regresi, dan pengujian sistem.

4. Pengujian Stres (*Stress Testing*): Pengujian stres, atau *stress testing*, adalah metode pengujian perangkat lunak yang bertujuan untuk mengevaluasi kinerja dan stabilitas perangkat lunak di bawah beban yang berat atau kondisi ekstrem. Dalam pengujian stres, perangkat lunak diuji dengan meningkatkan beban atau tekanan secara bertahap hingga mencapai batas kemampuannya atau hingga terjadi kegagalan.

5. **Pengujian A/B (*A/B Testing*)**: Pengujian A/B, atau A/B testing, adalah metode pengujian perangkat lunak yang membandingkan dua versi dari suatu fitur atau elemen perangkat lunak secara langsung dengan tujuan untuk menentukan versi mana yang memberikan hasil yang lebih baik dalam mencapai tujuan yang diinginkan. Dalam pengujian A/B, dua kelompok pengguna dibagi secara acak: satu kelompok diberikan versi A dari fitur atau elemen, sedangkan kelompok lainnya diberikan versi B. Pengujian A/B memungkinkan pengembang untuk menguji perubahan-perubahan kecil atau besar dalam perangkat lunak secara langsung dengan pengguna nyata.

6. **Pengujian Usabilitas (*Usability Testing*)**: Pengujian Usabilitas (*Usability Testing*) adalah metode pengujian perangkat lunak yang fokus pada pengalaman pengguna saat menggunakan perangkat lunak. Tujuannya adalah untuk mengevaluasi sejauh mana perangkat lunak dapat digunakan dengan efektif, efisien, dan memuaskan oleh pengguna sesuai dengan tujuan yang diinginkan. Dengan menerapkan berbagai metode pengujian ini secara komprehensif, pengembang perangkat lunak dapat memastikan bahwa perangkat lunak yang dikembangkan memenuhi standar kualitas yang ditetapkan dan berfungsi dengan baik dalam lingkungan produksi.

Dengan menerapkan berbagai metode pengujian ini secara komprehensif, pengembang perangkat lunak dapat memastikan bahwa perangkat lunak yang dikembangkan memenuhi standar kualitas yang ditetapkan dan berfungsi dengan baik dalam lingkungan produksi.

7.3 Proses Pengujian Perangkat Lunak

Proses pengujian perangkat lunak adalah serangkaian langkah-langkah sistematis yang dilakukan untuk memastikan kualitas perangkat lunak sebelum dirilis ke pasar. Tahapan-tahapan dalam proses pengujian mencakup perencanaan pengujian, desain kasus uji, eksekusi uji, dan pelaporan hasil (Suryn, 2014). Berikut adalah penjelasan rinci tentang setiap tahapan:

1. **Perencanaan Pengujian**
   - Tahap ini merupakan awal dari proses pengujian perangkat lunak di mana rencana pengujian disusun.
   - Rencana pengujian mencakup pengidentifikasian sumber daya yang diperlukan, penentuan lingkup pengujian, pembuatan jadwal, dan penentuan strategi pengujian.
   - Tujuan dari perencanaan pengujian adalah untuk memastikan bahwa pengujian dilakukan secara efisien dan efektif sesuai dengan tujuan dan persyaratan proyek.

2. **Desain Kasus Uji**
   - Tahapan ini melibatkan pembuatan kasus uji yang akan digunakan untuk menguji fungsionalitas dan kinerja perangkat lunak.
   - Kasus uji adalah rangkaian langkah-langkah yang merinci situasi penggunaan perangkat lunak serta hasil yang diharapkan.
   - Desain kasus uji juga melibatkan identifikasi data uji yang diperlukan dan persiapan lingkungan pengujian.

3. **Eksekusi Uji**
   - Setelah kasus uji telah dibuat, tahap selanjutnya adalah eksekusi uji di mana perangkat lunak diuji sesuai dengan skenario yang telah ditentukan.
   - Pada tahap ini, pengujian dilakukan oleh tim pengujian yang telah disiapkan dengan lingkungan pengujian yang sesuai.

- Setiap kasus uji dieksekusi dengan memasukkan data uji yang relevan dan mengamati hasilnya untuk memastikan bahwa perangkat lunak berperilaku sesuai dengan yang diharapkan.

4. **Pelaporan Hasil**
   - Setelah semua kasus uji telah dieksekusi, hasil pengujian dicatat dan dilaporkan kepada pemangku kepentingan proyek.
   - Laporan hasil pengujian berisi informasi tentang bug yang ditemukan, kinerja perangkat lunak, kelengkapan pengujian, dan rekomendasi untuk perbaikan.
   - Pelaporan hasil penting untuk memberikan pemahaman yang jelas tentang kualitas perangkat lunak kepada tim pengembang dan manajemen proyek, sehingga tindakan perbaikan dapat diambil jika diperlukan.

Proses pengujian perangkat lunak adalah bagian integral dari siklus pengembangan perangkat lunak yang bertujuan untuk memastikan bahwa produk yang dihasilkan memenuhi standar kualitas yang ditetapkan (Leloudas, 2023). Dengan mengikuti tahapan-tahapan dalam proses pengujian secara sistematis, tim pengujian dapat mengidentifikasi dan mengatasi masalah dengan efektif sebelum perangkat lunak dirilis ke pasar.

7.4 Aspek Kualitas dalam Pengujian Perangkat Lunak

Aspek kualitas dalam pengujian perangkat lunak mencakup berbagai dimensi yang harus dievaluasi untuk memastikan bahwa perangkat lunak yang dikembangkan memiliki tingkat kualitas yang tinggi. Ini meliputi kualitas fungsional, kualitas non-fungsional, dan kualitas proses (Yi & Alytona, 2022). Berikut adalah penjelasan rinci tentang setiap aspek:

1. **Kualitas Fungsional**
   - Kualitas fungsional berkaitan dengan kemampuan perangkat lunak untuk memenuhi persyaratan fungsional yang telah ditetapkan.

- Ini mencakup sejauh mana perangkat lunak dapat melakukan fungsi-fungsi yang diinginkan dan diharapkan oleh pengguna.
- Aspek-aspek kualitas fungsional dapat mencakup keakuratan, keandalan, keamanan, dan kesesuaian perangkat lunak dengan spesifikasi yang ada.
- Pengujian fungsional dilakukan untuk memastikan bahwa setiap fitur atau fungsi perangkat lunak berfungsi dengan benar sesuai dengan yang diharapkan.

2. Kualitas Non-fungsional

- Kualitas non-fungsional berkaitan dengan aspek-aspek perangkat lunak yang tidak langsung berkaitan dengan fungsionalitasnya, tetapi mempengaruhi pengalaman pengguna dan kinerja keseluruhan perangkat lunak.
- Ini termasuk kinerja (performa), keamanan, ketersediaan, skalabilitas, dan kegunaan perangkat lunak.
- Pengujian non-fungsional dilakukan untuk mengevaluasi sejauh mana perangkat lunak dapat memenuhi persyaratan non-fungsional yang telah ditetapkan.
- Contoh pengujian non-fungsional mencakup pengujian beban, pengujian kecepatan, pengujian keamanan, dan pengujian aksesibilitas.

3. Kualitas Proses

- Kualitas proses berkaitan dengan konsistensi, efisiensi, dan efektivitas proses pengembangan perangkat lunak yang digunakan untuk menghasilkan produk perangkat lunak.
- Ini meliputi penerapan praktik-praktik terbaik dalam pengelolaan proyek, pengembangan perangkat lunak, pengujian, dan penjaminan kualitas.
- Kualitas proses juga mencakup kesesuaian dengan standar dan pedoman yang relevan, seperti ISO 9001 atau *Capability Maturity Model Integration* (CMMI).
- Evaluasi kualitas proses membantu memastikan bahwa pengembangan perangkat lunak dilakukan secara konsisten

dan efisien, sehingga menghasilkan produk yang berkualitas tinggi secara konsisten.

Dengan memperhatikan dan mengevaluasi ketiga aspek kualitas ini secara menyeluruh, tim pengembangan dan pengujian dapat memastikan bahwa perangkat lunak yang dihasilkan memenuhi standar kualitas yang ditetapkan dan memenuhi harapan pengguna. Ini penting untuk memastikan kepuasan pengguna, reputasi perusahaan, dan kesuksesan produk di pasar.

7.5 Aspek Kualitas dalam Pengujian Perangkat Lunak

Aspek kualitas dalam alat dan teknologi pengujian perangkat lunak sangat penting karena alat yang tepat dapat meningkatkan efisiensi, akurasi, dan cakupan pengujian. Berikut adalah penjelasan rinci tentang beberapa aspek kualitas dalam alat dan teknologi pengujian:

1. **Perangkat Lunak Otomatisasi Pengujian**
   - Perangkat lunak otomatisasi pengujian adalah alat yang digunakan untuk mengotomatisasi proses pengujian perangkat lunak, termasuk pengujian fungsional, pengujian regresi, dan pengujian beban.
   - Alat otomatisasi pengujian membantu meningkatkan efisiensi dan akurasi pengujian, mempercepat siklus pengembangan, dan mengurangi ketergantungan pada pengujian manual yang rentan terhadap kesalahan manusia.
   - Kualitas alat otomatisasi pengujian mencakup keandalan, skalabilitas, kemudahan penggunaan, dan kemampuan untuk mengintegrasikan dengan alat pengembangan perangkat lunak yang ada.
2. **Alat Manajemen Uji**
   - Alat manajemen uji adalah perangkat lunak yang digunakan untuk mengelola seluruh proses pengujian, termasuk perencanaan pengujian, pelaksanaan kasus uji, pelacakan hasil uji, dan pelaporan.

- Alat ini membantu tim pengujian dalam mengatur sumber daya, mengkoordinasikan pekerjaan, dan memantau kemajuan pengujian.
- Kualitas alat manajemen uji mencakup kemampuan untuk menyimpan dan mengelola informasi uji dengan aman, melacak bug dan masalah, serta menyediakan laporan yang informatif dan mudah dipahami.

3. **Alat Pelaporan**
   - Alat pelaporan digunakan untuk menyajikan hasil pengujian secara terstruktur dan informatif kepada pemangku kepentingan proyek.
   - Alat ini memungkinkan tim pengujian untuk membuat laporan yang mencakup statistik pengujian, tren, metrik kualitas, dan ringkasan kesimpulan.
   - Kualitas alat pelaporan mencakup kemampuan untuk menyesuaikan format laporan, menyediakan visualisasi data yang jelas, dan mendukung kolaborasi antar tim.

Pemilihan alat dan teknologi pengujian yang tepat sangat penting untuk memastikan efektivitas dan efisiensi pengujian perangkat lunak. Evaluasi kualitas alat harus mencakup pertimbangan terhadap kebutuhan proyek, kemampuan integrasi dengan alat dan proses yang ada, dukungan dari vendor, serta kemampuan untuk memenuhi persyaratan fungsional dan non-fungsional (Wu, 2023). Dengan menggunakan alat dan teknologi pengujian yang berkualitas, tim pengembangan perangkat lunak dapat meningkatkan kualitas produk, mempercepat waktu ke pasar, dan memastikan kepuasan pengguna akhir.

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).