Get Press Indonesia

REKAYASA PERANGKAT LUNAK:
KONSEP, METODE, DAN PRAKTIK TERBAIK

Ade Maulana, Nono Heryana, Johni S Pasaribu, Addin Aditya, Elisawati, Rudiansyah, Amna, Angga Aditya Permana, Arief Yanto Rukmana, Rahman Abdillah, Teguh Wahyono

ISBN 978-623-198-658-0

REKAYASA PERANGKAT LUNAK: KONSEP, METODE, DAN PRAKTIK TERBAIK

Ade Maulana
Nono Heryana
Johni S Pasaribu
Addin Aditya
Elisawati
Rudiansyah
Amna
Angga Aditya Permana
Arief Yanto Rukmana
Rahman Abdillah
Teguh Wahyono

GET PRESS INDONESIA

REKAYASA PERANGKAT LUNAK: KONSEP, METODE, DAN PRAKTIK TERBAIK

Penulis :
Ade Maulana
Nono Heryana
Johni S Pasaribu
Addin Aditya
Elisawati
Rudiansyah
Amna
Angga Aditya Permana
Arief Yanto Rukmana
Rahman Abdillah
Teguh Wahyono

ISBN : 978-623-198-658-0

Editor : Diana Purnama Sari,. S.E M.E
Penyunting : Tri Putri Wahyuni,S.Pd
Desain Sampul dan Tata Letak : Atyka Trianisa, S.Pd

Penerbit : GET PRESS INDONESIA
Anggota IKAPI No. 033/SBA/2022

Redaksi :
Jln. Palarik Air Pacah No 26 Kel. Air Pacah
Kec. Koto Tangah Kota Padang Sumatera Barat
Website : www. getpress.co.id
Email : adm.getpress@gmail.com

Cetakan pertama, 2 September 2023

KATA PENGANTAR

Segala Puji dan syukur atas kehadirat Allah SWT dalam segala kesempatan. Sholawat beriring salam dan doa kita sampaikan kepada Nabi Muhammad SAW. Alhamdulillah atas Rahmat dan Karunia-Nya penulis telah menyelesaikan Buku Rekayasa Perangkat Lunak: Konsep, Metode, Dan Praktik Terbaik ini.

Buku ini membahas Pengantar Rekayasa Perangkat Lunak, Manajemen Proyek Perangkat Lunak, Pengembangan Perangkat Lunak Agile, Rekayasa Kebutuhan Perangkat Lunak, Pemodelan Sistem, Arsitektur Perangkat Lunak, Desain dan Implementasi Perangkat Lunak, Pengujian Perangkat Lunak,Evolusi Perangkat Lunak, Penggunaan Ulang Perangkat Lunak, Rekayasa Perangkat Lunak Berbasis Komponen.

Proses penulisan buku ini berhasil diselesaikan atas kerjasama tim penulis. Demi kualitas yang lebih baik dan kepuasan para pembaca, saran dan masukan yang membangun dari pembaca sangat kami harapkan.

Penulis ucapkan terima kasih kepada semua pihak yang telah mendukung dalam penyelesaian buku ini. Terutama pihak yang telah membantu terbitnya buku ini dan telah mempercayakan mendorong, dan menginisiasi terbitnya buku ini. Semoga buku ini dapat bermanfaat bagi masyarakat Indonesia.

Padang, 2 September 2023

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENGANTAR REKAYASA PERANGKAT LUNAK

Oleh Ade Maulana

1.1 Pendahuluan

Rekayasa perangkat lunak memainkan peran yang sangat penting dalam menjalankan fungsi pemerintahan, masyarakat, bisnis, dan lembaga nasional serta internasional. Keberadaan perangkat lunak menjadi krusial dalam menjalankan dunia saat ini. Sistem komputer berbasis perangkat lunak mengendalikan infrastruktur dan utilitas nasional, sedangkan sebagian besar produk listrik seperti komputer dan perangkat lunak pengendali menjadi bagian tak terpisahkan. Secara menyeluruh, manufaktur dan distribusi industri telah sepenuhnya mengadopsi komputerisasi, dan hal yang sama juga berlaku untuk sistem keuangan. Industri hiburan, musik, permainan komputer, dan film serta televisi, sangat bergantung pada keberadaan perangkat lunak. Lebih dari 64.4% penduduk dunia memiliki telepon genggam yang dikontrol oleh perangkat lunak, dan pada tahun 2023, hampir seluruhnya terhubung dengan internet (Kemp, 2023) .

Pertumbuhan startup dan perangkat lunak saling terkait erat dalam era digital saat ini. Startup menggunakan teknologi dan prinsip rekayasa perangkat lunak untuk mengembangkan solusi inovatif yang memecahkan masalah atau memenuhi kebutuhan pasar yang belum terpenuhi (Maulana et al., 2018). Startup menerapkan metode pengembangan yang cepat seperti pendekatan Agile, sementara perangkat lunak menjadi inti dari produk atau layanan yang ditawarkan. Di sisi lain, perangkat

lunak memainkan peran penting dalam mendukung pertumbuhan startup dengan mengelola operasi internal, analisis data, memperluas jangkauan layanan, dan memberikan pengalaman pengguna yang baik. Pertumbuhan startup juga mendorong persaingan dan inovasi di industri perangkat lunak, menciptakan ekosistem dinamis yang terus berkembang dalam memenuhi kebutuhan pasar yang terus berubah.

Sistem perangkat lunak adalah entitas abstrak yang tidak terikat oleh sifat material, hukum fisik, atau proses manufaktur. Ini memberikan fleksibilitas tanpa batas dalam rekayasa perangkat lunak. Namun, kompleksitas dan biaya tinggi menjadi tantangan karena tidak ada batasan fisik. Setiap jenis sistem perangkat lunak, mulai dari yang sederhana hingga kompleks, membutuhkan pendekatan unik dalam rekayasa perangkat lunak. Mengembangkan sistem informasi organisasi, pengendali instrumen ilmiah, atau permainan komputer yang grafis intensif membutuhkan pendekatan yang berbeda. Meskipun semua aplikasi membutuhkan rekayasa perangkat lunak, tetapi metode dan teknik yang digunakan berbeda-beda.
Pada praktiknya, pengembangan perangkat lunak tidak selalu berjalan dengan lancar. Kegagalan rekayasa perangkat lunak disebabkan oleh dua hal yakni (Sommerville, 2016) :

1. Kompleksitas sistem yang meningkat : Dalam menghadapi peningkatan kompleksitas sistem yang dibangun, teknik rekayasa perangkat lunak harus berkembang untuk menyampaikan perangkat lunak yang lebih kompleks dengan lebih cepat dan mengatasi tantangan baru yang muncul.
2. Kegagalan dalam menggunakan metode rekayasa perangkat lunak : Seringkali, orang menulis program komputer tanpa menerapkan metode dan teknik rekayasa perangkat lunak, terutama dalam pengembangan perangkat lunak di perusahaan. Hal ini dapat mengakibatkan perangkat lunak yang lebih mahal dan

kurang dapat diandalkan. Untuk mengatasi masalah ini, diperlukan pendidikan dan pelatihan yang lebih baik dalam rekayasa perangkat lunak.

1.2 Pengembangan Perangkat Lunak Secara Profesional

Pengembangan perangkat lunak dapat dilakukan oleh siapa saj, selama mereka dapat menulis program, mulai dari orang bisnis menulis program dalam bentuk *spreadsheet* untuk mempermudah pekerjaannya, ilmuan dan akademisi yang menulis program untuk memproses data eksperimentnya, ada juga yang memiliki minat dan kesenangan pribadi untuk menulis program.

Sebagian besar dari orang yang menulis program merupakan orang yang menjalankan aktivitas professional di mana program ditulis untuk mengembakan perangkat lunak dengan tujuan bisnis, dimasukkan kedalam perangkat lain atau sebagai produk perangkat lunak seperti sistem informasi dan sistem desain berbantuan komputer. Perbedaan utamanya adalah bahwa perangkat lunak profesional ditujukan untuk digunakan oleh orang lain selain pengembangnya dan bahwa tim, bukan individu, biasanya mengembangkan perangkat lunak tersebut. Perangkat lunak tersebut dikelola dan diubah sepanjang masa penggunaannya.

Rekayasa perangkat lunak dimaksudkan untuk mendukung pengembangan perangkat lunak profesional daripada pemrograman individu. Ini melibatkan teknik yang mendukung spesifikasi program, desain, dan evolusi, yang tidak umumnya relevan untuk pengembangan perangkat lunak pribadi.

Banyak orang mengira bahwa perangkat lunak hanya merujuk pada program komputer. Namun, dalam konteks rekayasa perangkat lunak, perangkat lunak tidak hanya

mencakup program-program itu sendiri, tetapi juga segala dokumentasi terkait, perpustakaan, situs web dukungan, dan data konfigurasi yang diperlukan agar program-program tersebut dapat digunakan dengan baik. Sebuah sistem perangkat lunak yang dikembangkan secara profesional seringkali terdiri dari lebih dari satu program. Sistem tersebut dapat terdiri dari beberapa program yang berbeda dan file konfigurasi yang digunakan untuk mengatur program-program tersebut. Selain itu, sistem tersebut juga mungkin melibatkan dokumentasi sistem yang menjelaskan struktur sistem, dokumentasi pengguna yang memberikan petunjuk penggunaan sistem, serta situs web yang memungkinkan pengguna mengunduh informasi produk terbaru.

Pengembang perangkat lunak berfokus pada perangkat lunak yang dapat dijual kepada pelanggan, ada dua jenis produk perangkat lunak (Sommerville, 2016):

1. Produk generik, merupakan perangkat lunak yang diproduksi oleh organisasi dan dijual di pasar kepada siapapun yang ingin membelinya. Jenis perangkat lunak ini mencakup aplikasi yang dirancang untuk pasar tertentu seperti sistem informasi perpusatakaan, sistem akuntansi, termasuk aplikasi seperti gojek, youtube atau Instagram yang berbasis *software-as-a-service*.
2. Perangkat lunak yang disesuaikan, merupakan perangkat lunak yang dikembangkan khusus untuk pelanggan tertentu. Kontraktor perangkat lunak merancang dan mengimplementasikan perangkat lunak khusus tersebut untuk pelanggan. Contoh perangkat lunak ini adalah sistem pengendalian untuk perangkat elektronik tertentu, sistem yang ditulis unutk mendukung proses bisnis tertentu, atau sistem manajemen inventaris yang dikembangkan khusus untuk perusahaan tertentu.

Perbedaan antara jenis produk sistem semakin kabur seiring dengan semakin banyaknya sistem yang dibangun dengan dasar produk generik dan kemudian disesuaikan untuk memenuhi kebutuhan pelanggan. Sistem *Enterprise Resource Planning* (ERP), seperti SAP dan Oracle, merupakan contoh terbaik dari pendekatan ini. Dalam pendekatan ini, sistem yang besar dan kompleks disesuaikan untuk perusahaan dengan memasukkan informasi tentang aturan dan proses bisnis, laporan yang dibutuhkan, dan lain-lain.

Ketika membicarakan tentang kualitas perangkat lunak profesional, penting untuk mempertimbangkan bahwa perangkat lunak tersebut digunakan dan diubah oleh orang lain selain pengembangnya. Oleh karena itu, kualitas perangkat lunak tidak hanya berkaitan dengan apa yang dilakukan perangkat lunak secara fungsional. Lebih dari itu, kualitas juga mencakup perilaku perangkat lunak saat dieksekusi, struktur dan organisasi program sistem, serta dokumentasi terkait. Atribut kualitas atau non-fungsional perangkat lunak tercermin dalam hal seperti waktu respons perangkat lunak terhadap permintaan pengguna dan kejelasan kode program. Tabel 1.1 menjelaskan mengenai atribut perangkat lunak yang baik.

Tabel 1.1. Karakteristik Produk Perangkat Lunak yang Baik

Karakteristik Produk	**Deskripsi**
Akseptabilitas	Perangkat lunak harus dapat dimengerti, digunakan dan kompatibel dengan sistem lain yang digunakan
Keandalan dan keamanan	Perangkat lunak yang dapat diandalkan tidak boleh menyebabkan kerusakan fisik atau ekonomi dalam kasus kegagalan sistem dan harus dapat melindungi sistem agar tidak

Karakteristik Produk	Deskripsi
	dapat diakses atau dirusak oleh pengguna.
Efisiensi	Perangkat lunak harus efisien dalam pengelolaan sumber daya sistem, termasuk responsivitas, waktu proses, dan penggunaan sumber daya lainnya
Kemudahan Perawatan	Perangkat lunak harus ditulis sedemikian rupa sehingga dapat mengikuti perubahan kebutuhan pelanggan yang tidak terhindarkan dalam lingkungan bisnis

Sumber (Sommerville, 2016)

1.2.1 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak dapat didefinisikan sebagai pemrograman yang terintegrasi seiring waktu. Pemrograman memang merupakan bagian penting dari rekayasa perangkat lunak karena pemrograman adalah cara untuk menghasilkan perangkat lunak baru. Namun, terdapat perbedaan antara tugas pemrograman (pengembangan) dan tugas rekayasa perangkat lunak (pengembangan, modifikasi, pemeliharaan). Dalam konteks ini, penambahan dimensi waktu menjadi hal yang penting dalam pemrograman. Rekayasa perangkat lunak bukan hanya sekadar pemrograman, melainkan melibatkan aspek pengembangan perangkat lunak yang mencakup modifikasi dan pemeliharaan serta melibatkan dimensi waktu (Winters et al., 2020).

Rekayasa adalah bagaimana mencapai hasil dengan kualitas yang dibutuhkan sesuai jadwal dan anggaran. Secara umum, pengembang perangkat lunak mengadopsi pendekatan sistematis dan terorganisir dalam pekerjaan mereka sebagai

cara paling efektif untuk menghasilkan perangkat lunak yang berkualitas tinggi. Namun, pada rekayasa perangkat lunak pemilihan metode didasarkan pada sejumlah keadaan sehingga dibutuhkan pendekatan yang berbeda untuk mengembangkan perangkat lunak yang berbeda. Proses perangkat lunak yang lebih fleksibel yang dapat menyesuaikan perubahan dengan cepat sangat sesuai untuk pengembangan sistem berbasis web interaktif dan aplikasi seluler, yang membutuhkan kombinasi keterampilan perangkat lunak dan desain grafis.

Menurut Braude dan Bernstein (2016), untuk mencapai keberhasilan pengembangan perangkat lunak, setiap akitivitas harus diorganisir menggunakan proses teratur dan terdefinisi dengan baik. Aktivitas tersebut dikenal dengan 4P dalam rekayasa perangkat lunak, yakni *People, Product, Project* dan *Process.* Proyek perangkat lunak yang sukses harus merencanakan dan mengatasi keempat elemen ini dengan memadai. Terkadang, kebutuhan dari setiap elemen P saling bertentangan, dan diperlukan keseimbangan yang tepat agar proyek dapat berhasil. Mengefokuskan pada satu elemen P tanpa memperhatikan yang lain dapat menyebabkan kegagalan proyek. Sebagai contoh, jika orang-orang diorganisir menjadi tim yang efisien dan diberikan sumber daya yang mereka butuhkan untuk menjalankan peran mereka, sebuah proyek masih dapat gagal jika tidak ada proses perangkat lunak yang jelas yang harus diikuti, karena hal ini dapat menyebabkan kekacauan.

Pendekatan sistematis yang digunakan dalam rekayasa perangkat lunak kadang-kadang disebut sebagai proses perangkat lunak. Proses perangkat lunak adalah urutan aktivitas yang menghasilkan produk perangkat lunak. Ada empat aktivitas dasar yang umum pada semua proses perangkat lunak, yakni (Sommerville, 2016):

1. Spesifikasi perangkat lunak, di mana pelanggan dan pengembang mendefinisikan perangkat lunak yang akan diproduksi dan batasan-batasan operasinya.
2. Pengembangan perangkat lunak, di mana perangkat lunak dirancang dan diprogram.
3. Validasi perangkat lunak, di mana perangkat lunak diperiksa untuk memastikan sesuai dengan kebutuhan pelanggan.
4. Evolusi perangkat lunak, di mana perangkat lunak dimodifikasi untuk mencerminkan perubahan kebutuhan pelanggan dan pasar.

1.2.2 Keanekaragaman Rekayasa Perangkat Lunak

Rekayasa perangkat lunak adalah pendekatan sistematis dalam produksi perangkat lunak yang mempertimbangkan aspek praktis biaya, jadwal, dan keandalan, serta kebutuhan pelanggan dan produsen perangkat lunak. Metode, alat, dan teknik spesifik yang digunakan bergantung pada organisasi yang mengembangkan perangkat lunak, jenis perangkat lunak yang dibuat, dan orang-orang yang terlibat dalam proses pengembangan. Tidak ada metode rekayasa perangkat lunak universal yang cocok untuk semua sistem dan perusahaan. Menurut Sommerville (2016), Faktor paling signifikan dalam menentukan metode dan teknik rekayasa perangkat lunak yang paling penting adalah jenis aplikasi yang sedang dikembangkan. Ada banyak jenis aplikasi, diantaranya :

1. *Stand-alone application* : Aplikasi yang berjalan di komputer pribadi atau perangkat seluler dan mencakup semua fungsionalitas yang diperlukan, tanpa perlu terhubung ke jaringan atau internet.
2. *Interactive transaction-based applications* : aplikasi yang dieksekusi pada komputer jarak jauh dan diakses oleh pengguna dari komputer, ponsel, atau tablet mereka

sendiri. Termasuk aplikasi web seperti aplikasi e-commerce serta layanan berbasis cloud

3. *Embeded control system* : Sistem kontrol perangkat lunak yang mengendalikan dan mengelola perangkat keras.
4. *Batch processing system* : sistem bisnis yang dirancang untuk memproses data dalam jumlah besar, dengan memproses sejumlah besar input individual untuk menghasilkan output yang sesuai, seperti sistem penagihan berkala seperti sistem penagihan telepon dan sistem pembayaran gaji.
5. *Entertainment systems* : Sistem hiburan untuk penggunaan pribadi yang bertujuan untuk menyenangkan pengguna, terutama dalam bentuk permainan yang beragam dan mungkin dijalankan pada perangkat keras konsol khusus, dengan kualitas interaksi pengguna sebagai karakteristik yang paling menonjol.
6. *System for modeling and simulation* : Sistem yang dikembangkan oleh ilmuwan dan pengembang untuk memodelkan proses fisik atau situasi yang melibatkan interaksi antara banyak objek terpisah, seringkali memerlukan komputasi yang intensif dan kinerja tinggi dari sistem paralel untuk menjalankannya.
7. *Data collection and analysis systems* : Sistem yang mengumpulkan data dari lingkungannya dan mengirimkannya ke sistem lain untuk diproses, dengan perangkat lunak yang mungkin harus berinteraksi dengan sensor dan sering kali diinstal dalam lingkungan yang tidak ramah seperti di dalam mesin atau di lokasi yang sulit dijangkau, dan analisis "big data" dapat melibatkan penggunaan sistem berbasis cloud yang melakukan analisis statistik dan mencari hubungan dalam data yang terkumpul.
8. *System of systems* : Merupakan kumpulan dari berbagai sistem perangkat lunak yang digunakan dalam skala besar,

seperti dalam perusahaan atau organisasi besar. Beberapa sistem dalam perangkaiannya dapat berupa produk perangkat lunak generik yang umum digunakan, seperti sistem ERP (*Enterprise Resource Planning*).

1.2.2 Rekayasa Perangkat Lunak Internet

Perkembangan internet dan World Wide Web telah memberikan dampak ayng besar pada kehidupan saat ini. Pada awal komersialisasi internet, web hanya berfungsi sebagai took informasi yang dapat diakses secara universal dan memiliki sedikit dampak pada sistem perangkat lunak (Muttaqin, Romindo, *et al.,* 2023). Sekitar tahun 2000, web mulai berevolusi, dan semakin banyak fungsionalitas yang ditambahkan ke dalam browser. Hal ini memungkinkan pengembangan sistem berbasis web di mana sistem-sistem ini dapat diakses menggunakan web browser sebagai antarmuka pengguna.

Perangkat lunak berbasis web diinstal pada web server, hal ini membuat perubahan dan peningkatan perangkat lunak menjadi jauh lebih murah karena tidak perlu menginstal pada setiap PC. Hal ini juga mengurangi biaya pengembangan antarmuka pengguna yang biasanya mahal. Saat ini banyak sistem perangkat lunak perusahaan telah beralih ke interaksi berbasis web. Dengan menggunakan perangkat lunak berbasis web, memungkinkan perangkat lunak dapat diakses kapan saja dan dimana saja (Muttaqin, Sondakh, *et al.,* 2023)

Kehadiran web telah menghasilkan perubahan dramatis dalam cara perangkat lunak bisnis dikelola. Sebelum adanya web, perangkat lunak bisnis sebagian besar berupa program monolitik yang berjalan pada satu komputer atau kumpulan komputer. Komunikasi dilakukan secara lokal, dalam suatu organisasi. Namun, sekarang perangkat lunak menjadi sangat terdistribusi, bahkan melintasi dunia. Perangkat lunak bisnis

tidak lagi dikembangkan dari awal, tetapi melibatkan penggunaan kembali terhadap komponen dan program.

1.3 Etika Rekayasa Perangkat Lunak

Rekayasa perangkat lunak dilakukan dengan menerapkan kerangka sosial dan hukum untuk membatasi kebebasan orang-orang yang berkerja pada bidang ini. Seorang pengembang perangkat lunak harus menerima bahwa pekerjaan yang dilakukan melibatkan tanggung jawab yang jauh lebih luas dari sekedar penerapan keterampilan teknis. Pengembang perangkat lunak harus berprilaku dengan etika serta tanggung jawab moral agar dapat dihormati sebagai seorang pengembang professional dengan menjunjung tinggi kejujuran dan integritas. Secara umum ada beberapa etika yang harus ditaati oleh pengembang perangkat lunak, yakni (Sommerville, 2016):

1. Kerahasiaan : Seorang pengembang perangkat lunak harus menjaga kerahasiaan dari pihak pengusaha atau klien, tanpa memandang apakah ada perjanjian kerahasiaan resmi yang telah ditanda tangani
2. Kompetensi : Seorang pengembang perangkat lunak tidak boleh dengan sengaja berbohong mengenai kompetensinya dan menerima pekerjaan diluar dari kemampuannya.
3. Hak kekayaan intelektual : Seorang pengembang perangkat lunak harus memahami hukum yang mengatur penggunaan hak kekayaan intelektual seperti paten dan hak cipta.
4. Penyalahgunaan Komputer : Seorang pengembang perangkat lunak tidak boleh menggunakan keterampilan teknis untuk menyalahgunakan komputer orang lain. Penyalahgunaan computer dapat bervariasi dari hal yang relative sepele seperti bermain game di computer

perusahaan sampai hal yang sangat serius seperti menyebarkan virus atau malware.

DAFTAR PUSTAKA

Braude, E. J., & Bernstein, M. E. 2016. *Software Engineering : Modern Approaches Second Edition*. Waveland Press.

Kemp, S. 2023. *2023 Global Digital Report*.

Maulana, A., Sucahyo, Y. G., Ruldeviyani, Y., & Gandhi, A. 2018. Requirements for platform-based startup survival: A qualitative exploratory study. *2018 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 137–142.

Muttaqin, M., Romindo, R., Moedjahedy, J., Pratama, Y. A., Andryanto, A., Sihananto, A. N., Widarman, A., Kurniadi, W., Maulana, A., Simarmata, J., & others. 2023. *Pengantar Internet*. Yayasan Kita Menulis.

Muttaqin, M., Sondakh, D. E., Syafriadi, S., Rengganis, A., Simarmata, J., Rotikan, R., Warnilah, A. I., Lengkong, O. H., Adam, S. I., Fairuzabadi, M., & others. 2023. *Pengantar Teknologi Digital*. Yayasan Kita Menulis.

Sommerville, I. 2016. *Software engineering*.

Winters, T., Manshreck, T., & Wright, H. 2020. *Software engineering at google: Lessons learned from programming over time*. O'Reilly Media.

BAB 2
MANAJEMEN PROYEK PERANGKAT LUNAK

Oleh Nono Heryana

2.1 Pendahuluan

Manajemen proyek perangkat lunak merupakan bidang yang fokus pada pengaturan dan pengendalian proses dalam pengembangan perangkat lunak. Pentingnya manajemen proyek perangkat lunak tidak bisa diabaikan, mengingat perangkat lunak memegang peran penting dalam berbagai aspek kehidupan sehari-hari dan industri modern.

2.1.1 Pengertian Manajemen Proyek Perangkat Lunak

Manajemen proyek perangkat lunak adalah proses merencanakan, mengorganisir, dan mengendalikan sumber daya untuk mencapai tujuan spesifik yang telah ditentukan, dalam konteks pengembangan perangkat lunak (Project Management Institute, 2018). Ini melibatkan koordinasi berbagai aspek seperti sumber daya manusia, risiko, perubahan, kualitas, dan komunikasi.

Dalam pengertian yang lebih luas, manajemen proyek perangkat lunak mencakup semua aktivitas yang diperlukan untuk merencanakan, mengawasi, dan mengendalikan proyek perangkat lunak. Proyek tersebut dapat mencakup pengembangan perangkat lunak baru, perbaikan atau modifikasi perangkat lunak yang ada, atau bahkan penyesuaian perangkat lunak yang ada untuk memenuhi kebutuhan pengguna yang spesifik.

Manajemen proyek perangkat lunak juga mencakup kegiatan memonitor dan mengendalikan progres proyek, serta

memastikan bahwa proyek berjalan sesuai jadwal dan anggaran yang telah ditentukan. Ini melibatkan koordinasi antara berbagai tim dan individu yang terlibat dalam proyek, serta pengambilan keputusan tentang bagaimana menggunakan sumber daya yang tersedia dengan cara yang paling efektif dan efisien.

Berikut adalah gambaran umum tentang manajemen proyek perangkat lunak:

Gambar 2.1. Manajemen Proyek Perangkat Lunak

Dalam konteks ini, penting untuk memahami bahwa manajemen proyek perangkat lunak bukan hanya tentang pengkodean. Sebaliknya, itu mencakup banyak aspek lain dari

proyek, seperti perencanaan, analisis kebutuhan, desain, implementasi, pengujian, dan pemeliharaan.

2.1.2 Sejarah dan Perkembangan Manajemen Proyek Perangkat Lunak

Manajemen proyek perangkat lunak telah berkembang pesat sejak awal tahun 1960-an. Perubahan signifikan pertama datang dengan pengembangan model Waterfall oleh Winston W. Royce pada tahun 1970. Model ini merintis jalan bagi pendekatan sistematis terhadap pengembangan perangkat lunak dan menjadi fondasi dari sejumlah metodologi manajemen proyek yang ada saat ini. Bagaimanapun, model ini sering dikritik karena kurangnya fleksibilitas dan adaptabilitas terhadap perubahan dalam kebutuhan proyek (Royce, 1987).

Seiring waktu, perubahan dalam lingkungan bisnis dan teknologi membawa tantangan baru bagi manajemen proyek perangkat lunak. Pada akhir 1980-an dan awal 1990-an, pendekatan baru seperti *prototyping* dan *iterative development* mulai populer. Pendekatan ini memungkinkan tim proyek untuk merespon perubahan kebutuhan dan tantangan secara lebih efektif (Boehm, 1988).

Pada tahun 2001, Manifes Agile diperkenalkan, mengubah paradigma dalam manajemen proyek perangkat lunak. Agile memberikan lebih banyak fleksibilitas dan menekankan pada kolaborasi dan interaksi individu dibandingkan dengan proses dan alat (Beck *et al.,* 2001). Saat ini, pendekatan Agile menjadi sangat populer dan digunakan dalam berbagai proyek perangkat lunak.

Kemajuan dalam teknologi juga berdampak pada manajemen proyek perangkat lunak. Alat-alat modern seperti sistem pelacakan isu, papan kanban digital, dan perangkat lunak manajemen proyek membantu dalam meningkatkan efisiensi dan produktivitas dalam proyek perangkat lunak.

2.2 Prinsip-prinsip Manajemen Proyek

Manajemen proyek perangkat lunak mengikuti prinsip-prinsip umum manajemen proyek yang telah dikenal luas dalam berbagai industri. Prinsip-prinsip ini meliputi perencanaan, pengorganisasian, pengendalian, dan evaluasi.

2.2.1 Perencanaan

Perencanaan merupakan langkah awal dan fundamental dalam manajemen proyek perangkat lunak. Fase ini bertujuan untuk menentukan visi dan tujuan proyek, mengidentifikasi tugas-tugas dan aktivitas yang perlu dilakukan, merancang jadwal dan anggaran, serta menentukan sumber daya yang diperlukan (Dennis *et al.,* 2015).

Visi dan tujuan proyek didefinisikan berdasarkan kebutuhan stakeholder, baik internal maupun eksternal. Di sinilah peranan rekayasa kebutuhan berperan untuk mengumpulkan, menganalisis, dan menyusun persyaratan perangkat lunak (Sommerville, 2016). Tujuan proyek harus jelas dan dapat diukur, biasanya didefinisikan dengan metrik yang spesifik, seperti kapabilitas fungsional, kinerja, dan waktu peluncuran (Standish Group, 2013).

Berikutnya, rencana kerja rinci diatur dalam bentuk *Work Breakdown Structure* (WBS), yang mencakup semua aktivitas yang harus dilakukan dan waktu yang diperlukan untuk menyelesaikannya. WBS juga mencakup estimasi sumber daya yang diperlukan, seperti jumlah developer, tester, dan manajer proyek, serta perangkat keras dan perangkat lunak yang diperlukan.

Gambar 2.2. Diagram Work Breakdown Structure

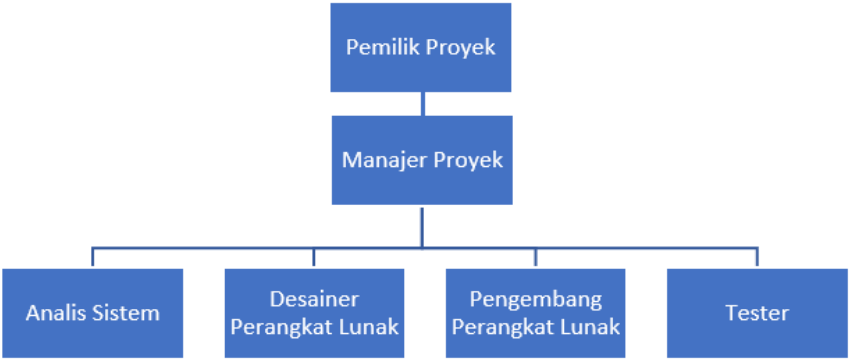
Menetapkan jadwal dan anggaran adalah bagian penting dari fase perencanaan. Jadwal mencakup durasi masing-masing aktivitas, tanggal mulai dan selesai, dan ketergantungan antaraktivitas. Anggaran mencakup biaya pengembangan, termasuk biaya tenaga kerja, perangkat keras dan perangkat lunak, dan biaya overhead lainnya. Estimasi waktu dan biaya biasanya dilakukan dengan teknik seperti *Function Point Analysis* atau COCOMO (Htay dkk., 2019).

Akhirnya, hasil dari fase perencanaan ini biasanya disusun dalam dokumen formal yang dikenal sebagai rencana proyek perangkat lunak, yang berfungsi sebagai "*blueprint*" bagi semua aktivitas selanjutnya dalam siklus hidup proyek.

2.2.2 Pengorganisasian

Pengorganisasian merupakan langkah penting dalam manajemen proyek perangkat lunak. Pengorganisasian merujuk kepada penetapan tugas dan tanggung jawab kepada anggota tim proyek, dengan tujuan untuk mencapai sasaran proyek secara efisien dan efektif (Larson & Gray, 2018).

Pengorganisasian juga mencakup pembentukan struktur organisasi proyek, yang menentukan bagaimana anggota tim berinteraksi satu sama lain dan bagaimana alur komunikasi dalam tim. Struktur organisasi proyek biasanya digambarkan dalam bentuk diagram organisasi (organogram) yang menunjukkan hierarki dan hubungan antara berbagai posisi atau peran dalam tim. Berikut ini adalah contoh sederhana dari struktur organisasi proyek:



Gambar 2.3. Contoh Diagram Organisasi Proyek Perangkat Lunak

Dalam menentukan struktur organisasi proyek, manajer proyek harus mempertimbangkan beberapa faktor, seperti ukuran dan kompleksitas proyek, keterampilan dan keahlian anggota tim, dan budaya organisasi (Kerzner, 2017). Tabel 3.2 berikut menguraikan beberapa peran dan tanggung jawab utama dalam tim proyek perangkat lunak:

Tabel 2.1. Peran dan Tanggung Jawab dalam Tim Proyek Perangkat Lunak

Peran	Tanggung Jawab
Manajer Proyek	Mengkoordinasikan semua aspek proyek, termasuk perencanaan, pengorganisasian, dan pengendalian aktivitas proyek.
Analis Sistem	Memahami dan mendokumentasikan kebutuhan pengguna, serta membantu dalam merancang solusi.
Desainer Perangkat Lunak	Membuat desain detail dari perangkat lunak berdasarkan kebutuhan yang telah dianalisis.
Pengembang Perangkat Lunak	Menulis kode dan mengimplementasikan desain perangkat lunak.
Tester	Melakukan pengujian untuk memastikan bahwa perangkat lunak bekerja seperti yang diharapkan dan bebas dari bug.

Akhirnya, pengorganisasian juga melibatkan pengaturan dan pengalokasian sumber daya proyek, seperti anggaran, peralatan, dan waktu. Manajer proyek harus memastikan bahwa sumber daya ini digunakan secara efisien dan efektif untuk mencapai sasaran proyek.

2.2.3 Pengendalian

Pengendalian merupakan aspek penting dalam manajemen proyek perangkat lunak yang merujuk pada serangkaian kegiatan untuk memastikan proyek berjalan sesuai dengan rencana yang telah ditetapkan. Dalam konteks proyek perangkat lunak, pengendalian mencakup pemantauan kemajuan proyek, deteksi perubahan dari rencana awal, dan

pengambilan tindakan korektif yang diperlukan untuk menyesuaikan keadaan yang sebenarnya dengan keadaan yang direncanakan (Schwalbe, 2015).

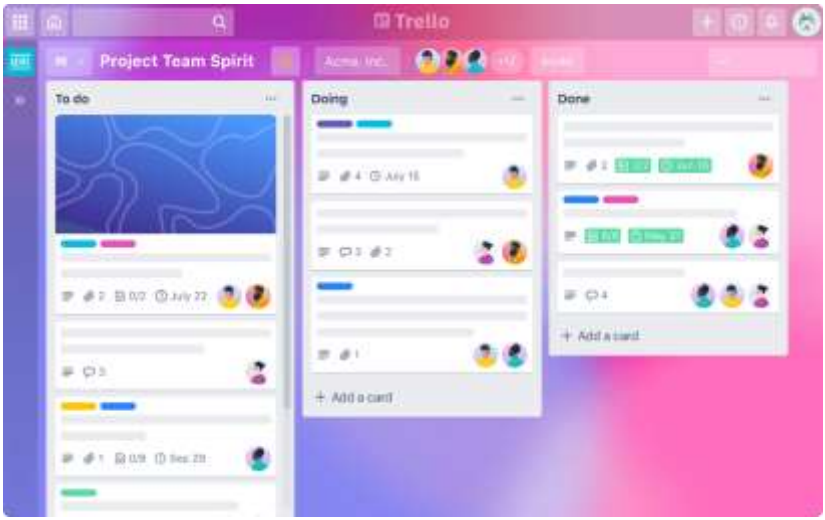
Salah satu aspek penting dari pengendalian adalah pemantauan. Pemantauan berfungsi untuk mengumpulkan data tentang kemajuan proyek dan menilai apakah proyek berjalan sesuai dengan rencana atau tidak. Metode pemantauan dapat berupa pertemuan reguler dengan tim proyek, laporan kemajuan, atau sistem pelaporan otomatis.

Tabel 2.2. Contoh format laporan kemajuan proyek

No	Kegiatan	Rencana	Realisasi	Selisih	Penyebab	Tindakan Korektif
1	Membuat modul login	2 hari	3 hari	1 hari	Bug pada sistem	Mengerahkan lebih banyak developer
2	Pengujian sistem	5 hari	5 hari	0 hari	-	-
3	Pelatihan pengguna	3 hari	2 hari	-1 hari	Pengguna cepat memahami	-

Setelah data dikumpulkan, tahap berikutnya adalah analisis dan evaluasi. Ini melibatkan perbandingan antara status proyek yang sebenarnya dengan apa yang direncanakan. Bila terdapat penyimpangan, penyebabnya perlu diidentifikasi. Akhirnya, tindakan korektif diperlukan untuk menyesuaikan kondisi proyek saat ini dengan rencana. Tindakan ini bisa berupa perubahan jadwal, pengalokasian ulang sumber daya, atau perubahan pada spesifikasi produk.

Pengendalian proyek perangkat lunak dapat dilakukan dengan menggunakan berbagai perangkat lunak manajemen proyek, seperti Microsoft Project, Jira, atau Trello, yang memungkinkan manajer proyek untuk melacak kemajuan proyek dan mengidentifikasi masalah secepat mungkin.



Gambar 2.4. Trello

Dengan manajemen pengendalian yang efektif, manajer proyek dapat memastikan bahwa proyek berjalan sesuai rencana, dan dapat segera merespons jika terjadi masalah atau perubahan yang tidak terduga.

2.2.4 Evaluasi

Evaluasi merupakan tahap penting dalam siklus manajemen proyek perangkat lunak, berfungsi untuk menilai efektivitas dan efisiensi proyek, dan identifikasi area yang memerlukan peningkatan (Kerzner, 2017). Evaluasi ini biasanya melibatkan penilaian terhadap sejumlah aspek kunci proyek, termasuk pengiriman waktu, kualitas hasil kerja, penggunaan sumber daya, dan kepatuhan terhadap anggaran.

Pada tahap evaluasi, biasanya dilakukan pemantauan terhadap beberapa indikator utama kinerja proyek (*Key Performance Indicators,* KPI) (Heldman, 2018). Indikator-indikator ini diukur secara periodik sepanjang durasi proyek untuk memantau kemajuan dan kinerja proyek.

Salah satu cara untuk mengevaluasi proyek adalah dengan menggunakan teknik *Earned Value Management* (EVM). EVM adalah metodologi yang menggabungkan pengukuran biaya proyek, jadwal, dan lingkup untuk menentukan apakah proyek berjalan sesuai dengan rencana (Fleming & Koppelman, 2016). Dengan metode ini, manajer proyek dapat dengan cepat mengidentifikasi masalah yang mungkin muncul dan mengambil tindakan korektif yang diperlukan.

Sebagai contoh, tabel berikut menyajikan contoh penggunaan EVM dalam proyek perangkat lunak:

Tabel 2.3. Contoh Penggunaan EVM dalam Proyek Perangkat Lunak

Periode	Biaya Diperkirakan	Biaya Aktual	Nilai Hasil
1	Rp 100.000.000	Rp 90.000.000	Rp 80.000.000
2	Rp 200.000.000	Rp 220.000.000	Rp 180.000.000
3	Rp 300.000.000	Rp 330.000.000	Rp 280.000.000

Begitu evaluasi selesai, hasilnya harus disampaikan kepada semua pemangku kepentingan proyek. Ini penting untuk memastikan bahwa semua pihak yang berkepentingan memahami status dan kemajuan proyek. Selain itu, evaluasi juga dapat digunakan sebagai dasar untuk perencanaan proyek masa depan, dengan menggunakan pelajaran yang telah dipelajari dari proyek saat ini untuk meningkatkan efisiensi dan efektivitas proyek di masa mendatang.

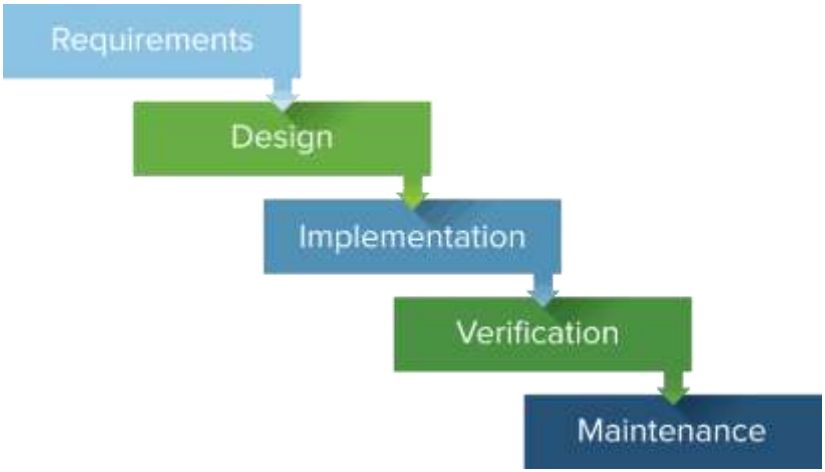
2.3 Metodologi Manajemen Proyek

Metodologi manajemen proyek mencakup serangkaian prinsip dan proses yang digunakan oleh tim proyek untuk mencapai tujuan dan mencapai hasil yang sukses. Metodologi ini membantu mengarahkan usaha tim dan mempengaruhi keputusan selama siklus hidup proyek (Wysocki, 2011). Empat

metodologi utama yang akan kita bahas adalah Waterfall, Agile, Scrum, dan Lean.

2.3.1 Waterfall

Model Waterfall adalah salah satu metodologi manajemen proyek paling awal dan paling sederhana. Model ini berfungsi dengan cara mengorganisir aktivitas proyek dalam urutan linear, di mana setiap fase harus selesai sebelum fase berikutnya dapat dimulai (Royce, 1970). Fasilitas ini mencakup spesifikasi kebutuhan, desain, implementasi, pengujian, pemasangan, dan pemeliharaan.



Gambar 2.5. Model Waterfall

Namun, metode ini memiliki beberapa keterbatasan, seperti kurangnya fleksibilitas untuk mengubah persyaratan setelah proyek dimulai dan potensi penundaan jika fase tertentu menemui hambatan.

2.3.2 Agile

Sebagai respons terhadap keterbatasan model Waterfall, metodologi Agile dikembangkan. Agile mempromosikan

pendekatan yang lebih iteratif dan inkremental untuk manajemen proyek, di mana persyaratan dan solusi berkembang melalui kolaborasi antara tim yang berfungsi (Highsmith & Cockburn, 2001).

Gambar 2.6. Metodologi Agile

Metodologi Agile memberi kesempatan untuk merevisi dan memperbaiki proyek di setiap tahapan, memberikan fleksibilitas yang lebih besar dan adaptasi terhadap perubahan.

2.3.3 Scrum

Scrum adalah kerangka kerja Agile yang berfokus pada pengiriman nilai yang maksimal dalam waktu minimum. Metodologi ini melibatkan pemisahan proyek menjadi 'sprint' atau periode waktu tertentu (biasanya 2-4 minggu), di mana tim memilih dan menyelesaikan serangkaian tugas yang paling penting (Schwaber & Sutherland, 2017).

Tabel 2.4. Contoh Struktur Timeline dalam Scrum

No	Aktivitas	Durasi
1	Perencanaan Sprint (*Sprint Planning*)	2-4 jam
2	Sprint	2-4 minggu
3	Pemeriksaan Harian (*Daily Scrum*)	15 menit per hari
4	Review Sprint	2-4 jam pada akhir Sprint
5	Retrospektif Sprint (*Sprint Retrospective*)	1,5-3 jam pada akhir Sprint

Tim kemudian mereview kinerja mereka dan merencanakan sprint berikutnya, membuat iterasi berkelanjutan dan peningkatan kinerja.

2.3.4 Lean

Lean adalah metodologi yang berfokus pada eliminasi pemborosan dan peningkatan efisiensi. Ini awalnya dikembangkan dalam konteks manufaktur oleh Toyota dan sejak itu telah diadaptasi untuk digunakan dalam manajemen proyek, termasuk pengembangan perangkat lunak (Womack & Jones, 1996).

Gambar 2.7. Metodologi Lean

Lean menekankan perbaikan proses, penghilangan pemborosan, dan peningkatan aliran kerja untuk mencapai pengiriman yang lebih cepat dan efisien.

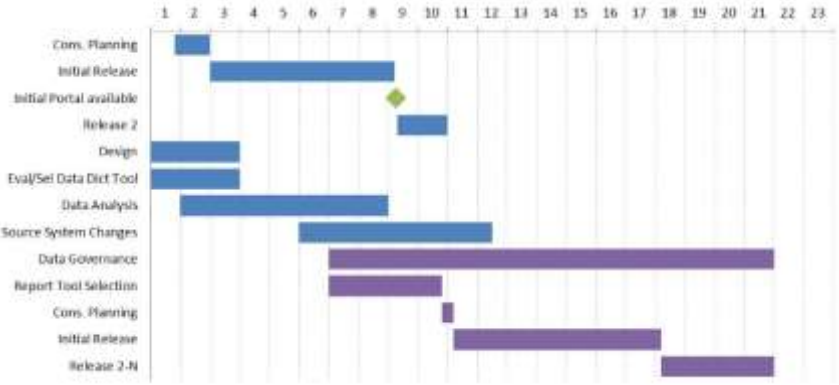
2.4 Alat dan Teknik Manajemen Proyek

Manajemen proyek perangkat lunak memerlukan penggunaan alat dan teknik khusus yang dirancang untuk membantu pengelolaan proyek, memonitor kemajuan, dan memastikan pemenuhan tujuan secara tepat waktu dan efisien.

2.4.1 Gantt Chart

Gantt Chart, yang dinamai dari penemu pertamanya Henry Gantt, adalah alat visualisasi yang digunakan untuk merencanakan dan menjadwalkan proyek. Dalam Gantt Chart, kegiatan atau tugas ditempatkan di sepanjang sumbu vertikal, sementara sumbu horizontal mewakili rentang waktu proyek.

Bar horizontal menunjukkan durasi masing-masing tugas (Lock, 2007).

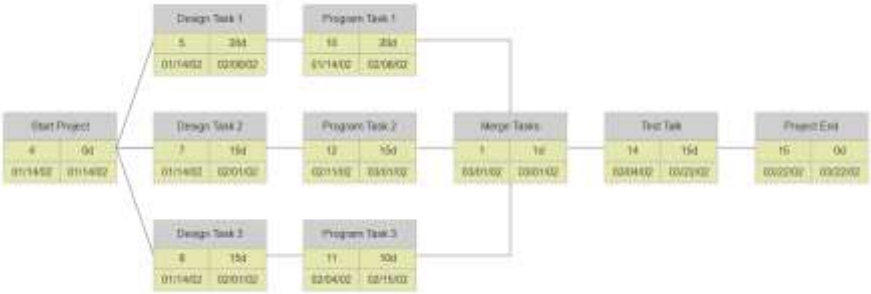


Gambar 2.8. Gantt Chart

Diagram Gantt membantu manajer proyek dan tim untuk melihat bagaimana tugas saling terkait, kapan masing-masing tugas harus dimulai dan selesai, dan ketergantungan antara tugas-tugas tersebut. Gantt Chart juga memfasilitasi identifikasi bottleneck atau penundaan dalam proyek, yang memungkinkan perubahan jadwal yang proaktif untuk mempertahankan alur kerja yang efisien (Kerzner, 2017).

2.4.2 Pert Chart

Program Evaluation and Review Technique (PERT) Chart adalah alat lain yang digunakan dalam manajemen proyek untuk menggambarkan urutan tugas dan hubungan antara tugas-tugas tersebut. PERT Chart digunakan untuk analisis jalur kritis dalam proyek, yang membantu dalam identifikasi tugas mana yang harus diutamakan dan bagaimana penundaan dapat mempengaruhi proyek secara keseluruhan.



Gambar 2.9. Pert Chart

2.4.3 Teknik Estimasi Biaya dan Waktu

Estimasi biaya dan waktu adalah elemen penting dalam manajemen proyek. Metode-metode yang umum digunakan meliputi metode top-down, bottom-up, dan teknik dekomposisi kerja. Metode top-down melibatkan estimasi biaya dan waktu berdasarkan pengalaman proyek sebelumnya atau pengetahuan pakar. Sementara itu, metode bottom-up melibatkan penjabaran detail tugas dan sub-tugas, lalu mengestimasi biaya dan waktu untuk masing-masing bagian tersebut. Teknik dekomposisi kerja biasanya digunakan bersamaan dengan pendekatan bottom-up untuk membantu dalam estimasi.

Tabel 2.5. Teknik Estimasi Biaya dan Waktu

Teknik	Deskripsi
Top-Down	Estimasi dilakukan berdasarkan pengalaman sebelumnya atau pengetahuan pakar
Bottom-Up	Melibatkan penjabaran detail tugas dan sub-tugas dan estimasi untuk masing-masing
Dekomposisi Kerja	Biasanya digunakan bersamaan dengan pendekatan bottom-up

2.5 Manajemen Risiko dalam Proyek Perangkat Lunak

Manajemen risiko merupakan elemen penting dalam manajemen proyek perangkat lunak. Risiko diartikan sebagai peristiwa atau kondisi yang tidak pasti yang, jika terjadi, memiliki efek pada setidaknya satu tujuan proyek seperti lingkup, jadwal, biaya dan kualitas (Project Management Institute, 2018). Dalam konteks proyek perangkat lunak, risiko dapat mencakup berbagai aspek seperti perubahan kebutuhan, teknologi yang belum teruji, atau sumber daya yang tidak mencukupi.

2.5.1 Pengenalan Risiko

Pengenalan risiko adalah langkah pertama dalam proses manajemen risiko. Ini melibatkan identifikasi dan penilaian faktor-faktor yang dapat menyebabkan proyek gagal mencapai tujuannya. Pengenalan risiko dapat dilakukan melalui berbagai metode seperti analisis checklist, analisis brainstorming, atau wawancara dengan stakeholder (Kerzner, 2017).

2.5.2 Analisis Risiko

Setelah risiko diidentifikasi, langkah selanjutnya adalah analisis risiko. Analisis ini dilakukan untuk mengevaluasi dampak potensial dan kemungkinan terjadinya setiap risiko. Dampak dan probabilitas kemudian digunakan untuk menghitung nilai risiko, yang dapat digunakan untuk menentukan prioritas penanganan risiko (Boehm, 1991).

Tabel 2.6. Contoh Tabel Analisis Risiko

Risiko	Dampak	Probabilitas	Nilai Risiko
Perubahan Kebutuhan	Tinggi	Sedang	Tinggi
Teknologi yang belum teruji	Sedang	Rendah	Sedang
Sumber daya yang tidak mencukupi	Tinggi	Tinggi	Sangat Tinggi

2.5.3 Mitigasi Risiko

Mitigasi risiko melibatkan pengembangan rencana dan tindakan untuk mengurangi dampak atau kemungkinan dari risiko yang telah diidentifikasi. Strategi mitigasi risiko bisa berupa penerimaan risiko (jika risiko memiliki dampak rendah atau probabilitas rendah), transfer risiko (misalnya melalui asuransi), penghindaran risiko (mengubah rencana proyek untuk menghilangkan risiko), atau penurunan risiko (melakukan tindakan untuk mengurangi probabilitas atau dampak risiko) (Schwalbe, 2015).

2.6 Manajemen Sumber Daya Manusia dalam Proyek Perangkat Lunak

Manajemen sumber daya manusia adalah aspek penting dalam proyek perangkat lunak. Dalam konteks ini, sumber daya manusia merujuk pada tim pengembang perangkat lunak,

termasuk perancang, programmer, tester, dan berbagai peran lainnya yang terlibat dalam pengembangan proyek. Manajemen yang efektif dari sumber daya manusia ini dapat menjadi kunci keberhasilan proyek.

2.6.1 Pengelolaan Tim

Pengelolaan tim mencakup pengaturan peran dan tanggung jawab tim, membangun dan mempertahankan dinamika tim yang positif, serta mendorong kolaborasi dan komunikasi yang efektif. Pembentukan tim yang efektif biasanya melibatkan pemilihan individu yang memiliki keterampilan dan keahlian yang diperlukan, serta kemampuan untuk bekerja dengan baik dalam tim. Diagram seperti tabel organisasi bisa membantu dalam menentukan struktur tim dan tugas masing-masing anggota (Belbin, 2012).

Selain itu, manajer proyek harus bekerja untuk memastikan bahwa timnya tetap termotivasi dan fokus pada tujuan proyek. Hal ini mungkin melibatkan penerapan teknik manajemen seperti pemberian umpan balik positif dan konstruktif, pengakuan atas prestasi, dan pembinaan individu yang merasa kesulitan atau terbebani.

2.6.2 Manajemen Konflik

Dalam setiap proyek, terutama yang melibatkan banyak orang dengan berbagai latar belakang dan keahlian, konflik hampir tidak dapat dihindari. Manajemen konflik melibatkan identifikasi dan penyelesaian konflik secara konstruktif dan tepat waktu, sebelum mereka berkembang menjadi masalah yang lebih besar. Metode penyelesaian konflik dapat mencakup mediasi, negosiasi, atau bahkan arbitrasi dalam beberapa kasus (Rahim, 2017).

Manajemen konflik menjadi aspek penting dalam manajemen sumber daya manusia dalam proyek pengembangan perangkat lunak. Konflik dapat muncul dari berbagai sumber dan memerlukan identifikasi dan intervensi

yang tepat. Proses manajemen konflik mencakup tahapan identifikasi, analisis, pembahasan, penyusunan solusi, penyelesaian, dan evaluasi. Identifikasi melibatkan deteksi dini konflik dan pemahaman sumber masalah, sedangkan analisis memfokuskan pada dampak konflik terhadap kinerja tim dan progres proyek. Pembahasan memungkinkan pihak-pihak yang terlibat untuk menyampaikan perspektif mereka, dan penyusunan solusi melibatkan pengembangan dan evaluasi opsi penyelesaian. Setelah solusi diterapkan, manajer proyek harus memantau proses tersebut dan mengevaluasi efektivitas solusi. Jika perlu, penyesuaian dapat dilakukan untuk menyelesaikan konflik dengan lebih efektif.

Tabel 2.7. Proses Penyelesaian Konflik

Langkah Proses Penyelesaian Konflik	**Keterangan**
1. Identifikasi Konflik	Menentukan sumber dan jenis konflik yang terjadi.
2. Pemahaman Konflik	Melakukan analisis yang mendalam tentang isu yang mendasari konflik.
3. Pembentukan Strategi	Memilih metode penyelesaian konflik yang paling tepat berdasarkan jenis dan tingkat konflik.
4. Implementasi Solusi	Melakukan tindakan untuk menyelesaikan konflik berdasarkan strategi yang dipilih.
5. Evaluasi Hasil	Mengevaluasi efektivitas solusi dan membuat penyesuaian jika

2.6.3 Pemimpin dan Gaya Kepemimpinan

Pemimpin proyek memainkan peran penting dalam membentuk kultur dan produktivitas tim. Gaya kepemimpinan yang mereka pilih dapat berdampak signifikan pada dinamika tim dan hasil akhir proyek. Gaya kepemimpinan yang berbeda dapat lebih efektif dalam situasi yang berbeda, dan pemimpin yang baik harus mampu beradaptasi gaya mereka sesuai dengan kebutuhan tim dan proyek. Pemimpin proyek perangkat lunak mungkin memilih untuk menerapkan berbagai gaya kepemimpinan, tergantung pada situasi dan kebutuhan tim (Bass & Riggio, 2006).. Beberapa contoh gaya kepemimpinan yang umum digunakan adalah:

1. Kepemimpinan Transaksional: Pemimpin transaksional berfokus pada tugas dan struktur, dan cenderung menghargai pertukaran antara pemimpin dan anggota tim. Mereka mendefinisikan tugas dan tujuan, dan memberi penghargaan atau hukuman berdasarkan kinerja.
2. Kepemimpinan Transformasional: Pemimpin transformasional berusaha untuk menginspirasi dan memotivasi anggota tim untuk melampaui tujuan yang telah ditetapkan. Mereka biasanya mendorong inovasi, dan mempromosikan nilai-nilai, etika, standar, dan visi jangka panjang.
3. Kepemimpinan Situasional: Gaya kepemimpinan ini melibatkan adaptasi dan perubahan gaya kepemimpinan berdasarkan situasi atau konteks tertentu. Pemimpin situasional memahami bahwa tidak ada satu gaya kepemimpinan yang cocok untuk setiap situasi dan oleh karena itu, mereka harus fleksibel dan mampu mengubah gaya mereka sesuai dengan perubahan kondisi.

Pemilihan gaya kepemimpinan yang tepat memerlukan pemahaman yang mendalam tentang sifat proyek, karakteristik tim, dan lingkungan kerja. Selain itu, pemimpin harus mampu

menyeimbangkan tuntutan proyek dengan kebutuhan dan kesejahteraan anggota tim.

Harap diingat bahwa gaya kepemimpinan tidak selalu kaku dan dapat berubah berdasarkan situasi dan kebutuhan individu atau tim. Pemimpin yang efektif biasanya menggabungkan berbagai gaya kepemimpinan sesuai kebutuhan.

2.7 Manajemen Komunikasi dalam Proyek Perangkat Lunak

Manajemen komunikasi merujuk pada koordinasi, pengumpulan, distribusi, dan penyimpanan informasi dalam sebuah proyek. Dalam konteks pengembangan perangkat lunak, manajemen komunikasi memiliki peran penting untuk memastikan seluruh tim memahami tujuan dan sasaran proyek, serta tanggung jawab dan tugas masing-masing. Faktor penting lainnya adalah pemahaman terhadap risiko, milestone, perkembangan, dan masalah yang mungkin muncul.

1. **Komunikasi Internal**
   Komunikasi internal melibatkan interaksi antar anggota tim proyek perangkat lunak. Hal ini mencakup pertemuan reguler, diskusi, dan alat komunikasi digital seperti email, sistem manajemen proyek, dan lain-lain. Penting bagi manajer proyek untuk memastikan bahwa komunikasi internal berjalan efektif untuk mendukung kerja sama tim dan mengurangi kesalahpahaman.
2. **Komunikasi Eksternal**
   Komunikasi eksternal mencakup pertukaran informasi antara tim proyek dengan pihak eksternal seperti pemangku kepentingan, klien, pengguna, atau departemen lain dalam organisasi. Komunikasi ini dapat berupa laporan perkembangan proyek, permintaan perubahan, atau isu-isu yang mempengaruhi proyek. Manajer proyek harus

memastikan bahwa komunikasi eksternal ini ditangani dengan cara yang tepat dan profesional, serta mencatat dalam bentuk dokumentasi yang rapi (Roger and Bruce, 2015).

3. **Teknologi dalam Komunikasi Proyek**
   Teknologi memainkan peran krusial dalam memfasilitasi komunikasi baik internal maupun eksternal dalam proyek perangkat lunak. Alat komunikasi digital seperti email, perangkat lunak manajemen proyek (misalnya, Jira, Trello), aplikasi pesan instan (seperti *Slack, Microsoft Teams*), dan platform video konferensi (seperti *Zoom, Google Meet*) semakin populer. Selain itu, penggunaan teknologi juga memungkinkan untuk menyimpan, mengakses, dan berbagi dokumen serta informasi proyek dengan mudah dan aman (PMBOK, 2018).

2.8 Evaluasi dan Penutupan Proyek

Evaluasi dan penutupan proyek adalah tahap akhir dari siklus hidup manajemen proyek. Tujuannya adalah untuk mengevaluasi kinerja keseluruhan proyek, merangkum hasil dan pelajaran yang diperoleh, serta merumuskan laporan akhir. Penyelesaian ini membantu organisasi dalam memahami sejauh mana proyek berhasil dalam mencapai tujuan yang telah ditetapkan dan dalam hal mana proyek tersebut bisa diperbaiki di masa mendatang (Kerzner, 2017).

2.8.1 Evaluasi Kinerja Proyek

Evaluasi kinerja proyek adalah proses yang sistematis dan objektif untuk menilai sejauh mana proyek telah mencapai tujuan dan sasarannya. Evaluasi ini umumnya dilakukan setelah penyelesaian proyek, tetapi bisa juga dilakukan selama proyek berlangsung untuk memantau progres dan melakukan koreksi bila diperlukan (Project Management Institute, 2017).

Aspek-aspek yang dievaluasi biasanya mencakup kualitas hasil yang dihasilkan, efisiensi penggunaan sumber daya, kepatuhan terhadap jadwal, dan kepuasan stakeholder. Proses evaluasi ini melibatkan pengumpulan data yang relevan, analisis data tersebut, dan interpretasi hasil analisis untuk membuat penilaian terhadap kinerja proyek (Heldman, 2018).

Metode yang umum digunakan dalam evaluasi kinerja proyek adalah *Post-Implementation Review* (PIR). PIR adalah penilaian formal yang dilakukan setelah proyek selesai untuk menilai sejauh mana proyek telah memenuhi tujuan dan sasarannya, termasuk aspek kualitas, biaya, dan waktu. PIR juga mencakup analisis tentang apa yang berhasil dan apa yang tidak, sehingga organisasi dapat belajar dari pengalaman tersebut untuk proyek masa mendatang (Kerzner, 2017).

Berikut adalah alur proses Post-Implementation Review (PIR):

1. Identifikasi Tujuan dan Sasaran Proyek: Menetapkan secara jelas apa yang ingin dicapai oleh proyek.
2. Pengumpulan Data: Mengumpulkan data yang relevan tentang proyek, seperti dokumentasi proyek, data kinerja, dan umpan balik dari stakeholder.
3. Analisis Data: Menganalisis data yang telah dikumpulkan untuk menilai kinerja proyek.
4. Penilaian: Membuat penilaian berdasarkan hasil analisis data.
5. Penyusunan Laporan: Menyusun laporan yang merangkum hasil evaluasi.

2.8.2 Laporan Akhir Proyek

Laporan akhir proyek adalah dokumen yang secara komprehensif merangkum semua aktivitas dan hasil dari suatu proyek. Dokumen ini disusun pada akhir proyek sebagai bagian dari tahap penutupan dan berfungsi sebagai catatan resmi tentang apa yang telah dicapai oleh proyek tersebut. Laporan ini dapat dijadikan sebagai rujukan dalam pengembangan

proyek masa depan dan sebagai sarana untuk berkomunikasi dengan pihak yang berkepentingan (*stakeholders*) tentang hasil proyek (Wysocki, 2011).

Struktur dan isi dari laporan akhir proyek mungkin berbeda-beda tergantung pada kebutuhan spesifik dari organisasi dan stakeholders. Akan tetapi, berikut adalah beberapa elemen yang umumnya termasuk dalam laporan akhir proyek:

1. Ringkasan Eksekutif: Bagian ini memberikan ikhtisar tentang proyek, termasuk tujuan, metode, hasil, dan kesimpulan utama.
2. Tujuan dan Sasaran: Bagian ini menjelaskan tujuan dan sasaran proyek secara lebih detail, dan mencakup penilaian tentang sejauh mana tujuan dan sasaran tersebut telah dicapai.
3. Metodologi: Bagian ini menjelaskan metode dan teknik yang digunakan dalam proyek, termasuk alat, teknologi, dan prosedur yang digunakan.
4. Jadwal dan Biaya: Bagian ini memberikan detail tentang jadwal proyek dan bagaimana biaya proyek dikelola.
5. Hasil: Bagian ini menjelaskan hasil yang dicapai oleh proyek, baik itu produk, layanan, atau hasil lainnya.
6. Evaluasi: Bagian ini mencakup evaluasi kinerja proyek, termasuk penilaian kualitas, efisiensi, dan dampak proyek.
7. Pelajaran yang Diperoleh: Bagian ini mencakup analisis tentang apa yang berhasil dan apa yang tidak dalam proyek, dan bagaimana pelajaran dari proyek ini dapat diterapkan pada proyek masa depan.

Tabel 2.8. berikut ini merangkum struktur dari laporan akhir proyek.

Tabel 2.8. Contoh struktur laporan akhir proyek

No	Bagian	Keterangan
1	Ringkasan Eksekutif	Deskripsi singkat tentang proyek dan hasilnya
2	Tujuan dan Sasaran	Detail tentang tujuan dan sasaran proyek
3	Metodologi	Penjelasan tentang teknik dan alat yang digunakan
4	Jadwal dan Biaya	Detail tentang jadwal proyek dan penggunaan anggaran
5	Hasil	Deskripsi tentang hasil proyek
6	Evaluasi	Diskusi tentang kinerja proyek
7	Pelajaran yang Diperoleh	Analisis tentang apa yang berhasil dan apa yang tidak

DAFTAR PUSTAKA

Bass, B.M. and Riggio, R.E., 2006. Transformational leadership.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. and Kern, J., 2001. Manifesto for agile software development.

Belbin, R.M., 2012. Team roles at work. Routledge.

Boehm, B.W., 1988. A spiral model of software development and enhancement. Computer, 21(5), pp.61-72.

Boehm, B.W., 2007. Software engineering economics. Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research, 69, p.117.

Dennis, A., Wixom, B. and Tegarden, D., 2015. Systems analysis and design: An object-oriented approach with UML. John wiley & sons.

Fleming, Q.W. and Koppelman, J.M., 2016, December. Earned value project management. Project Management Institute.

Heldman, K., 2018. PMP: project management professional exam study guide. John Wiley & Sons.

Highsmith, J. and Cockburn, A., 2001. Agile software development: The business of innovation. Computer, 34(9), pp.120-127.

Htay, K., Aung, M.M., Cho, Y.Y. and Thein, M.M., 2019. Software Engineering Cost Estimation using COCOMO II Model.

Hughes, B., 2009. Software Project Management 5e. McGraw Hill.

Kerzner, H., 2017. Project management: a systems approach to planning, scheduling, and controlling. John Wiley & Sons.

Larson, E.W. and Gray, C.F., 2018. Project Management: The Managerial Process. Language, 22(659p), p.25cm.

Lock, D., 2007. Project management 9th edition. Burlington, VT: Gower.

Project Management Institute. 2018. A guide to the project management body of knowledge. Project Management Institute. Pennsylvania.

Rahim, M.A., 2017. Managing diverse situations in organizations. In Diversity, Conflict, and Leadership (pp. 1-10). Routledge.

Roger, S.P. and Bruce, R.M., 2015. Software engineering: a practitioner's approach. McGraw-Hill Education.

Royce, W.W., 1987, March. Managing the development of large software systems: concepts and techniques. In Proceedings of the 9th international conference on Software Engineering (pp. 328-338).

Royce, W.W., 1987, March. Managing the development of large software systems: concepts and techniques. In Proceedings of the 9th international conference on Software Engineering (pp. 328-338).

Schwaber, K. and Sutherland, J., 2017. The Scrum Guide TM the definitive guide to Scrum: The rules of the game. Scrum Alliance.

Schwalbe, K., 2015. Information technology project management. Cengage Learning.

Sommerville, I., 2016. Software Engineering, 10h Ed., Harlow.

Standish Group, 2013. Chaos manifesto 2013. The Standish Group International.

Ward, S. and Chapman, C., 2011. How to manage project opportunity and risk: Why uncertainty management can be a much better approach than risk management. John Wiley & Sons.

Womack, J.P. and Jones, D.T., 1997. Lean thinking—banish waste and create wealth in your corporation. Journal of the Operational Research Society, 48(11), pp.1148-1148.

Wysocki, R.K., 2011. Effective project management: traditional, agile, extreme. John Wiley & Sons.

BAB 3
PENGEMBANGAN PERANGKAT LUNAK AGILE

Oleh Johni S Pasaribu

3.1 Pendahuluan

Metodologi pengembangan perangkat lunak Agile secara resmi diluncurkan pada tahun 2001, ketika 17 orang *software developers* (pengembang perangkat lunak) dalam komunitas rekayasa perangkat lunak menyusun Agile Manifesto melalui dua belas prinsip dan empat nilai. Kelincahan (*agility*) merupakan landasan pengembangan perangkat lunak (PL) Agile. Hal ini berbeda dengan teknik *plan-driven* dalam model konvensional sebelumnya seperti metode Waterfall. Saat ini, pengembangan PL Agile merupakan pendekatan pengembangan penting, dimana berasal dari penggunaan praktis dengan kerjasama antara pengguna dan pengembang sehingga proses pengembangan yang cepat dapat dilakukan dan beradaptasi dengan modifikasi sesuai lingkungan yang dinamis. Banyak metode PL Agile saat ini tersedia dalam literatur dengan metode Scrum dan Extreme Programming (XP) merupakan dua metode yang paling umum digunakan.

Rekayasa perangkat lunak adalah disiplin ilmu yang mengalami banyak kemajuan karena mengikuti perkembangan teknologi dan kebutuhan bisnis modern melalui pengembangan pendekatan yang efektif untuk mencapai produk akhir perangkat lunak, dimana pengembangan PL Agile adalah salah satu dari keberhasilan dalam rekayasa perangkat lunak. Pengembangan PL Agile adalah pendekatan ringan yang diusulkan untuk mengatasi keterbatasan metode

pengembangan konvensional dan untuk mengurangi waktu dan biaya sambil memberikan fleksibilitas untuk mengadopsi perubahan persyaratan pada tahap apa pun. Rekayasa perangkat lunak adalah bagian dari keluarga teknik, dimana produk perangkat lunak harus melewati semua proses analisis, desain, dan pengembangan yang sama seperti bidang teknik lainnya seperti teknik elektro, teknik mesin, dan teknik sipil. Perangkat lunak ini juga dikembangkan untuk digunakan dalam tujuan tertentu yang sama seperti bidang teknik lainnya (Braude, E. J., & Bernstein, 2011).

IEEE mendefinisikan *software engineering* (rekayasa perangkat lunak) sebagai penerapan pendekatan yang sistematis, disiplin, dan terukur untuk pengembangan, pengoperasian, dan pemeliharaan perangkat lunak; yaitu penerapan rekayasa ke dalam perangkat lunak (IEEE, 1990). Dapat dilihat dari definisi tersebut bahwa produk akhir itu penting tetapi pendekatan dan langkah-langkah yang diperlukan untuk menghasilkan perangkat lunak haruslah dilakukan secara sistematis, disiplin dan terukur pada semua siklus hidup perangkat lunak. Dengan mengikuti karakteristik pendekatan-pendekatan tersebut, perangkat lunak yang dihasilkan akan lebih dapat diandalkan, mudah dirawat dan memenuhi kebutuhan perangkat lunak, terutama ketika ukuran perangkat lunak sangat besar dan memiliki banyak fungsi (Braude, E. J., & Bernstein, 2011). Pengembangan dan rekayasa perangkat lunak adalah kerja kooperatif, tugas dapat didistribusikan ke tim yang berbeda, tugas ini harus dikelola dan diurutkan berdasarkan beberapa kriteria. Tugas dapat dilakukan secara paralel, tetapi ada beberapa tugas yang tidak dapat dimulai sebelum tugas lain berakhir. Jadi, kebutuhan koordinasi antara tugas, proses, dan tim ini adalah keharusan untuk mendapatkan perangkat lunak atau produk terbaik dengan biaya minimal (Giuffrida, R., & Dittrich, 2015). Proses pengembangan perangkat lunak didefinisikan sebagai "satu set

kegiatan yang mengarah untuk menghasilkan produk perangkat lunak" (Sommerville, 2011).

Ada banyak metode dan pendekatan pengembangan perangkat lunak tradisional, seperti pendekatan Waterfall, pendekatan iteratif dan inkremental, pendekatan spiral, pendekatan evolusioner, dan lain-lain (Mall, 2014). Pendekatan ini kadang-kadang disebut pendekatan pengembangan perangkat lunak terencana, atau pendekatan kelas berat. Pendekatan ini sangat berguna ketika ada kebutuhan untuk mengembangkan perangkat lunak yang sangat kompleks, ini dapat membantu menghilangkan pengembangan perangkat lunak informal mode lama dan memberikan perangkat lunak berkualitas tinggi dengan cara yang sistematis yang memenuhi kebutuhan pengguna dalam waktu terbatas yang telah ditentukan sebelumnya (Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, 2015).

Masalah metode pengembangan perangkat lunak tradisional membutuhkan proses yang sangat berat seperti berikut (Braude, E. J., & Bernstein, 2011):

1. Rencana proyek harus dilakukan terlebih dahulu.
2. Persyaratan (*requirements*) perangkat lunak tertulis.
3. Desain (*design*) yang sepenuhnya memenuhi persyaratan tersebut.
4. Membuat program (*coding*) perangkat lunak yang memenuhi semua persyaratan dan desain.
5. Uji perangkat lunak dan periksa apakah memenuhi persyaratan dan desain.

Banyak proyek yang mengikuti metode pengembangan perangkat lunak tradisional tersebut menghadapi masalah besar, terutama dalam pemeliharaan (*maintenance*) dan perubahan berdasarkan permintaan pengguna (*user*). Bahkan beberapa dari perubahan ini dapat menyebabkan perubahan besar yang dianggap sebagai masalah besar dalam

pengembangan perangkat lunak. Karena itu diperlukan metode pengembangan perangkat lunak yang ringan, dimana tujuan utama dari metode tersebut adalah untuk mempercepat pengembangan dan secara efektif merespon perubahan yang diminta oleh pengguna. Metode pengembangan perangkat lunak ringan ini disebut metode pengembangan perangkat lunak Agile (Braude, E. J., & Bernstein, 2011).

3.2 Model Proses Agile

Lebih cepat, lebih tinggi, lebih kuat—moto Olimpiade (bahasa Latin *Citius, Altius, Fortius*) juga bisa menjadi moto untuk pengembangan perangkat lunak saat ini. Pengembangan perangkat lunak perlu kreatif, dilakukan oleh tim dari lintas fungsi yang mengatur diri sendiri. Kolaborasi adalah kuncinya dan perangkat lunak yang berfungsi menjadi sorotan. Aspirasi ini, yang juga tercermin dalam "Agile Manifesto" (K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, 2001a), telah menjadi cita-cita yang dikejar banyak perusahaan. Pengembangan perangkat lunak Agile kini telah ada lebih dari 20 tahun, dan tidak dapat disangkal bahwa hal itu telah menghasilkan beberapa peningkatan (Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, 2012) seperti peningkatan kecepatan pengembangan perangkat lunak dan kolaborasi yang intensif antara pemangku kepentingan yang berbeda.

Arti Agile adalah cepat atau serbaguna. Model proses Agile mengacu pada pendekatan pengembangan perangkat lunak berdasarkan pengembangan iteratif. Metode Agile memecah tugas menjadi iterasi yang lebih kecil, atau bagian-bagian yang tidak secara langsung melibatkan perencanaan jangka panjang. Lingkup dan persyaratan proyek ditetapkan pada awal proses pengembangan. Rencana mengenai jumlah

iterasi, durasi, dan ruang lingkup setiap iterasi ditentukan dengan jelas sebelumnya.

Setiap iterasi dianggap sebagai "kerangka" waktu singkat dalam model proses Agile, yang biasanya berlangsung dari satu hingga empat minggu. Pembagian seluruh proyek menjadi bagian-bagian yang lebih kecil membantu meminimalkan risiko proyek dan mengurangi persyaratan waktu penyelesaian proyek secara keseluruhan. Setiap iterasi melibatkan tim yang bekerja melalui siklus hidup pengembangan perangkat lunak penuh termasuk perencanaan, analisis persyaratan, desain, pengkodean, dan pengujian sebelum produk yang berfungsi ditunjukkan kepada klien.

3.2.1 Empat nilai Agile

Pada tahun 2001, 17 profesional pengembangan perangkat lunak berkumpul untuk mendiskusikan konsep seputar ide pengembangan perangkat lunak ringan dan akhirnya menciptakan Agile Manifesto. Manifesto menguraikan empat nilai inti Agile, dan meskipun ada perdebatan tentang apakah Manifesto telah melampaui kegunaannya, Manifesto tetap menjadi inti dari gerakan Agile.

Empat nilai inti yang diuraikan dalam Agile Manifesto adalah sebagai berikut:

1. Interaksi individu lebih penting daripada proses dan alat. Orang-orang mendorong proses pengembangan dan menanggapi kebutuhan bisnis dimana mereka adalah bagian terpenting dari pengembangan dan harus dihargai di atas proses dan alat. Jika proses atau alat mendorong pengembangan, maka tim akan cenderung tidak menanggapi dan tidak beradaptasi dengan perubahan sehingga cenderung tidak memenuhi kebutuhan pelanggan (Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, 2018).
2. Fokus pada perangkat lunak yang berfungsi daripada dokumentasi menyeluruh: Dokumentasi dari setiap proses

pengembangan perangkat lunak Agile adalah komponen penting dan berharga tetapi jumlah waktu dan sumber daya yang diberikan kepadanya harus dikontrol dan dioptimalkan agar tidak membebani proses pengembangan perangkat lunak. Sebelum metode Agile, banyak waktu dihabiskan untuk mendokumentasikan produk selama pengembangan untuk pengiriman. Daftar persyaratan yang terdokumentasi sangat panjang dan akan menyebabkan penundaan yang lama dalam proses pengembangan (Larson, D., & Chang, 2016). Meskipun Agile tidak menghilangkan penggunaan dokumentasi, Agile menyederhanakannya sedemikian rupa sehingga hanya menyediakan informasi yang dibutuhkan pengembang untuk melakukan pekerjaan -- seperti tentang pengguna. Oleh karena itu, dokumentasi yang penting dalam Agile adalah dokumentasi yang menyajikan perangkat lunak yang bekerja (Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, 2018) dan menambah nilai pada proses tersebut. Manifesto Agile memberi nilai pada proses dokumentasi, tetapi menempatkan nilai lebih tinggi pada perangkat lunak yang berfungsi. (Gupta, S., & Gouttam, 2017).

3. Kolaborasi alih-alih negosiasi kontrak: Pengembangan perangkat lunak Agile diciptakan untuk mengatasi perubahan persyaratan pada tahap apa pun, akibatnya umpan balik pelanggan, negosiasi, dan kolaborasi dengan tim pengembangan diperlukan secara berkala di seluruh proses untuk mencapai kebutuhan nyata yang dibutuhkan pelanggan daripada melalui perjanjian dan kontrak formal. Namun, kontrak yang menentukan hubungan antara tim pengembangan PL dan pelanggan dan untuk menggambarkan bisnis perangkat lunak masih diperlukan (Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, 2018).

4. Fokus pada menanggapi perubahan: Saat proses pengembangan perangkat lunak sedang berlangsung, baik pengembang dan pelanggan akan memperoleh lebih banyak pengetahuan dan pemahaman yang lebih baik tentang sistem, oleh karena itu penambahan atau pembatalan beberapa persyaratan mungkin diperlukan. Prioritas dalam manifesto Agile diberikan terhadap perubahan dalam siklus hidup proses pengembangan daripada mengikuti rencana yang ditentukan secara ketat, karena tujuan akhir yang ingin dicapai adalah kepuasan pelanggan.

Gambar 3.1. Empat nilai Agile dan 17 orang pelopor Agile Manifesto

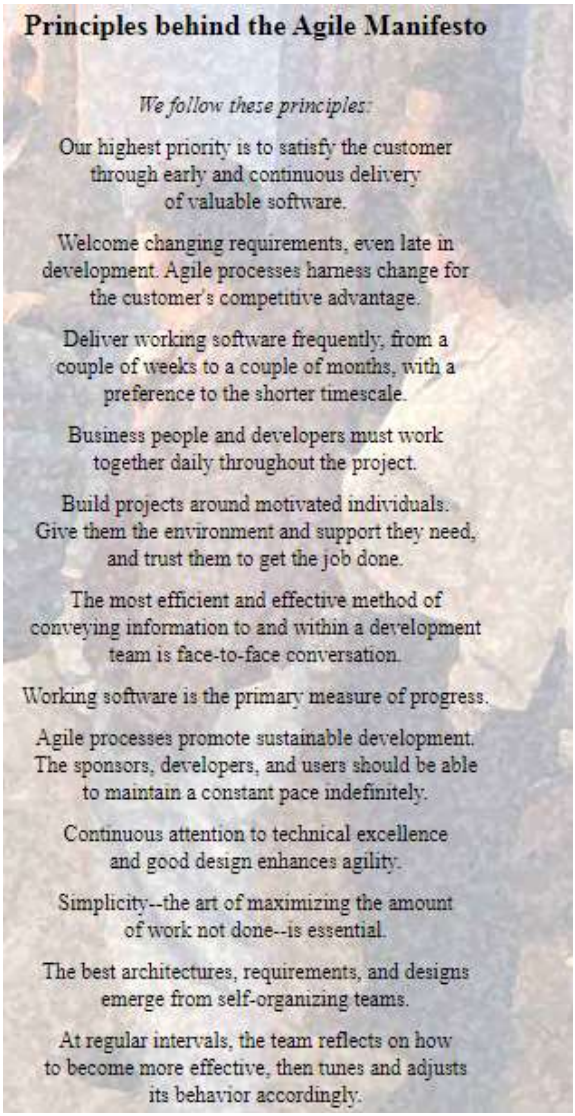
(Sumber: (K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, 2001a))

3.2.2 12 prinsip Agile

Pada tahun 2001, 17 profesional pengembangan perangkat lunak

Selain keempat nilai ini, Agile Manifesto menetapkan dua belas prinsip panduan untuk metodologi berbasis pengembangan Agile yang mencakup (Conboy, 2009):

1. Prinsip 1: Prioritas tertinggi adalah untuk memuaskan pelanggan melalui pengiriman perangkat lunak yang lebih awal dan berkelanjutan.
2. Prinsip 2: Pecah pekerjaan besar menjadi tugas-tugas kecil yang dapat diselesaikan dengan cepat.
3. Prinsip 3: Kenali bahwa karya terbaik muncul dari tim yang mengatur dirinya sendiri.
4. Prinsip 4: Menyediakan individu yang termotivasi dengan lingkungan dan dukungan yang mereka butuhkan dan mempercayai mereka untuk menyelesaikan pekerjaan.
5. Prinsip 5: Ciptakan proses yang mempromosikan upaya berkelanjutan.
6. Prinsip 6: Pertahankan kecepatan konstan untuk menyelesaikan pekerjaan.
7. Prinsip 7: Menyambut perubahan persyaratan, bahkan di akhir proyek.
8. Prinsip 8: Kumpulkan tim proyek dan pemilik bisnis setiap hari selama proyek berlangsung.
9. Prinsip 9: Mintalah tim merenungkan secara berkala tentang bagaimana menjadi lebih efektif, kemudian atur dan sesuaikan perilaku yang sesuai.
10. Prinsip 10: Ukur kemajuan dengan jumlah pekerjaan yang diselesaikan.
11. Prinsip 11: Terus mencari keunggulan.
12. Prinsip 12: Memanfaatkan perubahan untuk keunggulan kompetitif.

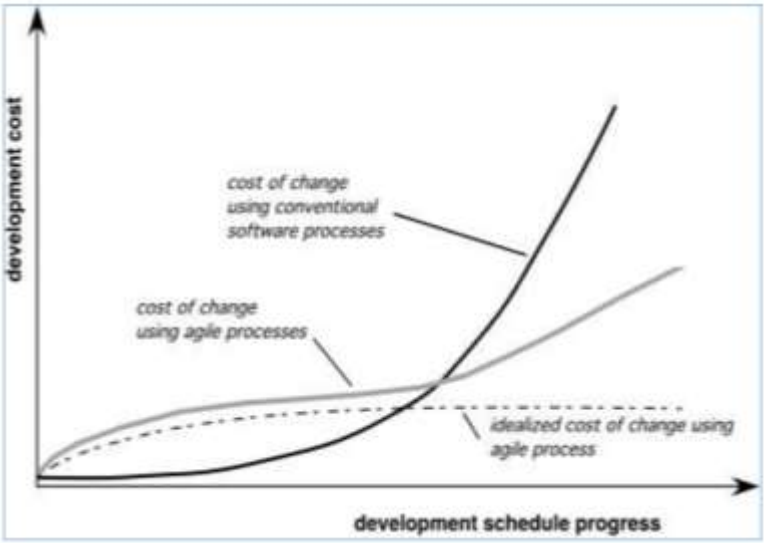


Gambar 3.2. 12 Prinsip Agile dalam Agile Manifesto
(Sumber: (K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, 2001b))

Karena pengurangan biaya dan produktivitas, kualitas, dan kepuasan yang lebih baik, paradigma pengembangan PL Agile menyebabkan gelombang besar dalam industri pengembangan perangkat lunak dalam usia hampir dua dekade (Erickson, J., Lyytinen, K., & Siau, 2005). Hal ini dilakukan dengan bantuan teknik penanganan yang fleksibel dan mekanisme komunikasi dan koordinasi yang lebih baik (Rauf, A., & AlGhafees, 2015). Keuntungan utama yang dicapai pengembangan perangkat lunak berbasis Agile (Choudhary, B., & Rakesh, 2016) adalah:

1. Peningkatan komunikasi dan koordinasi antar anggota tim.
2. Rilis cepat.
3. Fleksibilitas desain.
4. Proses yang lebih masuk akal.

Salah satu karakteristik utama dari pengembangan perangkat lunak Agile adalah kemampuan untuk beradaptasi dengan perubahan, seperti dapat dilihat pada Gambar 4.3 (Pressman, 2005). Dari gambar tersebut menyatakan bahwa proses pengembangan perangkat lunak konvensional menanggapi perubahan dengan peningkatan biaya secara nonlinier selama proyek berlangsung. Di sisi lain, dalam menggunakan proses Agile terdapat pengurangan biaya dan cenderung rata, dimana memungkinkan kemajuan (*progress*) pada tahap selanjutnya tanpa biaya dan waktu yang berlebihan.



Gambar 3.3. Biaya pengembangan versus perubahan dalam proses pengembangan
(Sumber: (Pressman, 2005))

3.3 Pengembangan Perangkat Lunak Agile vs Pengembangan Perangkat Lunak Tradisional

Terdapat perbedaan beberapa karakteristik antara metode pengembangan perangkat lunak Agile (ringan) dan metode pengembangan perangkat lunak tradisional (berat), karakteristik tersebut dapat dikategorikan dan disajikan pada Tabel 3.1 (Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, 2015; Vijayasarathy, L. R., & Butler, 2016):

Tabel 3.1. Pengembangan PL Agile vs Pengembangan PL Tradisional

Parameter	**Metode Tradisional**	**Metode Agile**
Kemudahan Modifikasi	Sulit	Mudah
Pendekatan Pengembangan	Prediktif	Adaptif
Orientasi Pengembangan	Berorientasi Proses	Berorientasi Pelanggan
Ukuran Proyek	Besar	Kecil atau Sedang
Skala Perencanaan	Jangka Panjang	Jangka Pendek
Gaya Manajemen	Komando dan Kontrol	Kepemimpinan dan Kolaborasi
Belajar	Pembelajaran Berkelanjutan sambil Pengembangan	Pembelajaran adalah sekunder untuk Pengembangan
Dokumentasi	Tinggi	Rendah
Tipe Organisasi	Pendapatan Tinggi	Pendapatan Sedang dan Rendah
Jumlah Karyawan Organisasi	Besar	Kecil
Anggaran	Tinggi	Rendah
Jumlah Tim	Banyak	Satu
Ukuran Tim	Sedang	Kecil

Seperti yang ditunjukkan pada perbandingan antara metode pengembangan perangkat lunak Agile dan tradisional; metode Agile lebih adaptif terhadap perubahan yang diminta,

karena itu selalu ada kolaborasi langsung antara pengembang dan pelanggan. Agile lebih efisien untuk digunakan pada proyek-proyek kecil dan menengah. Perencanaan pada metode Agile bersifat jangka pendek untuk fungsionalitas tertentu yang lebih mudah dilakukan dibandingkan dengan metode tradisional yang membutuhkan perencanaan jangka panjang atau perencanaan proyek secara keseluruhan. Pendokumentasian untuk metode tradisional harus tingkat tinggi, lebih rinci, sehingga membutuhkan kerja keras dan hal itu akan menjadi dokumen besar. Sejumlah tim dan karyawan yang terlibat dalam proyek perangkat lunak dengan menggunakan metode tradisional lebih banyak daripada metode Agile dan tentunya akan membutuhkan jumlah anggaran yang lebih tinggi (Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, 2015; Vijayasarathy, L. R., & Butler, 2016).

3.4 Siklus pengembangan perangkat lunak Agile

Siklus pengembangan perangkat lunak Agile dapat dipecah menjadi enam langkah berikut:

1. Pengumpulan persyaratan (*Requirements gathering*)
2. Desain persyaratan (*Design the requirements*)
3. Konstruksi/iterasi (*Construction/iteration*)
4. Pengujian/Jaminan kualitas (*Testing/Quality assurance*)
5. Rilis produk (*Deployment*)
6. Umpan balik (*Feedback*)

Gambar 3.4. Siklus pengembangan PL Agile
(Sumber: (Javatpoint, 2021))

Pengumpulan persyaratan: Pada fase ini, tim proyek harus menentukan dulu persyaratan. Tim proyek harus menjelaskan peluang bisnis dan merencanakan waktu dan upaya yang diperlukan untuk membangun proyek. Berdasarkan informasi ini, tim proyek dapat mengevaluasi kelayakan teknis dan ekonomi.

Rancang persyaratan: Ketika tim proyek telah mengidentifikasi proyek, bekerjalah dengan pemangku kepentingan untuk menentukan persyaratan. Tim proyek dapat menggunakan diagram alir atau diagram UML untuk menunjukkan cara kerja fitur baru dan menunjukkan bagaimana penerapannya pada sistem yang ada.

Konstruksi/ iterasi: Saat tim menentukan persyaratan, pekerjaan dapat dimulai. Desainer dan pengembang mulai mengerjakan proyek tersebut, yang bertujuan untuk

menerapkan produk yang berfungsi. Produk akan menjalani berbagai tahap penyempurnaan, sehingga mencakup fungsionalitas yang sederhana dan minimal.

Pengujian: Pada fase ini, tim *Quality Assurance* memeriksa kinerja produk dan mencari *bug*.

Rilis produk: Pada fase ini, tim mengeluarkan produk untuk lingkungan kerja pengguna (*user*).

Umpan balik: Setelah merilis produk, langkah terakhir adalah umpan balik. Dalam hal ini, tim proyek menerima umpan balik tentang produk dan bekerja melalui umpan balik tersebut.

3.5 Metode-metode Agile

Metode Agile adalah proses yang mendukung filosofi Agile, yaitu nilai dan prinsip Agile. Setiap metode Agile terdiri dari kombinasi praktik yang berbeda, yang merupakan deskripsi tentang bagaimana pekerjaan sehari-hari dilakukan oleh pengembang perangkat lunak. Setiap metode berbeda satu sama lain dengan memilih rangkaian terminologi dan praktik yang sesuai (Elbanna, A., & Sarker, 2016). Ada berbagai jenis metode Agile seperti metode Test-Driven Development (TDD), metode Feature Driven Development (FDD), metode Extreme Programming (XP), metode Scrum, metode Dynamic System Development Model (DSDM) dan metode Crystal dan lain-l. Setiap metode memiliki prinsip, siklus hidup, peran, kelebihan aindan kekurangannya sendiri, dll. Semua metode pengembangan perangkat lunak tangkas ini membangun perangkat lunak dalam iterasi dan proses inkremental.

SCRUM adalah proses pengembangan tangkas yang berfokus terutama pada cara mengelola tugas dalam kondisi pengembangan berbasis tim. Ada tiga peran di dalamnya, dan tanggung jawab mereka adalah:

1. Scrum Master: Scrum dapat mengatur tim master, mengatur pertemuan dan menghilangkan hambatan untuk proses tersebut
2. Pemilik produk: Pemilik produk membuat backlog produk, memprioritaskan penundaan dan bertanggung jawab atas distribusi fungsionalitas pada setiap pengulangan.
3. Tim Scrum: Tim mengelola pekerjaannya dan mengatur pekerjaan untuk menyelesaikan sprint atau siklus.

eXtreme Programming (XP). Jenis metodologi ini digunakan saat pelanggan terus-menerus mengubah permintaan atau persyaratan, atau saat mereka tidak yakin dengan kinerja sistem.

Crystal. Ada tiga konsep dari metode ini:

1. Pencarteran: Berbagai aktivitas terlibat dalam fase ini seperti membuat tim pengembangan, melakukan analisis kelayakan, mengembangkan rencana, dll.
2. Pengiriman siklik: di bawah ini, terdiri dari dua siklus lagi, yaitu:
   A. Tim memperbarui rencana rilis.
   B. Pengiriman produk terintegrasi ke pengguna.
3. Penutup: Sesuai dengan lingkungan pengguna, fase ini melakukan penerapan, pasca penerapan.

Dynamic Software Development Method **(DSDM)**. DSDM adalah strategi pengembangan aplikasi cepat untuk pengembangan perangkat lunak dan memberikan struktur distribusi proyek yang gesit. Fitur penting DSDM adalah pengguna harus terhubung secara aktif, dan tim diberi hak untuk mengambil keputusan. Teknik yang digunakan dalam DSDM adalah:

1. Time Boxing
2. MoSCoW Rules
3. Prototyping

Proyek DSDM berisi tujuh tahap:

1. Pra-proyek
2. Studi Kelayakan
3. Studi Bisnis
4. Iterasi Model Fungsional
5. Merancang dan membangun Iterasi
6. Implementasi
7. Pasca proyek

Feature Driven Development **(FDD)**. Metode ini berfokus pada fitur "Merancang dan Membangun". Berbeda dengan metode pintar lainnya, FDD menjelaskan langkah-langkah kecil dari pekerjaan yang harus didapatkan secara terpisah per fungsi.

Lean Software Development. Metodologi pengembangan perangkat lunak lean mengikuti prinsip "produksi tepat waktu". Metode lean menunjukkan peningkatan kecepatan pengembangan perangkat lunak dan pengurangan biaya. Pengembangan lean dapat diringkas dalam tujuh fase:

1. Menghilangkan Limbah
2. Memperkuat pembelajaran
3. Menunda komitmen (memutuskan selambat mungkin)
4. Pengiriman awal
5. Memberdayakan tim
6. Membangun Integritas
7. Optimalkan secara keseluruhan

DAFTAR PUSTAKA

Braude, E. J., & Bernstein, M. E. 2011. *Software Engineering: Modern Approaches*. Second Edi. Waveland Press, Inc.

Choudhary, B., & Rakesh, S. K. 2016. 'An approach using agile method for software development', in *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, pp. 155–158. doi: 10.1109/icicccs.2016.7542304.

Conboy, K. 2009. 'Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development', *Information Systems Research*, 20(3), pp. 329–354. doi: 10.1287/isre.1090.0236.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. 2012. 'A decade of agile methodologies: Towards explaining agile software development', *Journal of Systems and Software*, 85(6), pp. 1213–1221. doi: 10.1016/j.jss.2012.02.033.

Elbanna, A., & Sarker, S. 2016. 'The Risks of Agile Software Development: Learning from Adopters', *IEEE Software*, 33(5), pp. 72–79. doi: 10.1109/ms.2015.150.

Erickson, J., Lyytinen, K., & Siau, K. 2005. 'Agile Modeling, Agile Software Development, and Extreme Programming', *Journal of Database Management*, 16(4), pp. 88–100. doi: 10.4018/jdm.2005100105.

Giuffrida, R., & Dittrich, Y. 2015. 'A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams', *Information and Software Technology Journal*, 63, pp. 11–30. doi: 10.1016/j.infsof.2015.02.013.

Gupta, S., & Gouttam, D. 2017. 'Towards changing the paradigm of software development in software industries: An emergence of agile software development', in *2017 IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, pp. 18–21. doi: 10.1109/icstm.2017.8089120.

IEEE. 1990. *IEEE Standard Glossary of Software Engineering Terminology*. New York. doi: 10.1109/IEEESTD.1990.101064.

Javatpoint. 2021. *Phases of Agile Model*, *javatpoint*. Available at: https://www.javatpoint.com/software-engineering-agile-model (Accessed: 1 July 2023).

K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. 2001a. *Manifesto for Agile Software Development*. Available at: https://agilemanifesto.org/.

K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. 2001b. *Principles behind the Agile Manifesto*, *Agile Manifesto*. Available at: https://agilemanifesto.org/principles.html (Accessed: 1 July 2023).

Larson, D., & Chang, V. 2016. 'A review and future direction of agile, business intelligence, analytics and data science', *International Journal of Information Management*, 36(5), pp. 700–710. doi: 10.1016/j.ijinfomgt.2016.04.013.

Mall, R. 2014. *Fundamentals of Software Engineering*. Fourth Edi. PHI Learning Private Limited.

Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, P. 2015. 'Empirical Study of Agile Software Development Methodologies', *ACM SIGSOFT Software Engineering Notes*, 40(1), pp. 1–6. doi: 10.1145/2693208.2693233.

Pressman, R. S. 2005. *Software Engineering: A Practitioner's Approach*. London: Palgrave Macmillan.

Rauf, A., & AlGhafees, M. 2015. 'Gap Analysis between State of Practice and State of Art Practices in Agile Software Development', in *2015 Agile Conference*, pp. 102–106. doi: 10.1109/agile.2015.21.

Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, P. 2018. 'Advances in Using Agile and Lean Processes for Software Development', in *Advances in Computers*, pp. 1–90. doi: 10.1016/bs.adcom.2018.03.014.

Sommerville, I. 2011. *Software Engineering*. Ninth Edit. Boston, USA: Addison-Wesley.

Vijayasarathy, L. R., & Butler, C. W. 2016. 'Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?', *IEEE Software*, 33(5), pp. 86–94. doi: 10.1109/ms.2015.26.

BAB 4
REKAYASA KEBUTUHAN PERANGKAT LUNAK

Oleh Addin Aditya

4.1 Pendahuluan

Seperti yang telah diketahui bersama bahwa dunia ini sudah memasuki era digitalisasi. Perangkat lunak atau software menjadi kebutuhan penting disetiap aspek kehidupan manusia, mulai dari bisnis hingga kehidupan sehari-hari. Kebutuhan yang variatif dari para pemangku jabatan dan individu menuntut adanya pengembangan perangkat lunak yang dapat memenuhi ekspektasi dan kebutuhan mereka. Di satu sisi, kurangnya keterampilan dalam memahami, mengumpulkan, dan mengelola kebutuhan ini dapat menyebabkan proyek perangkat lunak yang kurang memuaskan atau bahkan gagal sepenuhnya.

Untuk mengatasi dan mengantisipasi kegagalan proyek pengembangan perangkat lunak, maka penting untuk memahami konsep rekayasa kebutuhan perangkat lunak baik secara teoritik maupun secara praktik. Rekayasa Kebutuhan adalah disiplin ilmu dan praktik yang berfokus pada proses identifikasi, analisis, dokumentasi, dan manajemen kebutuhan dalam pengembangan perangkat lunak. Proses ini mencakup aktivitas-aktivitas penting seperti pengumpulan kebutuhan dari para stakeholder, analisis untuk mendalami kebutuhan pengembangan perangkat lunak dengan lebih komprehensif, serta pembuatan spesifikasi yang jelas, terstruktur dan tervalidasi.

Urgensi Rekayasa Kebutuhan dalam pengembangan perangkat lunak ialah memastikan bahwa perangkat lunak yang dikembangkan dapat memenuhi ekspektasi pengguna dan memenuhi tujuan bisnis yang mendasarinya. Dengan mengidentifikasi dan mendefinisikan kebutuhan dengan tepat sasaran, maka kegagalan proyek dapat diminimalisir. Rekayasa kebutuhan juga dapat membantu menekan biaya dan waktu karena perubahan yang terlambat dalam siklus pengembangan perangkat lunak.

Dalam implementasinya, Rekayasa Kebutuhan melibatkan berbagai pihak, seperti analis sistem, pengembang perangkat lunak, manajer proyek, pemilik bisnis, dan pengguna akhir. Komunikasi yang efektif dan kerjasama di antara semua pihak ini akan menjadi kunci untuk memahami kebutuhan yang bervariatif dan kompleks.

Rekayasa kebutuhan adalah fase paling efektif dari proses pengembangan perangkat lunak. Fase ini bertujuan untuk mengumpulkan kebutuhan yang baik dari pemangku kepentingan dengan cara yang benar. Penting bagi setiap organisasi untuk mengembangkan produk perangkat lunak berkualitas yang dapat memenuhi kebutuhan pengguna (Pandey, Suman, & Ramani, 2010). Tantangan dalam Rekayasa Kebutuhan Perangkat Lunak meliputi pemahaman mendalam tentang kebutuhan pengguna, manajemen perubahan kebutuhan yang terus berkembang, dan koordinasi yang baik di antara semua pemangku kepentingan. Oleh karena itu, alat dan teknik yang tepat seringkali digunakan untuk membantu dalam proses identifikasi, analisis, dan dokumentasi kebutuhan.

4.2 Pengertian dan Ruang Lingkup Rekayasa Kebutuhan Perangkat Lunak

Rekayasa Kebutuhan Perangkat Lunak adalah suatu disiplin ilmu dan proses yang berfokus pada identifikasi, analisis, dokumentasi, dan manajemen kebutuhan yang berkaitan dengan pengembangan perangkat lunak. Para ahli dalam bidang ini memberikan pengertian yang berbeda, namun pada dasarnya mereka sepakat bahwa Rekayasa Kebutuhan merupakan tahap penting dalam siklus pengembangan perangkat lunak yang bermakna untuk memastikan perangkat lunak yang dikembangkan memenuhi ekspektasi pengguna dan pemangku kepentingan lainnya.

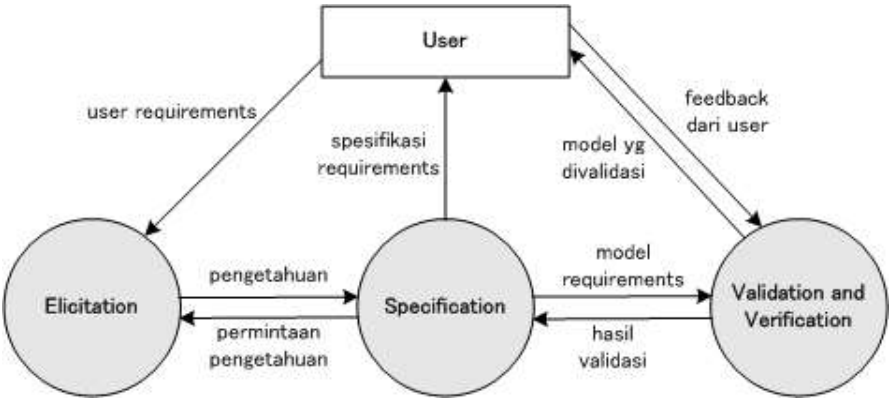
1. Rekayasa kebutuhan adalah serangkaian mekanisme yang tepat untuk memahami apa yang diinginkan pelanggan, menganalisis kebutuhan, menilai kelayakan, menegosiasikan solusi yang masuk akal, menentukan kata kunci dengan jelas, memvalidasi spesifikasi, dan mengelola persyaratan saat diubah menjadi sistem operasional (Pressman, 2010).
2. Rekayasa Kebutuhan adalah proses pemahaman kebutuhan dan kendala sistem yang akan dikembangkan dan menyusun spesifikasi dari kebutuhan-kebutuhan tersebut. Proses ini melibatkan komunikasi yang baik dengan pemangku kepentingan untuk memastikan bahwa kebutuhan mereka dipahami dengan benar dan diimplementasikan dalam perangkat lunak yang dikembangkan (Sommervile, 2011).
3. Rekayasa Kebutuhan adalah proses kolaboratif untuk mendefinisikan fitur-fitur produk yang akan dibangun atau sistem yang akan dikembangkan. Proses ini melibatkan mengumpulkan informasi dari berbagai sumber, memahami kebutuhan pengguna, menetapkan prioritas,

dan menyusun spesifikasi kebutuhan yang jelas dan terukur (Wiegers & Beatty, 2013)

4. Rekayasa kebutuhan merupakan proses penetapan kebutuhan dengan mempelajari kebutuhan pemangku kepentingan dan proses menganalisis dan menyempurnakan spesifikasi tersebut secara sistematis (Hofmann & Lehner, 2001)

Dari pengertian di atas, dapat disimpulkan bahwa Rekayasa Kebutuhan Perangkat Lunak melibatkan proses komprehensif untuk memahami, mendokumentasikan, dan mengelola kebutuhan dari perangkat lunak yang akan dikembangkan. Proses ini berpusat pada pengumpulan data dan informasi yang akurat dari berbagai aktor dan mentransformasikannya menjadi spesifikasi kebutuhan yang jelas dan tervalidasi, sehingga menjadi landasan untuk pengembangan perangkat lunak yang berhasil.

Ruang lingkup Rekayasa Kebutuhan Perangkat Lunak mencakup berbagai aspek yang terkait dengan identifikasi, analisis, dokumentasi, dan manajemen kebutuhan dalam pengembangan perangkat lunak. Ruang lingkup ini melibatkan berbagai tahap dan proses yang bertujuan untuk memastikan bahwa perangkat lunak yang dikembangkan dapat memenuhi harapan dan kebutuhan para pengguna dan pemangku kepentingan lainnya. Berikut adalah ruang lingkup atau cakupan aktivitas dari rekayasa kebutuhan perangkat lunak (Lamsweerde, 2000):



Gambar 4.2. Domain Utama Rekayasa Kebutuhan
Sumber: https://romisatriawahono.net/2006/04/29/menyegarkan-kembali-pemahaman-tentang-requirement-engineering/

1. **Domain Analysis**
   Sistem yang ada saat ini perlu untuk dipelajari terlebih dahulu. Para pemangku kepentingan yang memiliki akses langsung pada sistem perlu untuk diidentifikasi dan diwawancarai guna mendapatkan permasalahan yang terjadi pada sistem saat ini. Selanjutnya akan didapatkan peluang dan target yang akan dibuat.
2. **Elicitation**
   Setelah kebutuhan teridentifikasi, tahap analisis berfokus pada memahami, menganalisis, dan memodelkan kebutuhan tersebut. Analisis kebutuhan bertujuan untuk memastikan bahwa kebutuhan tersebut masuk akal, konsisten, dan dapat diimplementasikan dalam perangkat lunak.
3. **Negotiation and Agreement**
   Setiap alternatif dari kebutuhan pengembangan perangkat lunak akan dievaluasi dan dipertimbangkan berdasarkan analisis risiko dan proses pengambilan keputusan terbaik

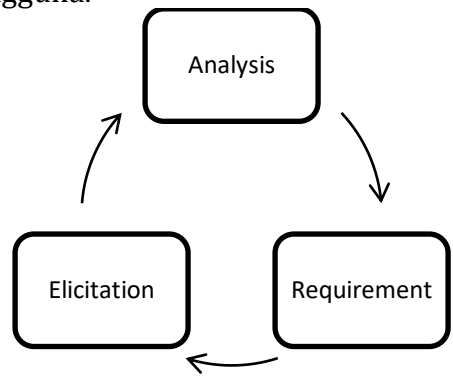
berdasarkan semua aktor yang terlibat dalam proyek pengembangan perangkat lunak.

4. **Verification and Validation**
   Tahap ini melibatkan proses memverifikasi dan memvalidasi kebutuhan untuk memastikan bahwa kebutuhan yang terdokumentasi benar, konsisten, dan memenuhi kriteria kualitas tertentu. Verifikasi kebutuhan berkaitan dengan keakuratan dan kesesuaian kebutuhan dengan kebutuhan pengguna, sementara validasi kebutuhan berfokus pada memastikan bahwa kebutuhan tersebut benar-benar memenuhi kebutuhan bisnis dan pengguna.
5. **Documentation**
   Ruang lingkup ini mencakup proses mendokumentasikan kebutuhan dengan jelas dan terstruktur. Dokumen kebutuhan ini menjadi panduan bagi para pengembang perangkat lunak untuk memahami apa yang harus mereka bangun dan mencapai.
6. **Evolution**
   Ruang lingkup ini mencakup pengelolaan perubahan dan konflik kebutuhan selama siklus hidup pengembangan perangkat lunak. Manajemen kebutuhan memastikan bahwa perubahan kebutuhan dikelola dengan baik dan dipahami oleh semua pemangku kepentingan.

4.3 Requirement Elicitation

4.3.1 Pengertian Elicitation

Requirement elicitation adalah proses pengumpulan, analisis, dan dokumentasi kebutuhan yang diperlukan untuk mengembangkan atau memperbarui suatu sistem atau perangkat lunak. Pengumpulan kebutuhan yang tepat dan komprehensif merupakan langkah penting dalam memastikan bahwa solusi yang dihasilkan akan memenuhi harapan dan kebutuhan pengguna.



Gambar 4.3. Siklus Hidup Requirement Elicitations

Pada gambar 4.2 menunjukkan siklus dari requirement elicitations. Analis akan melakukan kegiatan elisitasi atau pengumpulan kebutuhan, mempelajari atau menganalisis hasil pengumpulan kebutuhan. Analisis yang dimaksud adalah meninjau Kembali hasil pengumpulan kebutuhan, apakah sudah sesuai dengan tujuan atau perlu untuk dilakukan pengumpulan kebutuhan Kembali. Selanjutnya akan ditetapkan kebutuhan yang sah sebagai dasar untuk dikembangkan sebuah proyek perangkat lunak.

4.3.2 Teknik Elicitation

Teknik elisitasi kebutuhan adalah sarana yang digunakan analis sistem untuk menentukan masalah, peluang, dan kebutuhan pelanggan sehingga pengembang sistem dapat membangun sistem yang menyelesaikan masalah tersebut, memanfaatkan peluang tersebut, dan memenuhi kebutuhan pelanggan". Teknik elisitasi adalah alat untuk menemukan & pemahaman yang tepat. Tujuan dari teknik Elisitasi adalah untuk menemukan sebanyak mungkin permasalahan sehingga memudahkan stakeholder untuk mendapatkan aplikasi terbaik yang sesuai dengan kebutuhan (Khan, Dulloo, & Verma, 2014). Berikut adalah beberapa macam teknik elicitation :

1. **Wawancara**
   Wawancara terdiri dari mengajukan pertanyaan kepada pakar atau ahli tentang domain minat dan bagaimana mereka melakukan tugas mereka. Wawancara bisa tidak terstruktur, semi terstruktur, atau terstruktur. Wawancara melibatkan interaksi langsung dengan pemangku kepentingan, pengguna, atau ahli terkait untuk mendapatkan pemahaman mendalam tentang kebutuhan mereka. Wawancara dapat dilakukan secara individu atau dalam kelompok diskusi
2. **Kuesioner**
   Kuesioner adalah teknik yang sangat penting dalam teknik elisitasi kebutuhan, kuesioner membantu mendapatkan informasi dari banyak orang, analis dapat mengumpulkan pendapat dengan dua cara : untuk mendapatkan bukti statistik untuk asumsi, atau untuk mengumpulkan pendapat dan saran.
3. **Observasi**
   Kegiatan ini Melibatkan pengamatan langsung terhadap pengguna atau pemangku kepentingan saat mereka menggunakan sistem yang ada atau dalam konteks yang relevan. Observasi dapat memberikan wawasan tentang

perilaku, kebutuhan yang mungkin tidak disadari, dan tantangan yang dihadapi pengguna (Burge, 2001)

4. **Analisis Dokumen**
   Metode ini adalah metode tidak langsung dan bervariasi tergantung pada dokumen yang tersedia, dan interaksi dengan para ahli. Metode ini melibatkan analisis dokumen yang ada seperti laporan, kebijakan, spesifikasi, atau dokumentasi terkait sistem yang sedang direkayasa. Dokumen ini dapat memberikan pemahaman tentang kebutuhan yang sudah ada, proses bisnis yang ada, atau batasan yang harus diperhatikan.
5. **Prototyping**
   Prototyping telah digunakan untuk teknik elicitation di mana ada banyak ketidakpastian tentang kebutuhan, atau di mana diperlukan *feedback* awal dari pemangku kepentingan (Davis, 1992). Teknik ini melibatkan pembuatan prototipe awal atau model yang digunakan untuk berinteraksi dengan pengguna atau pemangku kepentingan. Prototipe ini membantu dalam memvalidasi dan menggali kebutuhan secara lebih konkret serta memungkinkan pengguna melihat dan merasakan fitur-fitur yang diusulkan.

4.4 Analisis Spesifikasi Kebutuhan Perangkat Lunak

Software Requirement Specifications (SRS) atau Spesifikasi perangkat lunak dapat didefinisikan sebagai pernyataan singkat tentang kebutuhan yang harus dijamin oleh perangkat lunak. Melalui kebutuhan ini, perangkat lunak harus menyediakan fasilitas atau kemampuan kepada pengguna, yang memungkinkan mereka mencapai tujuan organisasi yang ditentukan (Belfo, 2012). Spesifikasi Kebutuhan Perangkat Lunak adalah dokumen yang mendefinisikan kebutuhan

fungsional dan non-fungsional dari suatu sistem perangkat lunak. Dokumen ini menyediakan deskripsi yang lengkap tentang apa yang diharapkan dari sistem yang akan dikembangkan, termasuk fitur-fitur, fungsi, kinerja, batasan, dan persyaratan lainnya.

SRS biasanya dibuat pada tahap awal pengembangan perangkat lunak dan menjadi panduan untuk tim pengembang dalam merancang, mengimplementasikan, dan menguji sistem tersebut. Dokumen ini berfungsi sebagai kontrak antara pengguna atau pemilik proyek dengan tim pengembang, memastikan bahwa semua pihak memiliki pemahaman yang jelas tentang apa yang diharapkan dari sistem yang akan dikembangkan.

4.4.1 Proses Analisis Kebutuhan

Proses analisis kebutuhan dimulai dengan mengidentifikasi dan mendefinisikan masalah atau peluang yang ingin ditangani oleh sistem perangkat lunak. Ini diikuti dengan mengumpulkan persyaratan dari pemangku kepentingan, yang meliputi pengguna, pelanggan, dan pihak lain yang akan terpengaruh oleh sistem perangkat lunak. Persyaratan tersebut kemudian didokumentasikan dan diatur, dan setiap ketidakkonsistenan atau ambiguitas diselesaikan. Langkah terakhir dalam proses analisis persyaratan adalah memvalidasi persyaratan, yang mencakup pengujian dan evaluasi persyaratan untuk memastikan bahwa persyaratan tersebut lengkap, konsisten, dan dapat dicapai

4.4.4 Pemodelan dan Notasi dalam Spesifikasi Kebutuhan

1. **Notasi Pemodelan Proses Bisnis**

   Model dan Notasi Proses Bisnis telah menjadi standar de-facto untuk diagram proses bisnis (Object Management Group, 2011). Notasi Pemodelan Proses Bisnis (BPMN) adalah teknik yang banyak digunakan untuk memodelkan dan menganalisis proses bisnis. Diagram BPMN digunakan untuk merepresentasikan secara visual langkah-langkah

dalam suatu proses, termasuk input, output, dan poin keputusan. Diagram BPMN membantu mengidentifikasi dan mendokumentasikan persyaratan bisnis dan dapat digunakan untuk mengomunikasikan persyaratan tersebut kepada tim pengembangan perangkat lunak

2. **Unified Modelling Language**
   UML (Unified Modeling Language) adalah bahasa yang digunakan untuk mendeskripsikan, memvisualisasikan, membangun, dan mendokumentasikan berbagai artefak dari sistem intensif perangkat lunak (Object Management Group, 2017). Diagram UML dapat digunakan untuk memodelkan berbagai aspek sistem perangkat lunak, termasuk use case, class diagram, dan sequence diagram. Diagram UML berguna untuk memodelkan dan menganalisis persyaratan perangkat lunak dan dapat digunakan untuk mengomunikasikan persyaratan tersebut kepada tim pengembangan perangkat lunak.
3. **Teknik Flowchart**
   Flowchart adalah teknik yang banyak digunakan untuk secara visual mewakili langkah-langkah dalam suatu proses. Flowchart membantu mengidentifikasi dan mendokumentasikan persyaratan dan dapat digunakan untuk mengkomunikasikan persyaratan kepada tim pengembangan perangkat lunak. Flowchart juga membantu dalam mengidentifikasi potensi masalah dan peluang dalam suatu proses.
4. **Diagram Aliran Data (DFD)**
   Data Flow Diagram (DFD) adalah teknik untuk merepresentasikan secara visual aliran data melalui suatu sistem. Diagram DFD berguna untuk mengidentifikasi dan mendokumentasikan persyaratan dan dapat digunakan untuk mengkomunikasikan persyaratan kepada tim pengembangan perangkat lunak. Diagram DFD juga

berguna untuk mengidentifikasi potensi masalah dan peluang dalam suatu sistem.

5. **Gantt-Chart**
   Gantt Charts adalah teknik yang banyak digunakan untuk penjadwalan dan pelacakan kemajuan proyek. Bagan Gantt sangat membantu untuk mengelola dan memantau proses analisis persyaratan, dan dapat digunakan untuk mengkomunikasikan kemajuan proyek kepada pemangku kepentingan
6. **Analisis Kesenjangan**
   Analisis Gap dalam Rekayasa Kebutuhan Perangkat Lunak digunakan untuk mengidentifikasi perbedaan antara keadaan yang ada saat ini dan keadaan yang diharapkan atau yang seharusnya terjadi. Metode ini membantu dalam mengidentifikasi kekurangan, kesenjangan, atau perbedaan antara kebutuhan yang ada dan kebutuhan yang diinginkan. Analisis Gap membantu dalam mengidentifikasi kebutuhan yang hilang, tidak sesuai, atau tidak lengkap dalam pengembangan perangkat lunak. Hal ini memungkinkan tim pengembang untuk mengambil tindakan yang tepat untuk mengurangi kesenjangan dan memastikan bahwa perangkat lunak yang dikembangkan memenuhi kebutuhan yang diinginkan dan diharapkan.

4.5 Verifikasi dan Validasi Kebutuhan

Verifikasi dan validasi kebutuhan perangkat lunak adalah proses untuk memastikan bahwa kebutuhan yang diidentifikasi dalam tahap requirement elicitation telah terverifikasi dengan benar dan valid. Dalam konteks pengembangan perangkat lunak, verifikasi dan validasi kebutuhan penting untuk memastikan bahwa perangkat lunak yang dikembangkan memenuhi harapan dan kebutuhan pengguna. Aktivitas verifikasi dilakukan untuk memastikan

bahwa perangkat lunak dikembangkan dengan cara yang benar sementara aktivitas validasi dilakukan untuk memastikan bahwa perangkat lunak dikembangkan sesuai dengan kebutuhan pelanggan (Ponsard, Massonet, Rifaut, & Molderez, 2005).

Verifikasi kebutuhan berfokus pada pemeriksaan apakah kebutuhan yang diidentifikasi telah dijelaskan dengan benar dan apakah kebutuhan tersebut dapat diukur dan diverifikasi. Langkah-langkah verifikasi kebutuhan melibatkan memastikan bahwa setiap kebutuhan telah dituliskan dengan jelas, tidak ambigu, dan spesifik. Selain itu, verifikasi juga melibatkan pengecekan apakah kebutuhan konsisten dan tidak bertentangan satu sama lain.

Validasi kebutuhan berfokus pada memastikan bahwa kebutuhan yang diidentifikasi adalah kebutuhan yang benar-benar diperlukan oleh pengguna dan stakeholder. Proses validasi persyaratan membangun kepercayaan bahwa spesifikasi persyaratan mendefinisikan sistem yang harus memenuhi kebutuhan sebenarnya dari pemangku kepentingan sistem (Lobo & Arthur, 2005). Langkah-langkah validasi kebutuhan melibatkan komunikasi dan konfirmasi dengan stakeholder terkait untuk memastikan bahwa kebutuhan tersebut benar-benar mencerminkan kebutuhan mereka dan menggambarkan masalah yang ingin mereka selesaikan. Ada beberapa model verifikasi dan validasi kebutuhan perangkat lunak yang umum digunakan dalam industri. Berikut ini adalah beberapa model yang sering digunakan :

1. **Model Inspeksi**
   Model ini melibatkan pemeriksaan manual dokumen kebutuhan oleh tim yang terlatih. Tim melakukan analisis dan pengecekan terhadap kebutuhan yang telah diidentifikasi untuk memastikan kejelasan, kesesuaian, dan kebenaran kebutuhan tersebut.
2. **Model Peer-Review**

Dalam model ini, kebutuhan diperiksa oleh rekan sejawat atau anggota tim yang terlibat dalam proyek. Tujuan dari peer review adalah mendapatkan umpan balik dan masukan dari individu yang memiliki pengetahuan dan pengalaman yang relevan.

3. **Model Walkthrough**
   Dalam model walkthrough, kebutuhan disajikan secara formal kepada tim proyek atau kelompok pemangku kepentingan yang relevan. Selama sesi walkthrough, setiap kebutuhan dibahas secara rinci, dan umpan balik dan masukan dari peserta digunakan untuk memperbaiki dan memvalidasi kebutuhan.
4. **Prototyping**
   Dalam model ini, prototipe perangkat lunak dibangun untuk memvalidasi kebutuhan dengan cara memperlihatkan fungsionalitas sistem kepada pemangku kepentingan. Prototipe membantu memvisualisasikan dan menguji fitur-fitur yang diusulkan, dan umpan balik dari pemangku kepentingan digunakan untuk memperbaiki kebutuhan.
5. **Model Uji Kasus**
   Dalam model ini, kebutuhan diuji dengan menggunakan serangkaian kasus uji yang dirancang untuk menguji fungsionalitas dan kinerja perangkat lunak. Kebutuhan dianggap tervalidasi jika dapat memenuhi kasus uji dengan sukses.
6. **Model Formal**
   Dalam model ini, teknik formal seperti logika formal atau model matematika digunakan untuk memverifikasi dan memvalidasi kebutuhan. Metode ini melibatkan analisis formal yang ketat dan dapat membantu menemukan cacat logika atau kelemahan dalam kebutuhan.

4.6 Manajemen Kebutuhan

Memproduksi produk perangkat lunak yang sesuai dengan kebutuhan pemangku kepentingan serta memiliki kualitas yang baik, merupakan tantangan yang paling besar dalam rekayasa kebutuhan perangkat lunak. Buruknya pendefinisian kebutuhan perangkat lunak serta seringnya terjadi perubahan pada kebutuhan tersebut adalah salah satu penyebab jangka waktu pengerjaan proyek perangkat lunak semakin bertambah panjang (Baruah, 2015). Manajemen kebutuhan adalah salah satu kegiatan paling penting untuk keberhasilan pengiriman perangkat lunak dan siklus hidup proyek. Tim pengembangan perangkat lunak yang mengikuti konsep *agile software development* menyambut baik perubahan kebutuhan dalam fase apa pun selama siklus pengembangan perangkat lunak produk. Tabel 4.1 menunjukkan langkah-langkah manajemen kebutuhan dalam berbagai metodologi pengembangan perangkat lunak berbasis *agile.*

Tabel 4.2. Manajemen Kebutuhan Perangkat Lunak berdasarkan Metodologi Pengembangannya (Baruah, 2015)

No	Jenis Metodologi	Deskripsi
1	Extreme Programming	Mendefinisikan kebutuhan berdasarkan *user stories* dan *on-sites customer* atau pelanggan di tempat. *User stories* memiliki dua komponen, yakni **kartu tertulis** dalam dan **percakapan** setelah kartu tertulis. Kartu tertulis hanyalah sebuah '' janji untuk percakapan''. Kartu tertulis tidak perlu lengkap atau

No	Jenis Metodologi	Deskripsi
		dinyatakan dengan jelas. Kartu tertulis boleh dimusnahkan setelah kebutuhan telah didefinisikan
2	Scrum	Metode ini juga mendefinisikan kebutuhan aktual perangkat lunak berdasarkan *user stories.* Product owner memiliki peran penting dalam pengembangan perangkat lunak
3	Feature Driven Development (FDD)	Mengumpulkan kebutuhan pengguna lalu selanjutnya merepresentasikannya ke dalam diagram UML dengan daftar fitur. Daftar fitur mengelola kebutuhan fungsional dan tugas pengembangan. Analisis kebutuhan yang solutif dimulai dengan pemeriksaan tingkat tinggi dari ruang lingkup sistem & konteksnya. Tim menilai domain secara detail untuk setiap area pemodelan. Kelompok kecil menyusun model untuk setiap domain dan mempresentasikan model untuk tinjauan sejawat
4	Lean Software Development	Pengumpulan kebutuhan pengguna dilakukan dengan

No	Jenis Metodologi	Deskripsi
		menghadirkan layar ke pengguna akhir dan mendapatkan masukan mereka. Ideologi produksi *just-in-time* diterapkan untuk mengenali persyaratan dan lingkungan khusus
5	Adaptive Software Development	Pengumpulan kebutuhan dilakukan dalam fase spekulatif. Pertama, menetapkan misi dan tujuan proyek, memahami batasan, menetapkan organisasi proyek, mengidentifikasi dan menguraikan kebutuhan, membuat perkiraan ruang lingkup awal & mengidentifikasi risiko proyek utama
6	Kanban	*User Stories* digunakan untuk membantu memahami apa tujuan sebenarnya dari sebuah sprint. Sprint berisi satu kartu cerita. *User Stories* dipecah menjadi bagian-bagian yang lebih kecil. Sebuah cerita dibagi menjadi tugas sisi klien & sisi server. Tugas dibagi menjadi sub-tugas. Pengembang meminimalkan jumlah item dalam sprint untuk

No	Jenis Metodologi	Deskripsi
		mempertahankan waktu proyek
7	Agile Unified Process (AUP)	Fase kebutuhan termasuk mengidentifikasi pemangku kepentingan, memahami masalah pengguna, menetapkan dasar estimasi & mendefinisikan antarmuka pengguna untuk sistem. Aktivitas terjadi selama fase Inception & Elaboration tetapi berlanjut melalui fase untuk meningkatkan desain yang sedang berlangsung. Pada fase konstruksi, *user stories* diimplementasikan & dikerjakan ulang secara iteratif untuk mencerminkan pemahaman tentang domain masalah saat proyek berlangsung
8	Dynamic System Development Method (DSDM)	Kebutuhan untuk proyek tertentu dikumpulkan & diperiksa untuk kelayakan & penentuan prioritas

Manajemen kebutuhan perangkat lunak merupakan bagian kritis dari proses pengembangan perangkat lunak. Dengan mengelola kebutuhan dengan baik, tim pengembangan dapat meningkatkan peluang kesuksesan proyek, mengurangi risiko kegagalan, dan memastikan bahwa perangkat lunak yang dihasilkan sesuai dengan harapan dan kebutuhan pengguna.

Manajemen kebutuhan, terlepas dari metodologi pengembangan perangkat lunak apa pun dilakukan untuk seumur hidup sistem. Terlihat di industri perangkat lunak saat mengembangkan produk perangkat lunak apa pun bahwa kebutuhan sering berubah dari sisi pelanggan yang menjadi sulit bagi pengembang perangkat lunak untuk menghasilkan perangkat lunak yang berkualitas. Tidak ada pendekatan yang tepat untuk mengelola persyaratan yang sering berubah selama siklus pengembangan perangkat lunak. Memenuhi kebutuhan pelanggan sangat penting. Metodologi pengembangan perangkat lunak *Agile* mendukung perubahan kebutuhan. Jadi, metodologi pengembangan perangkat lunak *Agile* yang berbeda dipelajari, untuk memberikan gambaran tentang bagaimana praktik manajemen kebutuhan dilakukan untuk pengembangan perangkat lunak. Manajemen kebutuhan memastikan bahwa kebutuhan pelanggan selalu didengar di semua titik dalam proses pengembangan dan pada saat yang sama menjaga integritas informasi tersebut untuk kehidupan sistem dan juga sehubungan dengan perubahan sistem dan lingkungannya.

DAFTAR PUSTAKA

Baruah, N. 2015. Requirement Management in Agile Software Environment. *The 2015 International Conference on Soft Computing and Software Engineering (SCSE)* (hal. 81-83). Procedia Computer Science. doi:https://doi.org/10.1016/j.procs.2015.08.414

Belfo, F. 2012. People, Organizational and Technological Dimensions of Software Requirement Specification. *CENTERIS 2012 - Conference on ENTERprise Information Systems / HCIST 2012 - International Conference on Health and Social Care Information Systems and Technologies* (hal. 310-318). Elsevier Ltd.

Burge, J. E. 2001. Knowledge Elicitation Tool Classification. Artificial Intelligence Research Group, Worcester Polytechnic Institute.

Davis, A. 1992. Operational prototyping: a new development approach. *IEEE Software*, 70-78.

Hofmann, H. F., & Lehner, F. 2001. Requirements Engineering as a Success Factor in Software Projects. *IEEE Software*.

Khan, S., Dulloo, A. B., & Verma, M. 2014. Systematic Review of Requirement Elicitation Techniques. *International Journal of Information and Computation Technology, 4*(2), 133-138.

Lamsweerde, A. v. 2000. Requirements engineering in the year 00. *Procreedings of the 22nd International Conference on Software Engineering - ICSE 00*, (hal. 5-19).

Lobo, L., & Arthur, J. D. 2005. Effective Requirements Generation: Synchronizing Early Verification & Validation, Methods and Method Selection Criteria. *ArXiv. /abs/cs/0503004*.

Object Management Group. 2011. Business Process Model and Notation (BPMN).

Object Management Group. 2017. Unified Modeling Language (UML). Diambil kembali dari https://www.omg.org/spec/UML/2.5.1/PDF

Pandey, D., Suman, U., & Ramani, A. K. 2010. An Effective Requirement Engineering Process Model for Software Development and Requirements Management. *2010 International Conference on Advances in Recent Technologies in Communication and Computing* (hal. 287-291). Kottayam, India: IEEE.

Ponsard, C., Massonet, P., Rifaut, A., & Molderez, J. F. 2005. Early verification and validation of mission critical systems. *Electronic Notes in Theoretical Computer Science*, 237-254.

Pressman, R. S. 2010. *Software Engineering: A Practitioner's Approach* (7th ed.). New York: McGraw-Hill.

Sommervile, I. 2011. *Software Engineering.* Boston: McGraw Hill.

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

BAB 5
PEMODELAN SISTEM

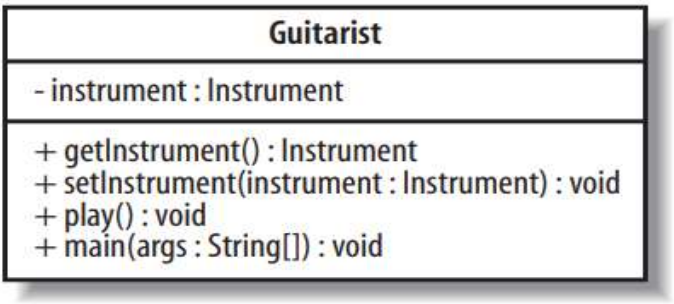
Oleh Elisawati

5.1 Pemodelan Sistem

Pemodelan (*Modelling*) adalah proses untuk Menyusun model, karena model adalah representasi lebih sederhana dari dunia nyata, maka penggunanya perlu memiliki bawah apa yang dibutuhkan sudah cukup di presentasikan. Misalnya apakah elemen variable yang penting telah dimasukkan, apakah Batasan model logis dan dapat diterima dan aspek – aspek lain. Ini membuat pemodelan menjadi harus secara disiplin menerapkan metodologi ilmiah untuk memastikan bahwa model yang disusun telah melalui sebuah proses baku.

Sedangkan Pemodelan sistem kegiatan teknologi yang menerjemahkan model aplikasi kedalam bangunan sistem operasional. Pemodelan sistem secara perincian spesifikasi yang menjelaskan bagaimana bagian-bagian sistem akan di wujudkan. Pemodelan sebuah sistem harus di arahkan kepada pemahaman terhadap keputusan apa yang dibutuhkan dalam sebuah proses pengambilan keputusan. Dengan demikian model menjadi lebih mudah dan terarah pengembangannya.

Bahasa model dapat terdiri dari pseudocode, kode nyata, gambar, diagram, atau bagian deskriptif yang panjang. Bahasa pemodelan ini banyak membantu dalam menggambarkan sistem. Unsur-unsur yang membentuk bahasa pemodelan disebut notasi.



Gambar 5.1. menunjukkan contoh notasi UML.

Namun, notasi bukanlah keseluruhan cerita. Tanpa diberitahu bahwa salah satu yang terdapat pada gambar 5.1. mewakili kelas dan anda bahkan tidak perlu tahu apa itu meskipun anda mungkin bisa menebak. Uraian tentang apa yang dimaksud dengan notasi adalah disebut semantik bahasa dan ditangkap dalam model meta bahasa.

5.2 Pengenalan UML

Pertama UML (*Unified Modeling Language* digunakan untuk mengkomunikasikan desain dengan jelas, menyampaikan desain dengan jelas dan memetakan solusi perangkat lunak. Namun, kenyataannya pemodelan sistem, bukan hanya pemodelan perangkat lunak, juga bisa manfaat dari bahasa terpadu seperti UML.

UML adalah notasi yang dihasilkan dari penggabungan OMT:

1. Teknik Pemodelan Objek OMT [James Rumbaugh 1991] - terbaik untuk sistem informasi analitis dan padat data.
2. Booch [Grady Booch 1994] - Sangat bagus untuk desain dan eksekusi. Grady Booch telah bekerja secara ekstensif dengan bahasa yang ada dan telah menjadi pemain kunci dalam pengembangan teknologi berorientasi objek untuk bahasa tersebut. Terlepas dari metode Booch, pelabelan

tidak dilakukan dengan baik (model didominasi oleh banyak bentuk awan - tidak terlalu bagus).

3. OOSE (Rekayasa Perangkat Lunak Berorientasi Objek [Ivar Jacobson 1992]) - menyajikan model yang dikenal sebagai kasus penggunaan. Use case adalah teknik yang ampuh untuk memahami perilaku seluruh sistem.

UML bukanlah bahasa pemrograman, melainkan sebuah alat yang menyediakan generator kode UML untuk beberapa bahasa pemrograman dan dapat membuat model rekayasa balik dari program yang ada. UML bukanlah bahasa yang sangat formal untuk membuktikan teorema. Ada beberapa bahasa seperti itu, tetapi tidak mudah dipahami atau digunakan untuk sebagian besar tujuan. UML digunakan untuk memodelkan sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi objek dan menciptakan bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin.

5.3 Sejarah UML

Bahasa berorientasi objek pertama umumnya dikenal sebagai Simula-67, yang dikembangkan pada tahun 1967. Bahasa ini tidak pernah memiliki pengikut yang signifikan, meskipun sangat mempengaruhi pengembang beberapa bahasa berorientasi objek selanjutnya. Gerakan berorientasi objek diaktifkan oleh ketersediaan Smalltalk secara luas pada awal 1980-an, diikuti oleh bahasa berorientasi objek lainnya seperti Objective C, C, Eiffel, dan CLOS. Penggunaan bahasa berorientasi objek sebenarnya pada awalnya terbatas, tetapi pendekatan berorientasi objek menarik banyak perhatian.

Upaya sebelumnya dilakukan untuk menghubungkan konsep antar metode. Contoh penting adalah Penggabungan Coleman dan rekan [Coleman-94], termasuk konsep OMT [Rumbaugh-91], Booch [Booch-91] dan CRC [Wirfs-Brock 90].

Karena tidak melibatkan penulis asli, maka harus dianggap baru, bukan pengganti metode yang sudah ada. Upaya sukses pertama untuk menggabungkan dan mengganti pendekatan yang ada dilakukan ketika Rumbaugh bergabung dengan Booch di Rational Software Corporation pada tahun 1994. Mereka mulai menggabungkan konsep dari metode OMT dan Booch, yang mengarah ke proposal pertama pada tahun 1995. Sekitar waktu yang sama, Jacobson juga bergabung Rasional dan mulai bekerja dengan Booch dan Rumbaugh. Kolaborasi mereka disebut *Unified Modeling Language* (UML).

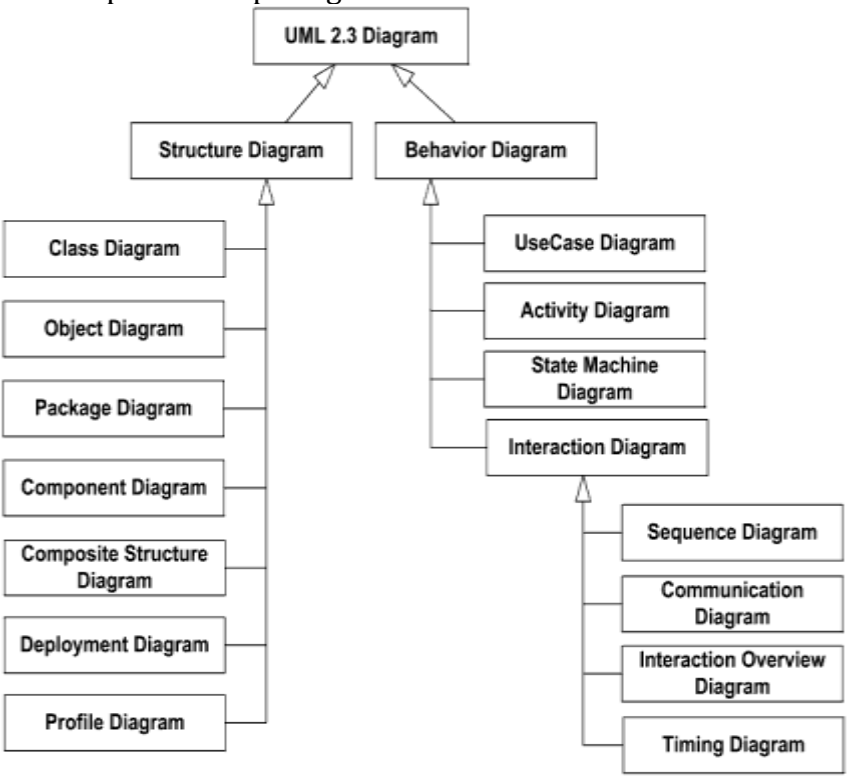
Pada tahun 1996, *Object Management Group* (OMG) mengusulkan pendekatan standar untuk pemodelan objek. Penulis UML Booch, Jacob son, dan Rumbaugh mulai bekerja dengan ahli metodologi dan pemrogram dari perusahaan lain untuk mengembangkan proposal yang menarik untuk keanggotaan OMG, seperti bahasa pemodelan yang akan diterima secara luas oleh pembuat alat, ahli metodologi, dan pemrogram yang akan menjadi pengguna akhir. UML secara resmi dimulai pada Oktober 1994 ketika proyek berfokus pada penggabungan metode Booch dan Rumbaugh (OMT). Pada Oktober 1995, UML merilis versi 0.8 dan pada saat yang sama Jacobson bergabung dengan Relational. Juga, ruang lingkup UML berkembang.

UML adalah bahasa yang digunakan untuk membangun, memvisualisasikan, mendefinisikan dan mendokumentasikan sistem informasi. Grady Booch, Jim Rumbaugh, dan Ivar Jacobson mengembangkan UML sebagai alat untuk desain dan analisis berorientasi objek.

UML menyediakan standar pemodelan sistem, seperti bahasa lain, UML mendefinisikan notasi dan sintaks/semantik. Notasi UML adalah seperangkat format khusus untuk menggambarkan diagram perangkat lunak yang berbeda. Setiap bentuk memiliki arti tertentu, dan sintaks UML menentukan bagaimana bentuk dapat digabungkan.

5.4 Diagram UML

UML 2.3 berisi 13 diagram berbeda yang dikelompokkan ke dalam tiga kategori. Pembagian kategori dan berbagai jenis huruf dapat dilihat pada gambar di bawah ini.



Gambar 5.2. Diagram UML

5.4.1 *Use Case Diagram*

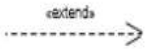
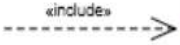
Pemodelan *use case* adalah pemodelan struktural yang berkaitan dengan pemodelan fungsionalitas suatu sistem. Biasanya menerapkan pemodelan *use case* selama aktivitas persyaratan untuk menangkap persyaratan yang menentukan apa yang harus dilakukan sistem. Pemodelan *use case* biasanya

di mulai di awal proyek dan berlanjut sepanjang proses pengembangan sistem. Hal ini biasanya dilakukan sebagai rangkaian lokakarya antara pengguna dan analis sistem di mana ide dapat dieksplorasi dan persyaratan dapat disempurnakan dari waktu ke waktu.

Ada dua hal utama tentang *use case*, yaitu definisi aktor dan *use case*.

1. Aktor adalah orang yang menjelaskan siapa yang berinteraksi dengan sistem, aktor memberikan informasi kepada sistem, ketika informasi diterima oleh sistem, keduanya dapat terjadi pada waktu yang bersamaan. Meskipun lambang seorang aktor adalah gambaran seseorang, namun aktor tersebut belum tentu merupakan orang.
2. *Use case* adalah sistem yang digambarkan dalam bentuk kotak. Perannya adalah untuk membatasi use case pada interaksi non-sistem.

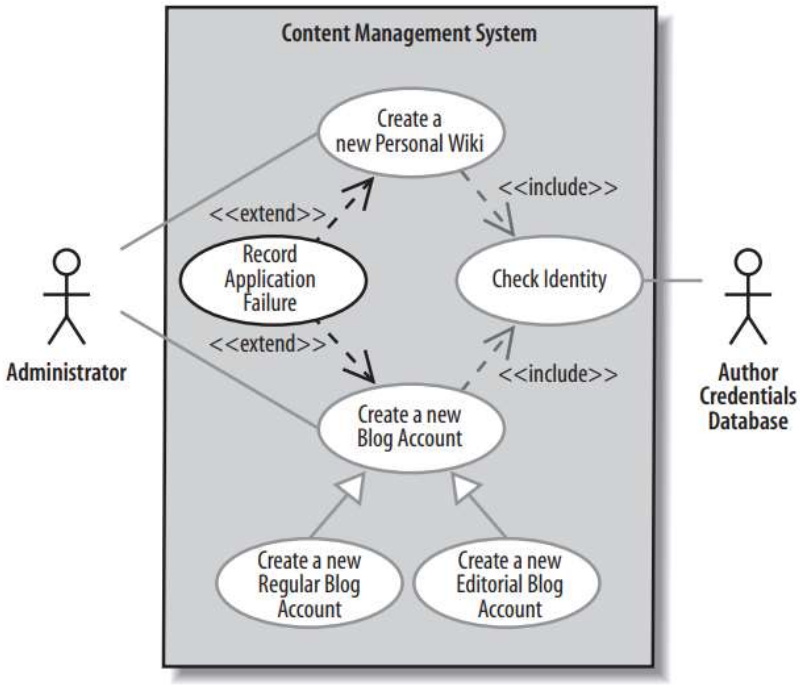
Tabel 5.1. Simbol-simbol *Use Case*

	*ACTOR* Orang proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari actor adalah gambar orang, biasanya dinyatakan menggunakan kata benda di awal frase nama *actor*.
	*USE CASE* Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesar antar unit atau actor biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama use case.
	ASOSIASI/*ASSOCIATION* Komunikasi antara *actor* dan use case yang berpartisipasi pada use case atau use case memiliki interraksi dengan *actor*.
	EKSTENSI/*EXTEND* Relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan dapat berdiri sendiri walau tanpa use case tambahan memiliki nama depan yang sama dengan use case yang di tambahkan.
	GENERALISASI/*GENERALIZATION* Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah use case dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya.
	MENGGUNAKAN/*INCLUDE* Relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsional atau sebagai syarat dijalankan use case ini.

Menurut persyaratan, perilaku yang dapat digunakan bersifat opsional, kedua kasus; Anda tidak ingin mencatat penolakan jika aplikasi untuk akun blog atau Wiki pribadi diterima. Hubungan <<extend>> ideal untuk jenis ini

menggunakan kembali situasi, seperti yang ditunjukkan pada Gambar 5.3.

Gambar 5.3. Relasi <<extend>> ikut bermain untuk menunjukkan bahwa "Buat yang baru Kasus penggunaan Wiki Pribadi" dan "Buat Akun Blog baru" kadang-kadang dapat dibagikan perilaku perekaman penolakan aplikasi.



Gambar 5.3. Contoh Use Case Diagram

5.4.2. *Activity Diagram*

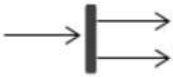
Grafik kinerja membantu Anda menentukan bagaimana sistem Anda mencapai tujuannya. Diagram aktivitas menunjukkan aktivitas bersama untuk mewakili proses di sistem Anda. Misalnya, Anda dapat menggunakan bagan alur

untuk memodelkan langkah-langkah dalam membuat akun blog.

Diagram aktivitas sangat cocok untuk memodelkan proses bisnis. Proses bisnis adalah urutan tugas yang terkoordinasi untuk mencapai tujuan bisnis, seperti mengirimkan pesanan pelanggan. Beberapa alat manajemen proses bisnis (BPM) memungkinkan proses bisnis didefinisikan menggunakan diagram aktivitas atau notasi grafis. Ini memungkinkan Anda untuk mengatur dan menjalankan, misalnya, proses persetujuan pembayaran di mana salah satu langkahnya adalah meminta layanan persetujuan kartu kredit online menggunakan notasi grafis yang nyaman seperti grafik aktivitas.

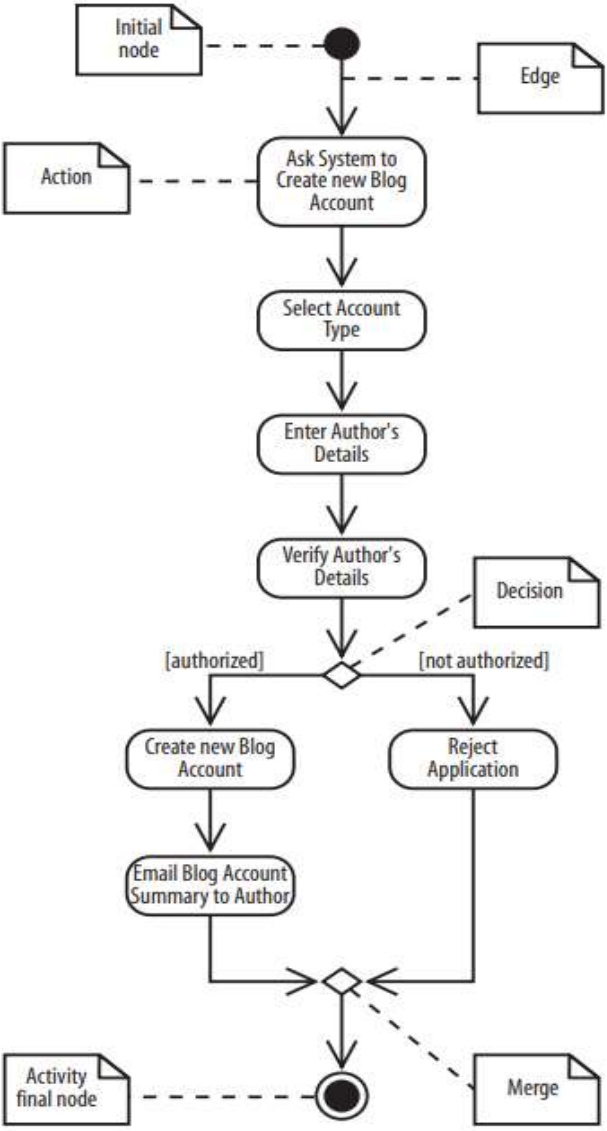
Activity diagram adalah salah satu diagram UML yang paling mudah diakses karena mereka menggunakan simbol yang mirip dengan notasi flowchart yang dikenal luas, membuatnya berguna untuk menggambarkan proses ke khalayak luas. Faktanya, diagram aktivitas berakar pada diagram alur, seperti halnya diagram status UML dan diagram aliran data.

Tabel 5.2. Simbol-simbol *Activity Diagram*

	STATUS AWAL/*INITIAL* Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah satutus awal.
	AKTIVITAS/ *ACTIVITY* Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
	*PERCABANGAN / DECISION* Asosiasi percabangan dimana lebih dari satu aktivitas digabungkan menjadi satu.
	PENGGABUNGAN/ *JOIN* Asosiasi penggabungan dimana lebih dari satu aktivitas lebih dari satu.
	STATUS AKHIR/ *FINAL* Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status satu.
	*SWIMLINE* Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi.

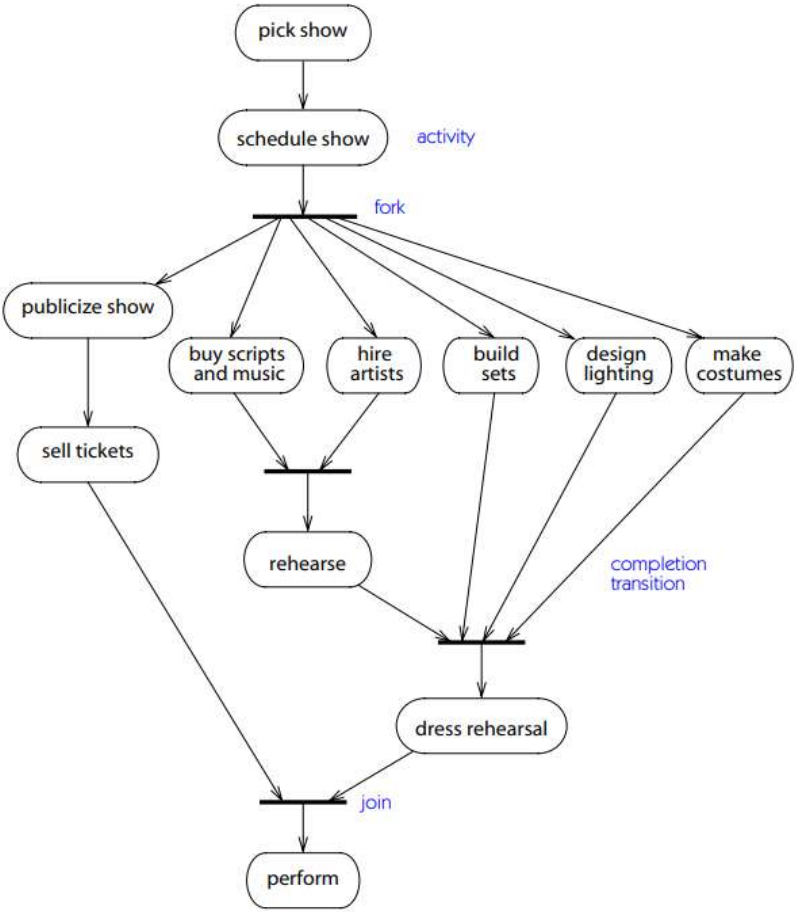
Gambar 5.4 menunjukkan proses pembuatan akun blog dalam notasi diagram aktivitas. Sebuah diagram aktivitas membantu untuk memvisualisasikan kasus penggunaan dengan lebih baik. Diagram ini merupakan aktivitas yang memodelkan perilaku dinamis dengan fokus pada proses, dasar

elemen diagram aktivitas ditampilkan dalam proses pembuatan akun blog.



Gambar 5.4. Contoh *Activity Diagram*

Contoh lain dapat di lihat di Gambar 5.5 menunjukkan diagram aktivitas untuk box office. Diagram ini menunjukkan kegiatan yang terlibat dalam pemasangan pertunjukan. Tanda panah menunjukkan ketergantungan berurutan—misalnya, pertunjukan harus dipilih sebelum dijadwalkan. setelah pertunjukan dijadwalkan, teater dapat dimulai mempublikasikannya, membeli skrip, mempekerjakan seniman, membangun set, merancang pencahayaan, dan membuat kostum, semuanya secara bersamaan. Namun, sebelum latihan dapat dimulai, skrip harus sudah siap dipesan dan artis harus dipekerjakan.

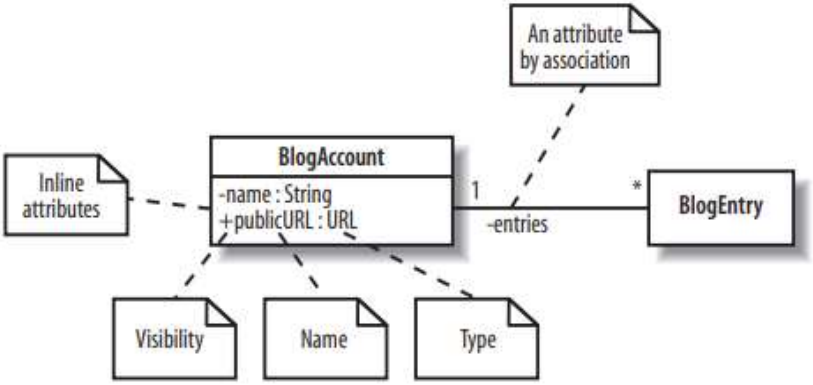


Gambar 5.5. Contoh Activity Diagram

5.4.3 *Class Diagram*

Diagram kelas adalah inti dari sistem berorientasi objek, sehingga diagram UML yang paling populer adalah diagram kelas. Struktur sistem terdiri dari kumpulan potongan-potongan, sering disebut objek. Kelas menggambarkan berbagai jenis objek yang dapat dimiliki sistem, dan diagram kelas menunjukkan kelas terkait.

Atribut kelas adalah potongan informasi yang mewakili keadaan objek. Atribut ini dapat direpresentasikan pada diagram kelas dengan menempatkannya di dalam bagian mereka dari kotak kelas dikenal sebagai atribut sebaris atau dengan asosiasi kelas lain, seperti yang ditunjukkan pada Gambar 6.6.



Gambar 5.6. *Class Blog Account*

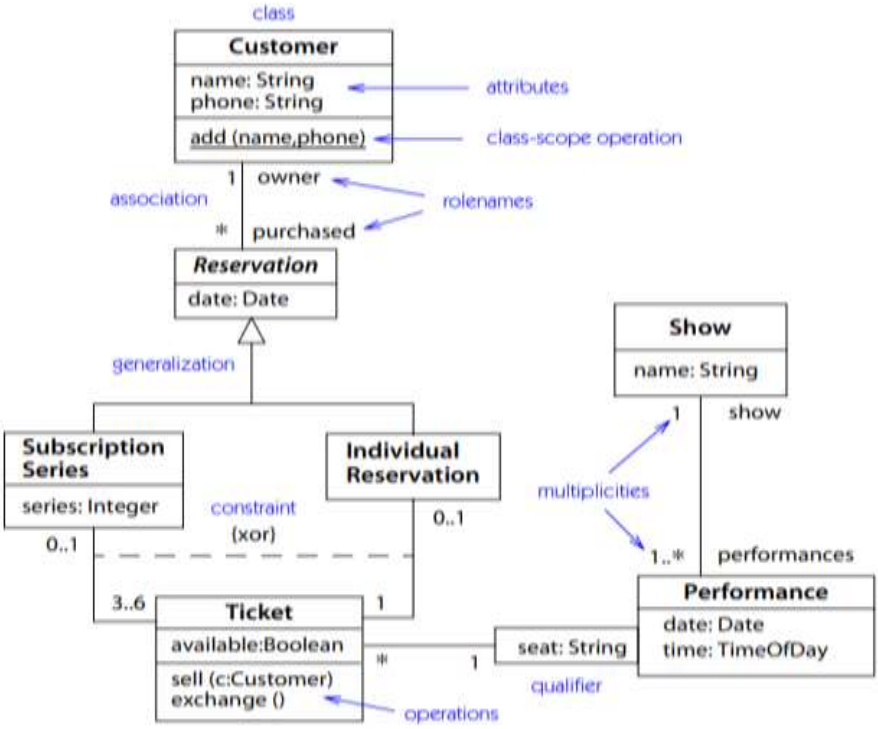
Pada Gambar 5.6. tidak masalah jika mendeklarasikan inline atau atribut terkait. Minimal atribut biasanya akan memiliki tanda tangan yang berisi properti visibilitas, nama, dan jenis, meskipun nama atribut adalah satu-satunya bagian dari tanda tangannya yang benar-benar harus ada agar kelas menjadi valid.

Tabel 5.3. Simbol-simbol *Class Diagram*

GAMBAR	NAMA	KETERANGAN
	Generalization	Hubungan dimana objek anak *(descendent)* berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
	*Nary Association*	Upaya untuk menghindari asosiasi dengan lebih dari 2 objek.
	*Class*	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
	*Collaboration*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu actor
	Realization	Operasi yang benar-benar dilakukan oleh suatu objek.
	Dependency	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempegaruhi elemen yang bergantung padanya elemen yang tidak mandiri
	Association	Apa yang menghubungkan antara objek satu dengan objek lainnya

Gambar 5.7 menunjukkan class diagram berisi bagian dari model domain penjualan tiket. Ini menunjukkan beberapa kelas penting, seperti Pelanggan, Reservasi, Tiket, dan Kinerja. Pelanggan mungkin memiliki banyak reservasi, tetapi setiap

reservasi dilakukan oleh satu pelanggan. Reservasi adalah dari dua jenis: seri langganan dan reservasi individu. Keduanya memesan tiket dalam satu kasus, hanya satu tiket, dalam kasus lain, dan beberapa tiket. Setiap tiket adalah bagian dari seri langganan atau reservasi individu, tetapi tidak keduanya. Setiap pertunjukan memiliki banyak tiket yang tersedia, masing-masing dengan nomor kursi yang unik. Pertunjukan dapat diidentifikasi dengan acara, tanggal, dan waktu.



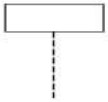
Gambar 5.7. Contoh Class Diagram

5.4.4 *Sequence Diagram*

Sequence diagram adalah anggota penting dari kelompok yang dikenal sebagai diagram interaksi. Diagram interaksi memodelkan interaksi runtime kritis antara komponen sistem yang membentuk bagian dari tampilan logis model.

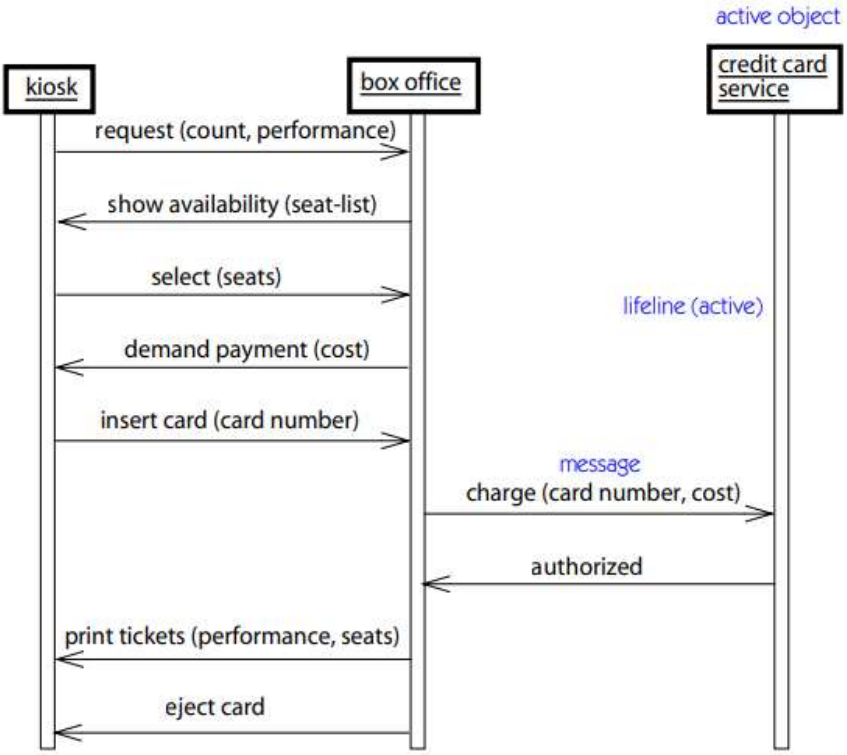
Sequence Diagram merupakan urutan interaksi antara bagian-bagian sistem dengan menggunakan diagram urutan, dan dapat menggambarkan interaksi mana yang akan terjadi dan ketika kasus penggunaan tertentu dijalankan bagaimana urutan interaksi tersebut akan terjadi. Diagram urutan menunjukkan banyak informasi lain tentang interaksi, dan efektif untuk menyampaikan perintah peristiwa dalam interaksi tersebut.

Tabel 5.4. Simbol-simbol *Sequence Diagram*

GAMBAR	NAMA	KETERANGAN
	*LifeLine*	Objek *entity*, antarmuka yang saling berinteraksi.
	*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi
	*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi

Gambar 5.8 menunjukkan sequence diagram untuk kasus penggunaan pembelian tiket. Kasus ini diprakarsai oleh pelanggan di kios berkomunikasi dengan box office. Dengan

penggunaan biaya termasuk dalam urutan, yang melibatkan komunikasi dengan kios dan layanan kartu kredit. Sequence diagram ini masih dalam tahap awal pengembangan dan belum menunjukkan secara utuh.



Gambar 5.8. Sequence Diagram

DAFTAR PUSTAKA

A.S, R., & Shalahuddin, M. 2011. Modul Pembelajaran Rekayasa Perangkat Lunak (Terstruktur dan Berorientasi Objek). Bandung: Modula.

Hidayatno Akhmad. 2022. *Prinsip dan Konsep Pemodelan Sistem.* Jakarta: Universitas Indonesia.

James, R., Ivar, J., & Grady, B. 1998. *The Unified Modeling Language Reference Manual.* Canada: Addison Wesley Longman, Inc.

russ, M., & Kim, H. 2006. *Learning UML 2.0.* United States of America: O'Reilly Media.

Simarmata, J. 2010. *Rekayasa Perangkat Lunak.* Yogyakarta: Andi Yogyakarta.

BAB 6
ARSITEKTUR PERANGKAT LUNAK

Oleh Rudiansyah

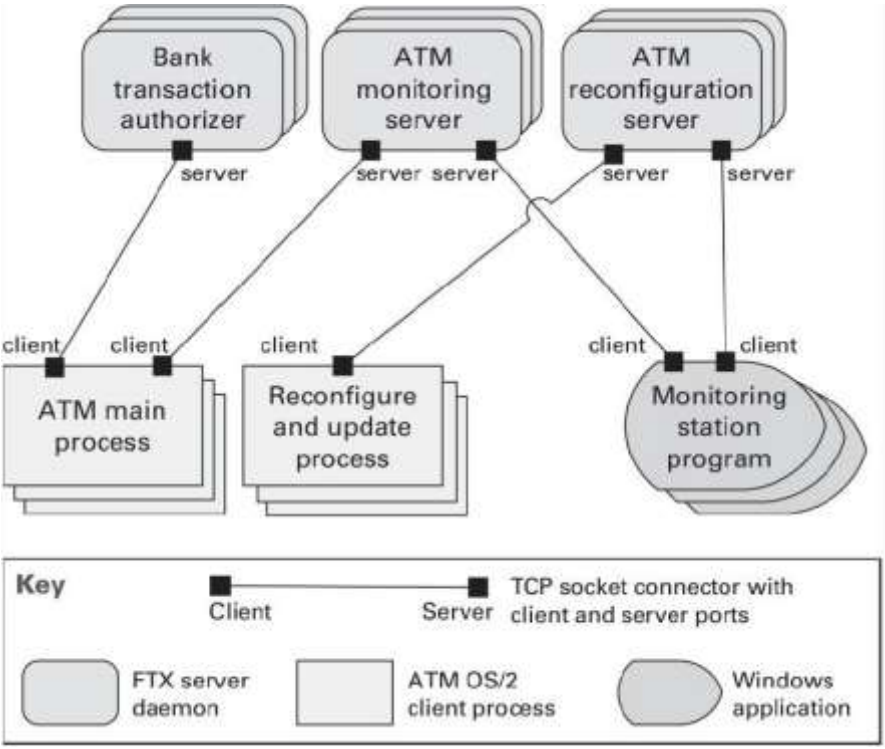
6.1 Pendahuluan

Ragam (*style*) arsitektural adalah suatu kekhususan dari jenis elemen dan relasi, bersama dengan himpunan pembatasan bagaimana semua itu dapat digunakan (Clements, 2011).

Pada sisi lain, pola (*pattern*) arsitektural mengekspresikan suatu skema organisasi struktural yang fundamental bagi sistem *software* (Clements, 2011).

6.2 Client Server

Komponen-komponen pada *style client-server* berinteraksi dengan meminta layanan dari komponen lain. Requester (yang meminta layanan disebut sebagai *client*, dan *service provider* disebut sebagai server, yang menyediakan sejumlah layanan melalui satu atau lebih portnya, client menginisiasi interaksi, meminta layanan yang dibutuhkkan dari server dan menunggu hasil dari permintaan tersebut dimana client terkoneksi ke server melalui konektor request/response, komponen server dapat menjadi client bagi server-server lain. Contoh umum dari sistem bergaya *client-server* adalah sistem informasi yang berjalan pada jaringan lokal, dimana klienya adalah aplikasi GUI (seperti *Visual Basic*) dan server adalah sistem manajemen database (seperti *oracle*), aplikasi berbasis web dan server adalah komponen yang berjalan pada server web (seperti *Tomcat atau Apache*). (Clements, 2011).



Arsitektur *client-server* ATM sistem Perbanka, menggunakan notasi resmi.

6.2.1 *Service-Oriented Architecture* (SOA)

Service-Oriented Architecture (SOA) terdiri dari kumpulan komponen terdistribusi yang menyediakan dan / atau mengkonsumsi jasa (*service*). Dalam gaya ini, komponen penyedia layanan dan komponen konsumen (pelanggan) layanan dapat menggunakan bahasa implementasi dan platform yang berbeda. Sebagian besar layanan bersifat mandiri. Komputasi dicapai dengan satu set komponen yang bekerja sama untuk menyediakan atau mengkonsumsi layanan melalui jaringan. (Clements, 2011).

6.2.2 SOA : Elemen -elemen

Elemen-elemen SOA yaitu:

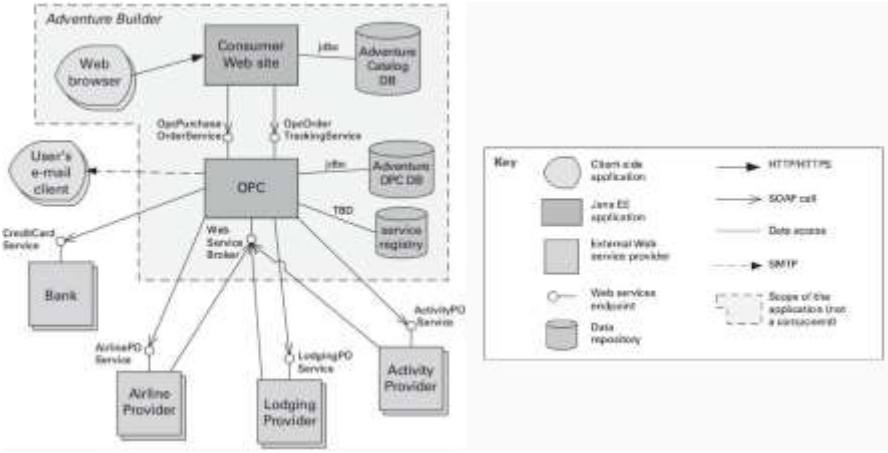
1. Penyedia (pemasok,provider) layanan: menyediakan satu atau lebih layanan melalui antarmuka yang diterbitkan (publish)
2. Properti-properti dapat berbeda berdasarkan teknologi terapannya (seperti EJB atau ASP.NET) tetapi mencakup kinerja, kendala otorisasi, ketersesiaan dan biaya
3. Pelanggan layanan (konsumer, customer): memanggil layanan langsung atau melalui perantara.
4. Simple *Object Acces Protocol* (SOAP) connector, yang menggunakan protokol SOAP untuk komunikasi sinkron antar Web Services, biasanya memanfaatkan HTTP.
5. Port dari komponen yang menggunakan SOAP sering dideskripsikan dalam WSDL
6. Konektor *Representational State Transfer* (REST), yang bersandar pada operasi request/response dasar dari protokol HTTP.

6.2.3 SOA : Ketentuan khusus

1. Konsumen layanan terhubung ke penyedia layanan, namun komponen perantara (seperti ESB, registry, atau server BPEL) dapat pula digunakan
2. ESB mengakibatkan hadirnya topologi hub-and-spoke
3. Penyedia service juga dapat menjadi pelanggan dari service
4. Pola SOA spesifik menghadirkan ketentuan khusus tambahan

6.2.4 SOA : Contoh Adventure Builder 2010

1. Sistem ini diambil dari dokumen arsitektur perangkat lunak contoh yang menyertai buku ini secara online, di wiki.sei.cmu.edu/sad
2. Sistem *Adventure Builder* (*Adventure Builder* 2010) berinteraksi melalui layanan Web SOAP dengan beberapa penyedia layanan eksternal
3. Penyedia service eksternal dapat berupa mainframe, Java atau NET sifat komponen eksternal ini transparan karena konektor SOAP menyediakan interoperabilitas yang diperlukan



Gambar 6.1. Diagram pandangan SOA dari Adventure Builder 2010, menggunakan notasi resmi
(Sumber : Clements et al, 2011)

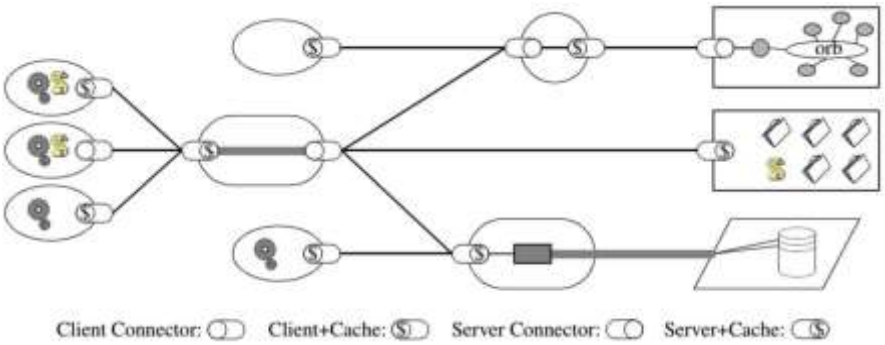
6.3 *Representational State Transfer* (REST)

Representational State Transfer (REST) adalah architectural style perangkat lunak dari *World Wide Web, REST* merupakan suatu *architectural style* bagi sistem hypermedia terdistribusi, menggambarkan prinsip-prinsip software

enggineering yang mengarahkan REST dan ketentuan interaksi terpilih untuk mempertahankan prinsip-prinsip tersebut, sekaligus membedakan mereka dari ketentuan architectural styles lain. Sejak 1994 REST telah digunakan untuk memandu rancangan dan pengembangan dari arsitektur web modern. Dikerjakan sebagai bagian kerja dari *Internet Engineering Task Force* (IETF) dan World Wide Web Consortium (W3C) untuk mendifinisikan standar arsitektural bagi Web : *Hypertext Transfer Protocol* (HTTP), *Uniform Resource Identifier* (URI), dan *Hyper Text Markup Language* (HTML)

6.4 REST: RESTful

Aplikasi yang dibangun mengikuti ketentuan REST disebut RESTful, sistem RESTful biasanya berkomunikasi diatas HTTP dengan metode sama (GET, POST, PUT, DELETE, dll) yang digunakan web browser untuk *me-retrieve* halaman web dan mengirimkan data ke server remote, sistem REST menyediakan antarmuka dengan sistem eksternal sebagai sumber daya web yang diidentifikasi dengan URI. Contoh Operasi dapat menggunakan metode standar adalah seperti GET/People/1 atau DELETE/people/1.



Gambar 6.2. REST Style (Fielding, 2000)

6.5 Microservices

Microservices sebagai konsep inti adalah yang penting untuk dipahami dengan pola ini adalah gagsan dari komponen layanan (*service component*). Komponen layanan mengandung satu atau lebih modul misalkan kelas java yang mewakili suatu fungsi bertujuan tunggal misalkan memberikan informasi cuaca untuk kota/daerah tertentu atau bagian terpisah dari aplikasi bisnis besar misalkan informasi posisi perdagangan saham atau penentuan tarif asuransi mobil. Aplikasi yang dibangun menggunakan pola arsitektur layanan mikro (*mikroservices*) umumnya lebih kuat, menyediakan skalabilitas yang lebih baik dan dapat lebih mudah mendukung pengiriman berkelanjutan (*continuous delivery*).

6.6 Microservices: Arsitektur Terdistribusi

Konsep kunci lain dalam pola arsitektur microservices adalah arsitekturnya terdistribusi, berarti bahwa semua komponen dalam arsitektur sepenuhnya terpisah (decoupled) dari satu sama lain dan diakses melalui protokol akses remote (misalnya, JMS, AMQP, REST. SOAP, RMI, dll).

Tujuan terdistribusi dari pola arsitektur ini adalah untuk memperoleh skalabilitas dan karakteristik penyebaran yang tinggi (superiror).

6.7 *Internet of Things* (IoT)

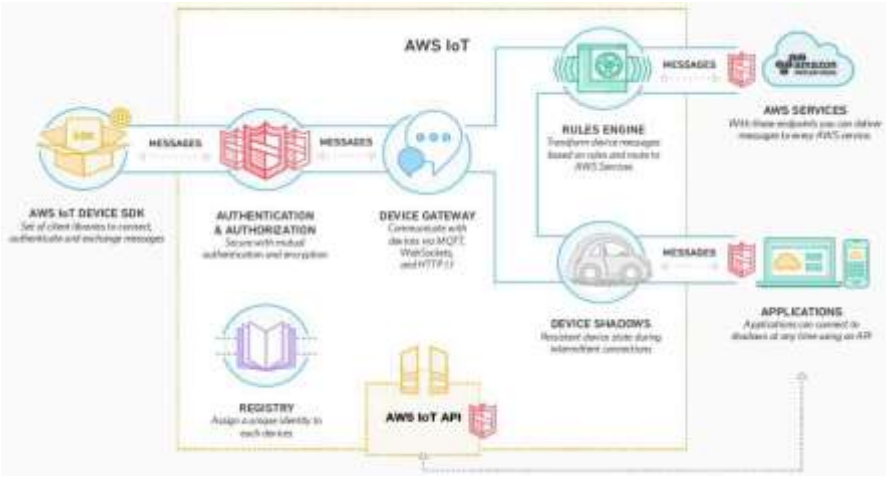
Istilah *internet of things* (IOT) pertama kali digunakan pad atahun 1999 oleh Kevin Ashton yang bekerja pada standar penandaan (*tagging*) obejk menggunakan RFID untuk aplikasi logistik, namun ide komputasi dimana-mana (*ubiquitous computing*) sudah ada sejak akhir 1980-an, sejak itu itu para peneliti telah bekerja pada banyak sistem yang fokus pada tag dan sensor, middleware dan teknologi awan, dan jaringan komunikasi.

Internet of things (IOT) berurusan dengan fungsi inovatif dan produktivitas yang lebih baik dengan menghubungkan perangkat-perangkat dengan lancar (*seamlessly*) yang akan meningkatkan sejumlah besar inovasi, efisiensi dan kualitas, menghubungkan bidang produksi, kesehatan, otomotif atau sistem transportasi dengan sistem Teknologi Informasi. Informasi bisnis penting akan memberikan nilai yang sangat besar untuk organisasi. Perusahaan Teknologi Informasi besar seperti CISCO dan SAP memperkirakan ada miliaran perangkat jaringan dan layanan bisnis alam semesta berbasis Teknologi Informasu dengan ekspektasi triliunan dolar.

6.8 *Internet of Things* (IoT): Marketplace

Sebuah tonggak yang sangat terlihat dicapai ketika IETF merilis Ipv6, protokol yang memungkinkan IOT. Baru – baru ini ,IOT telah menerima usulan keterlibatan dan kerja komersial pada referensi arsitekturnya. Ini didukung oleh industri ndustri besar; Google mengumumkan Brillo sebagai OS untuk perangkat IoT dalam rumah pintar (*smart home*), banyak perangkat komersial yang tersedia telah menggunakan standard komunikasi *machine to machine* (m2m) seperti bluetooth, zigbee dan *low-power wi- fi*, dalam perkembangannya Microsoft juga telah mengumumkan bahwa Windows 10 akan mendukung sistem embedded untuk mikro-kontroller yang beredar luas seperti Raspberry Pi 2 dan 3

Samsung dan perusahaan lainnya telah mengumumkan generasi baru chip mereka untuk perangkat pintar, telah banyak laporan implementasi menjelaskan bagaimana mikro kontroller jaringan yang difungsikan sebagai hub untuk sensor aktuator dan penandaan (tagging). Pada Tahun 2015 amazon merilis layanan cloud untuk IoT, AWS IoT.



Gambar 6.3. AWS IoT

DAFTAR PUSTAKA

Clements, D. H., Sarama, J., Spitler, M. E., Lange, A. A., & Wolfe, C. B. 2011. Mathematics learned byyoung children in an intervention based on learning trajectories: A large-scale cluster randomized trial. Journal for Research in Mathematics Education, 42(2), 127–166.

Fielding, Roy Thomas. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. University of California : Irvine.

Daft L, Richard. 2015. Manajemen. Jakarta : Erlangga

BAB 7
DESAIN DAN IMPLEMENTASI PERANGKAT LUNAK

Oleh Amna

7.1 Desain Perangkat Lunak

Desain sistem perangkat lunak merupakan proses yang dilakukan dalam mengubah kebutuhan pengguna dalam beberapa bentuk yang tepat sehingga dapat membantu *programmer* melakukan *coding* dan implementasi sistem (Jauhari *et al*., 2022). Sebelum melakukan desain perangkat lunak, diperlukan analisis kebutuhan pada pengguna dengan cara mencari permasalahan yang dimiliki. Selanjutnya, masalah-masalah tersebut tidak akan difokuskan keseluruhannya melainkan akan kembali kepada permintaan pengguna. Permasalahan yang ada akan dicari solusi yang tepat untuk menyelesaikannya. Setelah mendapatkan solusi yang paling baik, maka desain akan dapat dieksekusi dalam bentuk prototipe awal.

7.2 Teknik Terjemahan Analisis ke Desain

Secara garis besar terdapat tiga teknik untuk mendapatkan data bagi tujuan terjemahan dari analisis ke desain yang dapat dilakukan pada perangkat lunak. Teknik wawancara yaitu proses yang dilakukan dengan memberikan pertanyaan tertentu dan mendapatkan jawaban dari narasumber yang di wawancara. Teknik ini bertujuan untuk memahami latar belakang, pandangan narasumber dan sikapnya terhadap sebuah permasalahan. Selain itu, terdapat teknik observasi yang merupakan cara mengumpulkan data

dengan menuliskan semua tingkah laku, tindakan serta lingkungan dari pengguna. Yang terakhir terdapat teknik kuesioner. Teknik kuesioner adalah teknik yang dilakukan dengan mengumpulkan data dengan melakukan pertanyaan tertulis yang diberikan kepada responden. Teknik ini selalu dimanfaatkan pada survei yang dilakukan kepada responden (Rachmad et al., 2023).

Informasi atau data yang di dapat dari teknik di atas, baik teknik wawancara, teknik observasi maupun teknik kuisioner akan di analisa secara mendalam oleh penganalisa sistem. Hasil analisa ini akan diterjemahkan kedalam berbagai bentuk model yang bersesuaian agar mudah di fahamai oleh pengimplementasi.

Menurut (Utami & Asnawati, 2015), dalam teknik analisis data dapat digambarkan menjadi sebuah pengembangan korelasi yang dapat diuraikan dalam beberapa interpretasi akibat yang disebabkan olehnya. Teknik ini disebut dengan nama model sebab-akibat (*causing modeling*). Diberikan penamaan ini karena analisis yang dilakukan dapat membantu pengguna untuk menguji proposisi teoritis dari hubungan sebab akibat tanpa perlu memanipulasi atau memberikan perlakuan pengukuran tertentu pada variabel-variabel. Model ini memberikan asumsi bahwa terdapat hubungan yang dekat antar satu variabel dengan variabel yang lain sehingga perkembangan pada jalur analisis dapat diperluas dan diperdalam menjadi sebuah bentuk analisis *Structural Equation Modelling* atau SEM.

7.3 Hal-hal yang Penting dalam Desain Perangkat Lunak

Menurut (Dwanoko, 2016), hal-hal penting yang harus dilakukan saat mendesain sebuah perangkat lunak mencakup membuat perincian mengenai apa saja yang akan dibutuhkan

dalam pengembangan sistem. Selain itu, diperlukan juga perencanaan untuk membangun proyek sistem nantinya. Tahapan desain dan pengembangan sistem akan membutuhkan desain konseptual, desain basis data awal dan sistem, desain dari detail input/output dari sistem informasi. Pada desain fungsi dan desain operasi akan dijelaskan secara detail tata letak layar, diagram proses yang akan dieksekusi dan dokumentasi lainnya.

Tahapan desain dapat menjelaskan sistem besar yang akan dibuat dalam bentuk beberapa subsistem yang lebih kecil sehingga desain perangkat lunak dapat lebih mudah untuk dimengerti dan lebih jelas untuk dilakukan saat akan dieksekusi.

7.4 Jenis-jenis Desain Perangkat Lunak

Menurut (Budiman, 2022) terdapat beberapa jenis desain sistem perangkat lunak. Desain logis merupakan gambaran abstrak pada aliran data, *input* dan *output* dari sistem. Desain membantu representasi dari *input* (sumber), *output* (tujuan), *database* (penyimpanan data), prosedur (aliran data) sesuai dengan kebutuhan dan yang diminta oleh pengguna.

Diperlukan analisis sistem yang lebih spesifik untuk membuat desain logis sehingga didapatkan aliran informasi yang masuk dan keluar dari sumber data yang dibutuhkan.

Desain fisik akan memproses *input* dan *output* sebenarnya dari sistem. Dengan fokus utama yaitu cara data masuk ke dalam sistem akan melalui proses verifikasi, diproses, dan akan ditampilkan keluarannya. Desain fisik memiliki hubungan dengan *interface*, desain proses dan desain data.

Desain arsitektur yaitu desain yang berfokus dengan desain arsitektur sistem dengan memberikan penggambaran

struktur dan perilaku dari sistem. Ia akan membantu membentuk struktur hubungan antara satu modul proses pengembangan dengan modul proses pengembangan lainnya pada sistem.

Terakhir terdapat desain rinci, dimana desain ini mengikuti desain arsitektur dan mengutamakan pengembangan pada setiap modul.

Pola desain arsitektur yaitu keseluruhan bentuk struktur dan organisasi dari sistem perangkat lunak. Ia dapat menunjukkan unsur-unsur sistem perangkat lunak seperti subsistem, komponen, kelas dan lainnya. Desain arsitektur juga dapat menjelaskan hubungan yang dimiliki antar unsur sesuai aturan dan pedoman yang harus diikuti untuk menentukan hubungannya. Pola desain lebih berfokus pada elemen desain yang detail yaitu seperti hubungan pada komponen atau mekanisme yang menjelaskan interaksi komponen ke komponen yang lain yang ada pada pola desain. Pola pengkodean (idiom) dapat merepresentasikan implementasi dari komponen yang digunakan dalam perangkat lunak serta metode yang ada pada interaksi antar komponen tersebut. Idiom dapat disebut sebagai pola pengkodean (Rianto, 2021).

7.5 Teknik Desain Perangkat Lunak

Dalam membuat perangkat lunak, dibutuhkan penggambaran desain sehingga tim pengembang dapat menerapkan desain menjadi nyata. *Unified Modelling Language* (UML) yaitu pemodelan yang akan menggambarkan dan mendokumentasikan perangkat lunak. Terdapat berbagai macam jenis UML yang dapat digunakan dalam merancang desain. UML dapat dibagi menjadi dua jenis yaitu diagram yang dapat menunjukkan perilaku dan diagram struktural dari perangkat lunak yang akan dikembangkan. Tidak semua UML akan digunakan dalam desain sehingga pengembang harus

memilih diagram yang akan dibutuhkan. UML yang paling umum digunakan yaitu *class diagram, use case diagram, sequence diagram dan activity diagram* (Raharjana & Justitia, 2015).

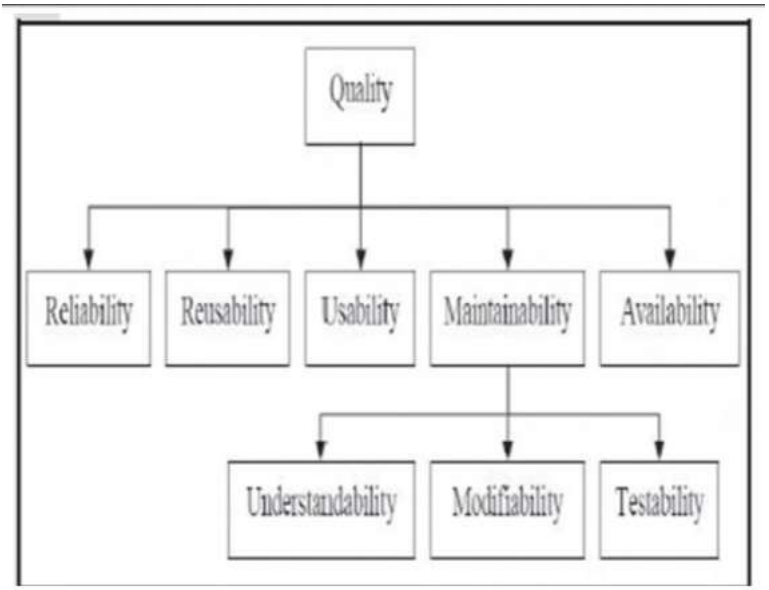
Menurut (Darni *et al.,* 2023), perancangan desain perangkat lunak dapat menjadi proses yang sulit dan kompleks. Untuk mengurangi kerumitan tersebut desain dapat dibagi menjadi beberapa bagian kecil yang akan dikerjakan satu per satu. Terdapat dua tingkat yang dapat dilakukan untuk membagi bagian perangkat lunak yang akan di desain yaitu tingkat atas dan tingkat bawah. Pada tingkat atas, desain akan dibagi menjadi beberapa bagian seperti antarmuka pengguna, logika bisnis dan basis data. Pada tingkat bawah, desain dapat dibagi menjadi bagian yang lebih kecil lagi seperti modular sehingga dapat diterapkan salah satu pola desain perangkat lunak. Setiap satu komponen modular akan dilakukan perancangan detail untuk menghubungkannya dengan komponen yang lain pada sistem.

7.6 Kualitas Desain Perangkat Lunak

Desain perangkat lunak dapat dikatakan baik jika ia dapat memenuhi beberapa atribut utama. Atribut ketangguhan yaitu desain mampu menerima inputan yang salah atau kesalahan dari pengembang maupun pengguna sehingga hal tersebut tidak menyebabkan sistem *crash*. Atribut fleksibilitas adalah cara perangkat lunak untuk beradaptasi dengan segala perubahan yang dapat terjadi di masa akan datang. Atribut *reuseable* dimana kode program dari perangkat lunak akan dapat digunakan juga di perangkat lunak lainnya. Atribut Efisiensi yang akan memperlihatkan bagaimana perangkat lunak dapat menyimpan data dari pengguna dan setiap fitur memberikan kemudahan kepada pengguna. Atribut keandalan yaitu dengan melihat berapa kali sistem melakukan kesalahan.

Atribut skalabilitas terkait erat dengan fleksibilitas yaitu kemudahan yang dapat dilakukan pengembang pada perangkat lunak untuk mengembangkannya atau melakukan *upgrade*. Atribut keamanan yaitu bagaimana keamanan diterapkan pada perangkat lunak untuk menghindari gangguan eksternal (Hadiprakoso, 2021).

Menurut (Latelay & Azhari, 2012), kualitas perangkat lunak dari sudut pandang pengguna mencakup kebutuhan, keinginan dan kemauan sedangkan kualitas perangkat lunak dari sudut pandang pengembang yaitu dengan terpenuhinya beberapa karakteristik.



Gambar 7.1. Faktor-faktor kualitas perangkat lunak (Latelay & Azhari, 2012)

Definisi dari faktor-faktor kualitas perangkat lunak yaitu:

1. *Reliability*, yaitu fitur dan fungsi dari perangkat lunak dapat digunakan tanpa adanya kesalahan.

2. *Reusability*, yaitu kemampuan perangkat lunak atau bagian kecil dari perangkat lunak sehingga dapat digunakan kembali di perangkat lunak lain sebagai sebuah komponen yang *reuseable*.
3. *Usability*, yaitu kemudahan pengguna sehingga dapat mengoperasikan dan menggunakan perangkat lunak untuk memenuhi tujuan yang ia miliki.
4. *Maintainability*, yaitu perangkat lunak mengetahui bagian yang terjadi kesalahan dan dapat memperbaikinya. Pada faktor ini dapat dibagi menjadi tiga subfaktor seperti *understandability, modifiability* dan *testability*.
5. *Understandability*, yaitu besar kerumitan pada perangkat lunak secara psikologis.
6. *Modifiability*, yaitu kemudahan sistem atau komponen sehingga dapat dimodifikasi untuk menyesuaikan dengan sebuah aplikasi atau lingkungan tertentu.
7. *Testability*, yaitu kemudahan pengujian ketika melakukan proses pengujian kualitas perangkat lunak sesuai persyaratan yang ada.
8. *Availability*, yaitu kemudahan sistem perangkat lunak atau komponennya untuk dioperasikan atau diakses ketika ingin digunakan.
9. *Complexity*, faktor ini tidak disebutkan dalam hierarki namun masih termasuk ke dalam salah satu faktor. Kompleksitas dapat dibagi menjadi dua jenis yaitu *complexity of problem* dan *complexity of solution*. C*omplexity of problem* merupakan total sumber daya yang akan dipergunakan sebagai solusi dari permasalahan. C*omplexity of solution* dapat dibagi lagi menjadi *time complexity* dan *space complexity*. Sesuai namanya, *time complexity* yaitu sumber daya waktu yang dibutuhkan sistem sedangkan *space complexity* adalah sumber daya memori yang ada pada sistem.

7.7 Implementasi Perangkat Lunak

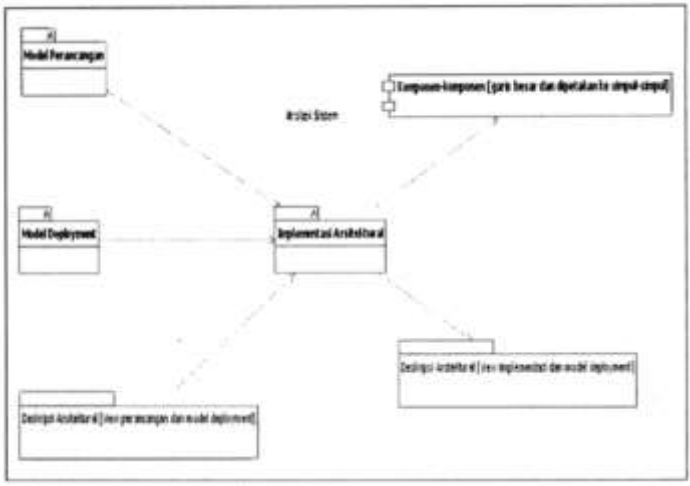
Implementasi adalah melakukan sebuah rencana secara matang dan detail untuk melakukan sesuatu hal. Implementasi akan dieksekusi ketika perencanaan sudah disetujui atau merupakan keputusan akhir (Rusida & Noer, 2018). Implementasi pada perangkat lunak dapat dilakukan dengan acuan perencanaan atau desain yang telah dibuat sebelumnya sehingga implementasi perangkat lunak akan mencakup kebutuhan dan perencanaan yang telah dipertimbangkan dan dibuat dari awal. Menurut (Simarmata, 2010), implementasi dapat dilakukan dalam delapan langkah utama yaitu analisis domain, definisi kebutuhan, arsitektur perangkat lunak, penilaian risiko, prototipe, pengembangan lingkungan dan rangkaian pengujian, integrasi dengan iterasi yang sebelumnya dan pelepasan iterasi.

Implementasi yaitu pelaksanaan atau penerapan maupun pengembangan dari sebuah kerja sistem dari desain yang sudah dibuat. Dalam perencanaan perangkat lunak, implementasi perangkat lunak menjadi salah satu tahapan mengenai perangkat lunak yang akan dibuat. Tujuannya agar dapat diketahui secara jelas proses pembuatan dari setiap komponen yang ada pada perangkat lunak. Menurut (Romindo et al., 2023), terdapat beberapa tujuan dari implementasi yaitu:

1. Membuat perancangan yang baik dengan masukan individu maupun keputusan akhir bersama tim.
2. Mendokumentasikan dan menyimpan prosedur ketika melaksanakan rancangan atau kebijakan.
3. Memenuhi tujuan akhir yang telah ditetapkan sejak awal pembuatan perancangan.
4. Mengetahui banyak masyarakat atau orang yang akan melakukan kebijakan atau perencanaan yang telah dibuat.
5. Menentukan kesuksesan dari rencana yang telah dibuat sehingga dapat ditingkatkan jika masih dirasa kurang.

7.8 Terjemahan Desain ke Implementasi Perangkat Lunak

Hasil desain yang ada pada dokumen persyaratan pengguna dapat menjadi pola yang berguna untuk membantu *programmer* untuk melakukan pengkodean maupun implementasi perangkat lunak. Model desain yang telah ada akan diubah menjadi model desain yang lebih detail dengan mendeskripsikan secara tepat bentuk atau struktur dari perangkat lunak. Struktur yang diperlukan untuk membantu proses implementasi yaitu seperti arsitektur sistem, antarmuka dan komponen yang akan digunakan dalam membangun sistem. Untuk memenuhi persyaratan pengguna, SKPL (Spesifikasi Kebutuhan Perangkat Lunak) dibuat. Selain itu, dilakukan juga pengkodean dan implementasi uraian yang secara spesifik diajukan dalam bentuk terminologi. Hasil akhir yang didapatkan akan di implementasikan menggunakan bahasa pemrograman yang akan digunakan (Rianto, 2019).



Gambar 7.2. Asupan dan Hasil Implementasi Arsitektural (Nugroho, 2010)

Implementasi arsitektural dari perangkat lunak dapat membantu dalam merepresentasikan model implementasi dan arsitektur dari sistem. Setiap bagian komponen yang ada pada sistem dapat dipetakan dalam satu bagian dan akan saling berhubungan antar satu sama lain. Ketika proses implementasi, pengembang dapat memanfaatkan implementasi subsistem-subsistem implementasi dengan cara melacak perancangan dasar dan menyediakan antarmuka-antarmuka yang sama. Selama proses implementasi, pengembang bertanggung jawab memelihara, menghaluskan dan memodifikasi arsitektur model perangkat lunak dan *deployment model* yang telah dibuat (Nugroho, 2010).

7.9 Hal-hal penting dalam Implementasi Perangkat Lunak

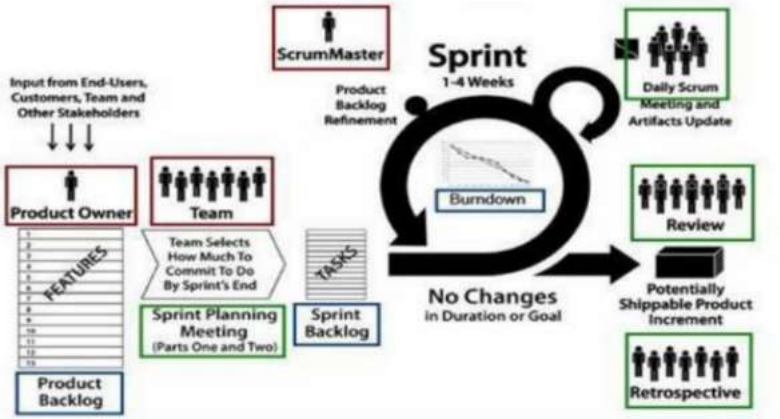
Menurut (Zein *et al.,* 2023), tahapan implementasi terkait erat dengan tahapan pengujian. Pada implementasi perangkat lunak, pengembang mengambil desain sistem yang telah dibuat sebelumnya diimplementasikan menjadi sebuah sistem perangkat lunak. Implementasi perangkat lunak dilakukan dengan menulis kode, menghubungkan setiap komponen dalam sistem dan menguji perangkat lunak yang telah diimplementasikan. Selanjutnya, di tahap pengujian, tim akan menguji kerja perangkat lunak untuk memastikan ia dapat bekerja dengan baik dan telah memenuhi kebutuhan pengguna. Terdapat beberapa jenis pengujian yang berbeda yang dapat dilakukan yaitu pengujian fungsional, pengujian integrasi, pengujian sistem, dan pengujian penerimaan pengguna.

Beberapa hal penting yang harus dilakukan ketika melakukan implementasi sistem yaitu konversi, pelatihan dan *testing* penerimaan. Konversi dilakukan dengan mengambil sistem lama ke sistem baru apabila di sistem lama telah

menggunakan aplikasi terkomputerisasi juga. Pelatihan berfungsi agar pihak yang akan menggunakannya mendapatkan informasi atau pengetahuan tentang sistem. Sosialisasi dapat dilakukan terhadap pihak yang terkait walaupun ia tidak secara langsung akan menggunakan sistem. Untuk periode waktu tertentu, sistem membutuhkan *testing* penerimaan agar sistem dapat melakukan proses belajar dan pengujian sehingga nantinya akan lebih siap untuk digunakan oleh pengguna (Muslihudin & Oktafianto, 2016).

7.10 Teknik Implementasi Perangkat Lunak

Scrum tidak termasuk sebuah teknik dalam membuat sebuah perangkat lunak namun ia bertindak sebagai kerangka kerja dimana di dalam perangkat kerja ini tim dapat melakukan berbagai proses dan teknik untuk mengimplementasikan desain pada perangkat lunak yang sedang dikembangkan. Di dalam *scrum* terdapat beberapa peranan yang akan diperlukan dalam tim yang terdiri dari *product owner, scrum master* dan tim. *Product owner* merupakan orang yang akan bertanggung jawab kepada nilai bisnis yang dimiliki oleh perangkat lunak sedangkan *scrum master* yaitu orang yang harus mengatur dan memantau tim sehingga tim dapat melakukan pekerjaannya dengan lebih efektif, efisien dan produktif. Terakhir, tim yaitu sekumpulan orang yang melakukan pekerjaan yang telah diberikan atau dpilih oleh dirinya sampai selesai (Azdy & Azhari, 2012).



Gambar 7.3. Proses *Scrum* (Ahmad et al., 2022)

Menurut (Ahmad *et al.,* 2022), untuk melakukan implementasi dalam metode *scrum* terdapat empat langkah utama atau artifak yaitu sebagai berikut:

1. Implementasi *Product Backlog* yang memiliki beberapa daftar tugas yang harus dilakukan dalam proyek perangkat lunak dengan estimasi waktu tertentu dan jumlah tim yang akan mengerjakannya.
2. Implementasi perencanaan sesuai perhitungan yang telah dilakukan dengan membuat *Sprint* Desain.
3. Melihat hasil perhitungan yang telah dilakukan pada Desain *Sprint*.
4. Melakukan analisis pada sebuah *Burndown Chart*.

Proses implementasi membutuhkan sebuah metode atau kerangka kerja sebagai wadah melakukan teknik implementasi sehingga proses implementasi dapat dilakukan dengan konsisten dan baik. Menurut (Mutawali et al., 2020), *Scrum* yaitu kerangka kerja yang dapat membantu dalam permasalahan pengembangan perangkat lunak yang kompleks.

Salah satu cara untuk mengimplementasikan desain menjadi sebuah perangkat lunak yaitu dengan melalui beberapa aktivitas seperti *sprint planning*. *Sprint planning* merupakan sebuah cara agar tim pengembang dapat menyepakati rencana yang dibuat bersama sedangkan *Daily Scrum* dilakukan dengan menyusun rencana kerja dalam 24 jam ke selanjutnya sehingga kerjasama antar tim pengembang dapat dilakukan dengan lebih efektif dan efisien. Setelah perencanaan-perencanaan tersebut sudah jelas, maka tim dapat mengeksekusi dan melakukan implementasi sesuai yang telah dibagikan. Kemudian akan dilakukan *Sprint Review* dan *Sprint Restrospective*. *Sprint Review* dilakukan agar setiap anggota dapat mengetahui apa saja yang telah dilakukan atau dikerjakan oleh teman tim yang lain atau dalam kata lain meninjau pekerjaan yang telah dilakukan, yang akan dilakukan dan kendala yang terjadi selama pengerjaan bagian pekerjaan sebelumnya. Lalu pada *Sprint Restrospective*, tim akan menyusun perencanaan yang akan dilakukan pada *sprint* berikutnya sehingga kendala yang sebelumnya terjadi dapat ditangani dan tim dapat melakukan *sprint* berikutnya lebih baik dari *sprint* sebelumnya.

7.11 Kualitas Implementasi Perangkat Lunak

Menurut (Darni *et al.,* 2023), dalam melakukan proyek perangkat lunak terdapat empat hal yang akan dipertimbangkan dan saling berhubungan yang harus didiskusikan. Empat hal tersebut yaitu waktu, biaya, ruang lingkup dan kualitas dari perangkat lunak. Jika ukuran produk perangkat lunak besar, maka akan semakin besar juga ukuran proyek yang akan dikerjakan oleh tim. Untuk menetapkan hal-hal tersebut, para pemangku kepentingan dalam proyek perangkat lunak perlu mempertimbangkan batas waktu yang diberikan kepada tim dalam mengerjakan proyek dan batas

biaya proyek yang diberikan karena hal ini dapat berpengaruh pada kualitas produk akhir dari proyek perangkat lunak.

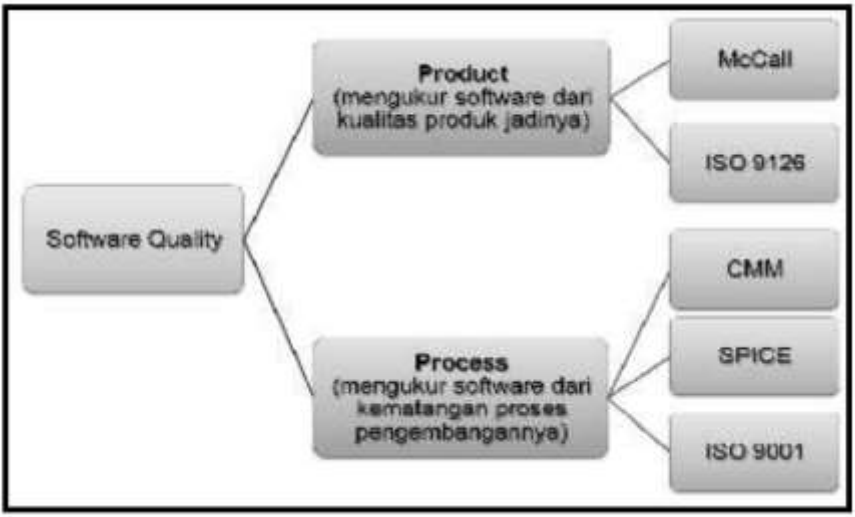
ISO 9126 adalah standar kualitas internasional yang telah diuji dari segi validitas dan reliabilitasnya yang telah teridentifikasi pada WebQEM (Sari, 2016). *International Organization for Standarization* (ISO) pada ISO *Standard* 9126 mengusulkan enam karakteristik utama untuk menguji kualitas yang dimiliki oleh sebuah perangkat lunak sebagai berikut:

1. *Functionality*
2. *Reliability*
3. *Usability*
4. *Efficiency*
5. *Maintability*
6. *Portability*

Karakteristik	Sub-Karakteristik
Functionality	*suitability, accuracy, interoperability, security*
Reliability	*maturity, fault tolerance, recoverability*
Usability	*understandability, learnability, operability, attractiveness*
Efficiency	*time behavior, resource utilization*
Maintainability	*analyzability, changeability, stability*
Portability	*daptability, installability, co-existence, replacability*

Gambar 7.4. Karakteristik dan Sub-karakteristik ISO 9126 (Sari, 2016)

Kualitas pada perangkat lunak dapat diukur melalui beberapa kategori dan aspek tertentu. Salah satunya yaitu dilihat dari hasil produk akhir dan proses pembuatan perangkat lunak tersebut. Kualitas perangkat lunak dari produknya dapat dilihat pada produk jadi di akhir sedangkan pengukuran yang dilakukan dalam prosesnya dilihat dari tingkat kematangan proses pengembangan yang diterapkan (Febria, 2015).



Gambar 7.5. Aspek dan Model Pengukuran Kualitas Perangkat Lunak (Febria, 2015)

Dari penelitian yang dilakukan pada tesis Gregor Panovski yang berjudul "Product Software Quality", perangkat lunak dapat dibagi menjadi tiga kategori yaitu perangkat lunak infrastruktur, alat-alat pengembangan perangkat lunak, dan perangkat lunak aplikasi. Diterapkan metodologi yaitu dengan melakukan dekomposisi untuk mendapatkan sub karakteristik ISO/IEC 9126 yang disebut atribut, mengusulkan metrik eksternal dari atribut dan mendefinisikan metrik yang berhubungan dengan aplikasi domain perangkat lunak. Hasil akhir yang didapatkan dari analisis yang dilakukan pada tujuh perangkat lunak dari ketiga kategori sebelumnya. Pada tiga karakteristik yang ada pada IOS/IEC 9126 yaitu seperti kegunaan, kehandalan, dan efisiensi dengan adanya perbedaan yang terlihat pada ketiga kategori produk yang berbeda.

Untuk kategori kegunaan (*usability*) memiliki peranan penting pada produk perangkat lunak aplikasi namun kurang

penting pada alat pengembangan perangkat lunak dan perangkat lunak infrastruktur. Kesimpulan akhir yang didapatkan yaitu kualitas luar dari produk perangkat lunak bergantung pada domain atau kategorinya.

7.12 Tren Terkini dalam Desain dan Implementasi Perangkat Lunak

Pada saat ini, perangkat lunak berbasis *mobile* lebih banyak ditemukan di setiap lapisan masyarakat mulai dari anak muda sampai orang dewasa. Hal ini karena perangkat *mobile* dapat beradaptasi dengan setiap jenis gadget yang digunakan oleh pengguna seperti telepon genggam, laptop, *tablet* dan lainnya. Hal ini dapat dilihat pada peningkatan akses internet pada telepon genggam yang digunakan oleh pengguna setiap tahunnya.

Dari survey diketahui bahwa terdapat 63,2 juta pengguna *internet* yang menggunakan perangkat *mobile* pada tahun 2017, dengan 21,2 akses *internet* melalui laptop dan lewat perangkat *mobile* sebanyak 51,7. Beberapa aplikasi *mobile* yang banyak digunakan dan diakses oleh pengguna yaitu sistem operasi android Google Play, WhatsApp, dan Youtube. Rata-rata aplikasi tersebut memiliki fungsi utama yaitu untuk berkomunikasi, berbisnis dan mendapatkan informasi terkini (Ramadhan *et al.,* 2020).

Menurut (Gunawan *et al.,* 2020), *Internet of Things (IoT)* yaitu teknologi yang dapat menghubungkan antara satu objek teknologi dengan objek lainnya sehingga dapat berkomunikasi dengan orang-orang yang terhubung dengannya. *IoT* dapat dikatakan sebagai sistem jarak jauh yang ada antar proyek. Jumlah perusahaan teknologi informasi di dunia yang menggunakan *IoT* semakin bertambah. Perusahaan-perusahaan ini kebanyakan merilis produk yang berbentuk layanan sehingga penjelasan detail desain dan implementasi

tidak diungkapkan kepada publik. *IoT* dapat berguna dalam bentuk sensor, arsitektur komputer, perangkat lunak, keamanan, pengemasan, pemilihan teknologi berdasarkan jumlah data, sejauh data yang dibutuhkan dan apapun daya yang dimiliki.

Pada industri 4.0 ini, banyak perusahaan digital yang memanfaatkan *Artificial Intelligence* dalam pembuatan produknya dan bisnis perusahaan. Langkah penerapannya dilakukan mulai dari pengambilan data, desain struktur sistem sampai pencarian algoritma atau metode yang dapat membantu dalam penyelesaian masalah yang kompleks mulai dari awal produksi hingga produk dihasilkan. Mekanisme ini tidak hanya membutuhkan algoritma pada bagian teknik saja tapi juga akan membutuhkan aspek sains sehingga dibutuhkan lebih dari hanya membuat sebuah *web*, sistem informasi atau aplikasi *mobile* saja. Adanya pemanfaatan *AI* atau *machine learning* dapat membantu permasalahan diluar teknis sekalipun. Salah satu contohnya yaitu seperti *Predictive modelling* yang telah banyak diimplementasikan di perusahan penjualan *online* besar di dunia seperti *amazon, alibaba* dan lainnya(Lubis,2021).

DAFTAR PUSTAKA

Ahmad, N., Krisnanik, E., Rupilele, F. G. J., Muliawati, A., Syamsiyah, N., Kraugusteeliana, Cahyono, B. D., Sriyeni, Y., Kristanto, T., Irwanto dan Guntoro. 2022. *Analisa dan Perancangan Sistem Informasi Berorientasi Objek*. Penerbit Widina Media Utama: Bandung.

Azdy, R. A. dan Azhari, S. N. 2012. Implementasi Scrum pada Pengembangan Software Terdistribusi. *Prosiding Seminar Informatika 2012*. 32-37.

Budiman, K. 2022. *Rekayasa Perangkat Lunak*. Penerbit Pustaka Rumah Cinta: Mungkid.

Darni, R., Farell, G., Thamrin dan Agustiarmi, W. 2023. *Pengembangan Perangkat Lunak dan Sistem Informasi*. Penerbit PT Mafy Media Literasi Indonesia: Solok.

Darni, R., Faiza, D., Thamrin, Farell, G. dan Agustiarmi, W. 2023. *Pengujian dan Implementasi Perangkat Lunak*. Penerbit Mafy Media Literasi Indonesia: Solok.

Dwanoko, Y. S. 2016. Implementasi Software Development Life Cycle (SDLC) dalam Penerapan Pembangunan Aplikasi Perangkat Lunak. *Jurnal Teknologi Informasi*. 7(2). 83-94.

Febria, S. H. 2015. Perancangan Alat Ukur Kualitas Perangkat Lunak menggunakan Komponen ISO/IEC 9126. *Jurnal Sistem Informasi dan Teknologi Informasi*. 4(2). 103-115.

Gunawan, T. S., Hayadi, B. H., Paramitha, C. dan Sadikin, M. 2020. Tren Saat ini dan Kemajuan Terbaru Framework IoT untuk Manajemen Perusahaan pada PT.TNC. *Jurnal Inovasi Pengabdian kepada Masyarakat*. 1(2). 164-174.

Hadiprakoso, R. D. 2021. *Rekayasa Perangkat Lunak*. Penerbit Rumah Buku Hikmah: Jakarta.

Jauhari, A., Anamisa, D. R. dan Mufarroha, F. A. 2022. *Analisis dan Desain Sistem Informasi: Pendekatan Pengembangan System*. Penerbit Media Nusa Creative: Malang.

Letelay, K. dan Azhari SN. 2012. Evaluasi Kualitas Perangkat Lunak dengan Metrics Berorientasi Objek. *Prosiding Seminar Nasional Informatika 2022*. 139-145.

Lubis, M. S. Y. 2021. Implementasi Artificial Intelligence pada System Manufaktur Terpadu. *Prosiding Seminar Nasional Teknik UISU 2021*. 1-7.

Muslihudin, M. dan Oktafianto. 2016. *Analisis dan Perancangan Sistem Informasi menggunakan Model Terstruktur dan UML*. Penerbit Andi: Yogyakarta.

Mutawali, L., Fathoni, B. K. dan Asyari, H. 2020. Implementasi Scrum dalam Pengembangan Sistem Informasi Jasa Desain Grafis. *Jurnal Manajemen Informatika dan Sistem Informasi*. 3(2). 116-122.

Nugroho, A. 2010. *Rekayasa Perangkat Lunak Berorientasi Objek dengan Metode USDP (Unified Software Development Process)*. Penerbit Andi: Yogyakarta.

Rachmad, Y. E., Priska, L. D., Tampubolon, Siska, Purbaratri, W., Sudipa, I. G. I., Desyanti, Sulistianto, Migunani, Ariana, A. A. G. B., Faried, I., Atmojo, D. dan Kurniawan, H. 2023. *Rekayasa Perangkat Lunak*. Penerbit PT. Sonpedia Publishing Indonesia: Jambi.

Raharjana, I. K. dan Justitia, A. 2015. Pembuatan Model Sequence Diagram dengan Reverse Engineering Aplikasi Basis Data pada Smartphone untuk Menjaga Konsistensi Desain Perangkat Lunak. *Jurnal Ilmiah Teknologi Informasi*. 13(2). 133-142.

Ramadhan, R., Wardani, S. D. K. dan Amrozi, Y. 2020. Quo Vadis Pengembangan Rekayasa Perangkat Lunak. *Jurnal Teknologi Terapan*. 3(2). 237-244.

Rianto, I. 2021. *Rekayasa Perangkat Lunak*. Penerbit Lakeisha: Klaten.

Romindo, Mayefis, R., Yusnanto, T., Heryana, N., Jamaludin, Aulia, A. P., Permana, A. A., Aisa, S., Pasaribu, J. S., Wahyuddin, S. dan Sihombing, F. A. H. 2023. *Rekayasa Perangkat Lunak*. Penerbit PT Global Eksekutif Teknologi: Padang.

Rusida, N. D. dan Noer, Z. M. 2018. Perancangan Perangkat Lunak Bantu Sistem Penjualan Berbasis Aplikasi Desktop pada Cafe Instamie Pangandaran. *Jurnal Manajemen dan Teknik Informatika*. 1(1). 341-350.

Sari, T. K. 2016. Analisis Kualitas dan Pengembangan Sistem Informasi Akademik Berbasis Web menggunakan Standard ISO 9126. *Jurnal Informatika dan Komputer*. 1(1). 1-7.

Simarmata, J. 2010. *Rekayasa Perangkat Lunak*. Penerbit Andi: Yogyakarta.

Utami, F. H. dan Asnawati. 2015. *Rekayasa Perangkat Lunak*. Penerbit Deepublish: Yogyakarta.

Zein, A., Susilo, D., Mustakim, Effendi, R., Purbaratri, W., Ridwan, A., Nooriansyah, S., Nadziroh, F., Anyan dan Ibrahim, A. 2023. *Konsep Dasar Rekayasa Perangkat Lunak*. Penerbit Yayasan Cendikia Mulia Mandiri: Batam.

BAB 8
PENGUJIAN PERANGKAT LUNAK

Oleh Angga Aditya Permana

8.1 Pendahuluan

Gambar 8.1. Icon Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses penting dalam siklus pengembangan perangkat lunak yang bertujuan untuk menilai kualitas, fungsionalitas, dan kinerja perangkat lunak sebelum dirilis kepada pengguna akhir. Proses ini melibatkan merancang, mengembangkan, dan menjalankan serangkaian kasus uji untuk mengidentifikasi cacat atau kesalahan dalam perangkat lunak. Hasil pengujian dievaluasi dan dianalisis untuk mengidentifikasi masalah yang perlu diperbaiki oleh tim pengembangan sebelum perangkat lunak dapat diluncurkan secara resmi. Pengujian perangkat lunak mencakup berbagai jenis pengujian, termasuk pengujian fungsional, pengujian non-fungsional, pengujian integrasi, pengujian keamanan, dan banyak lagi. Tujuannya adalah untuk memastikan bahwa

perangkat lunak beroperasi dengan andal, memenuhi harapan pengguna, dan meminimalkan risiko terkait masalah atau kerusakan setelah perilisan. (Myers, Badgett and Sandler, 2011)

Pengujian perangkat lunak juga berfungsi sebagai sarana untuk mengukur sejauh mana perangkat lunak memenuhi persyaratan bisnis dan teknis yang telah ditetapkan. Selain itu, pengujian membantu mengidentifikasi area-area yang memerlukan perbaikan atau penyesuaian guna meningkatkan kualitas dan kinerja perangkat lunak. Dalam upaya untuk mencapai hasil pengujian yang komprehensif, seringkali digunakan berbagai metode dan pendekatan, seperti pengujian otomatis untuk menghemat waktu dan sumber daya, serta pengujian manual untuk menggali lebih dalam pada pengalaman pengguna.

Penting untuk memahami bahwa pengujian perangkat lunak bukan hanya tahap akhir sebelum perilisan, tetapi merupakan elemen integral dari keseluruhan proses pengembangan perangkat lunak. Pengujian yang baik memerlukan perencanaan yang cermat, pengembangan kasus uji yang representatif, pelaksanaan pengujian yang teliti, dan analisis mendalam terhadap hasil pengujian. Dengan melibatkan pengguna akhir atau pihak berkepentingan dalam pengujian, perangkat lunak dapat diperbaiki lebih lanjut sesuai dengan umpan balik langsung, sehingga menghasilkan produk yang lebih baik dan lebih sesuai dengan harapan.

Berikut adalah beberapa jenis pengujian perangkat lunak yang umum dilakukan: (Myers, Badgett and Sandler, 2011)

1. **Pengujian Fungsional**: Menguji apakah perangkat lunak berperilaku sesuai dengan spesifikasi fungsionalnya. Ini mencakup pengujian input-output, pengujian aliran kerja, dan pengujian skenario pengguna.

2. **Pengujian Non-Fungsional**: Meliputi pengujian aspek non-fungsional seperti kinerja, skalabilitas, keamanan, keandalan, dan ketersediaan perangkat lunak.
3. **Pengujian Integrasi**: Menguji interaksi antara komponen atau modul perangkat lunak untuk memastikan bahwa mereka berfungsi dengan baik dalam lingkungan yang terintegrasi.
4. **Pengujian Unit**: Menguji unit-unit terkecil perangkat lunak secara terisolasi untuk memastikan bahwa setiap unit berfungsi dengan benar.
5. **Pengujian Regresi**: Menguji perangkat lunak setelah ada perubahan atau penambahan fitur untuk memastikan bahwa perubahan tersebut tidak memengaruhi fungsi yang sudah ada.
6. **Pengujian Uji Beban**: Mengukur kinerja perangkat lunak di bawah beban kerja yang tinggi untuk mengidentifikasi masalah kinerja atau skalabilitas.
7. **Pengujian Keamanan**: Mengidentifikasi dan mengatasi potensi kerentanan keamanan dalam perangkat lunak.
8. **Pengujian Penerimaan Pengguna**: Dilakukan oleh pengguna akhir atau pihak yang berkepentingan untuk memastikan bahwa perangkat lunak memenuhi harapan mereka.
9. **Pengujian Pengguna Akhir**: Melibatkan pengguna akhir dalam proses pengujian untuk mendapatkan umpan balik langsung tentang pengalaman pengguna.
10. **Pengujian Kompatibilitas**: Memastikan bahwa perangkat lunak berfungsi dengan baik di berbagai lingkungan perangkat keras, sistem operasi, peramban, atau platform lainnya.
11. **Pengujian Penyusupan**: Menguji respons perangkat lunak terhadap upaya penetrasi atau serangan.

12. **Pengujian Otomatis**: Menggunakan alat dan skrip otomatis untuk melakukan pengujian perangkat lunak secara efisien.

Pengujian perangkat lunak dapat dilakukan dalam berbagai tahap pengembangan, seperti pengujian unit, pengujian integrasi, dan pengujian sistem. Penting untuk memiliki strategi pengujian yang baik, termasuk perencanaan, desain pengujian, pelaksanaan pengujian, pelaporan hasil, dan tindak lanjut perbaikan.

8.2 Pengujian Fungsional

Gambar 8.2. Icon Pengujian Fungsional

Pengujian fungsional adalah pendekatan kritis dalam pengujian perangkat lunak yang difokuskan pada memverifikasi apakah perangkat lunak berperilaku sesuai dengan spesifikasi fungsional yang telah ditetapkan. Tujuannya adalah untuk memastikan bahwa semua fitur, fungsi, dan interaksi yang diharapkan dari perangkat lunak berjalan dengan benar. Dalam pengujian fungsional, kasus uji dikembangkan untuk menguji berbagai aspek perangkat lunak, seperti input-output, aliran kerja, validasi data, pengaturan parameter, dan respon terhadap situasi-situasi yang mungkin

terjadi. Hasil dari pengujian fungsional memberikan gambaran menyeluruh tentang kualitas dan keandalan perangkat lunak, memungkinkan pengembang untuk mengidentifikasi masalah, cacat, atau penyimpangan dari spesifikasi yang telah ditetapkan. (Myers, Badgett and Sandler, 2011)

Pengujian fungsional sering dilakukan dengan menggunakan skenario pengujian yang direplikasi untuk memastikan bahwa perangkat lunak berperilaku konsisten dan sesuai di berbagai situasi. Selama proses ini, setiap fitur penting dieksplorasi secara menyeluruh untuk memastikan bahwa semua kemungkinan skenario penggunaan telah diuji dan dipertimbangkan. Pengujian fungsional merupakan langkah penting dalam memastikan bahwa perangkat lunak dapat memberikan manfaat yang diharapkan kepada pengguna, serta membantu mengurangi risiko kesalahan atau cacat fungsional yang dapat mempengaruhi pengalaman pengguna dan reputasi perangkat lunak secara keseluruhan.

Pada dasarnya, pengujian fungsional perangkat lunak mencakup serangkaian aktivitas yang menguji fungsionalitas atau fitur-fitur utama dari perangkat lunak. Ini melibatkan pengujian berbagai skenario penggunaan untuk memverifikasi apakah perangkat lunak berperilaku sesuai dengan yang diharapkan. Berikut adalah beberapa contoh aktivitas yang dilakukan dalam pengujian fungsional dan contoh konkretnya:

1. Pengujian Input-Output:
   Contoh: Dalam aplikasi formulir pendaftaran online, pengujian input-output dapat memastikan bahwa data yang dimasukkan oleh pengguna diambil dengan benar dan disimpan secara tepat dalam basis data.
2. Pengujian Aliran Kerja:
   Contoh: Dalam aplikasi e-commerce, pengujian aliran kerja bisa mencakup langkah-langkah seperti memilih produk, menambahkannya ke keranjang belanja, proses checkout, dan konfirmasi pembayaran.

3. Pengujian Validasi Data:
   Contoh: Dalam sistem manajemen inventaris, pengujian validasi data dapat memastikan bahwa entri data seperti jumlah stok barang atau harga berada dalam rentang yang valid.
4. Pengujian Pengaturan Parameter:
   Contoh: Dalam aplikasi pengaturan profil pengguna, pengujian pengaturan parameter dapat memeriksa apakah perubahan yang dibuat pada preferensi pengguna (misalnya, tema, bahasa, atau notifikasi) diterapkan dengan benar.
5. Pengujian Navigasi:
   Contoh: Dalam aplikasi peta, pengujian navigasi dapat menguji kemampuan perangkat lunak untuk menghitung dan menampilkan rute yang optimal antara dua titik.
6. Pengujian Fungsional pada Aplikasi Web:
   Contoh: Memastikan bahwa tombol-tombol, formulir, dan tautan di situs web e-commerce berfungsi dengan benar dan mengarahkan pengguna ke halaman yang tepat.
7. Pengujian Fungsional pada Aplikasi Mobile:
   Contoh: Menguji aplikasi seluler layanan pengiriman makanan untuk memastikan bahwa pesanan dapat ditempatkan dengan benar, alamat pengiriman diambil dengan akurat, dan pembayaran berhasil dilakukan.
8. Pengujian Fungsional dalam Perangkat Lunak Keuangan:
   Contoh: Pengujian perangkat lunak untuk sistem perbankan, menguji transfer dana antar akun, verifikasi tanda tangan digital, atau perhitungan bunga yang akurat.

Penting untuk mengembangkan skenario pengujian yang mendekati situasi penggunaan nyata sehingga perangkat lunak dapat diuji secara komprehensif dan masalah potensial dapat diidentifikasi dengan lebih baik.

8.3 Pengujian Non-Fungsional

Gambar 8.3. Icon Pengujian Non Fungsional

Pengujian non-fungsional adalah aspek kritis dari pengembangan perangkat lunak yang fokus pada evaluasi karakteristik kualitatif yang tidak secara langsung berkaitan dengan fungsionalitas utama perangkat lunak. Tujuannya adalah untuk memastikan bahwa perangkat lunak tidak hanya berfungsi, tetapi juga memenuhi standar kinerja, keamanan, dan pengalaman pengguna yang diharapkan. Jenis pengujian ini melibatkan pengukuran dan pengujian berbagai parameter kinerja, responsivitas, keamanan, serta kemampuan perangkat lunak dalam mengatasi beban kerja yang tinggi atau situasi ekstrem. (Myers, Badgett and Sandler, 2011)

Misalnya, dalam pengujian non-fungsional, kita dapat menguji kinerja perangkat lunak dengan memuatkannya dengan jumlah pengguna yang besar secara bersamaan untuk mengukur seberapa baik perangkat lunak menangani beban kerja yang tinggi. Pengujian juga dapat difokuskan pada responsivitas, di mana waktu respons perangkat lunak terhadap tindakan pengguna diukur dan dievaluasi. Selain itu, pengujian non-fungsional juga mencakup pengujian keamanan,

seperti mengidentifikasi potensi kerentanan atau risiko keamanan dalam perangkat lunak. Dalam rangka menciptakan pengalaman pengguna yang optimal, pengujian non-fungsional juga melibatkan pengujian antarmuka pengguna dan aksesibilitas perangkat lunak untuk memastikan bahwa pengguna dapat berinteraksi dengan mudah dan efektif dengan perangkat lunak tersebut. Dengan memperhatikan aspek-aspek ini, pengujian non-fungsional berperan penting dalam memastikan bahwa perangkat lunak tidak hanya berfungsi, tetapi juga memberikan kinerja yang handal, aman, dan sesuai dengan harapan pengguna.

Pengujian non-fungsional melibatkan serangkaian aktivitas untuk mengukur dan mengevaluasi aspek-aspek yang tidak terkait langsung dengan fitur atau fungsionalitas utama perangkat lunak. Berikut adalah beberapa hal yang dilakukan dalam pengujian non-fungsional beserta contohnya:

1. Pengujian Kinerja: Menguji bagaimana perangkat lunak berperilaku dalam hal kinerja, yaitu seberapa cepat atau lambat perangkat lunak merespons permintaan pengguna atau tindakan tertentu.
   Contoh: Mengukur waktu respons ketika pengguna melakukan pencarian atau mengakses halaman pada situs web e-commerce.
2. Pengujian Skalabilitas: Menguji kemampuan perangkat lunak untuk mengatasi peningkatan beban kerja atau jumlah pengguna tanpa mengalami penurunan signifikan dalam kinerja.
   Contoh: Memuatkan situs web dengan jumlah pengguna yang berkembang pesat untuk melihat apakah kinerja situs tetap memadai.
3. Pengujian Ketersediaan (*Availability Testing*): Menguji seberapa sering perangkat lunak tersedia dan berfungsi dalam periode waktu tertentu.

Contoh: Menguji berapa lama perangkat lunak dapat berjalan tanpa gangguan sebelum perlu dilakukan pemeliharaan.

4. Pengujian Keamanan: Menguji kerentanan keamanan dan melihat apakah perangkat lunak rentan terhadap serangan atau pelanggaran.
   Contoh: Melakukan serangkaian tes penetrasi untuk mengidentifikasi potensi titik masuk bagi peretas.
5. Pengujian Aksesibilitas: Menguji apakah perangkat lunak dapat diakses dan digunakan oleh pengguna dengan berbagai tantangan fisik atau kognitif.
   Contoh: Menguji bagaimana perangkat lunak dapat diakses oleh pengguna dengan disabilitas visual menggunakan teknologi pembaca layar.
6. Pengujian Responsivitas: Menguji bagaimana perangkat lunak berperilaku pada berbagai perangkat atau resolusi layar.
   Contoh: Mengujikan aplikasi seluler pada berbagai perangkat dan mengukur sejauh mana tampilan dan fungsionalitas tetap optimal.
7. Pengujian Kapasitas: Menguji kapasitas maksimum perangkat lunak, seperti berapa banyak pengguna yang dapat diakomodasi sebelum kinerja menurun.
   Contoh: Menguji berapa banyak pengguna yang dapat melakukan streaming video secara bersamaan tanpa pengurangan kualitas.
8. Pengujian Stabilitas: Menguji stabilitas perangkat lunak dengan memastikan bahwa tidak ada kegagalan sistem yang signifikan dalam jangka waktu tertentu.
   Contoh: Menjalankan aplikasi selama beberapa jam untuk melihat apakah ada penurunan kinerja atau crash yang tidak diinginkan.

9. Pengujian Beban (*Load Testing*): Menguji kinerja perangkat lunak di bawah beban kerja yang tinggi untuk melihat bagaimana perangkat lunak mengatasi situasi beban tinggi. Contoh: Memuatkan situs web dengan pengguna yang banyak secara bersamaan untuk mengamati apakah kinerjanya tetap stabil.

Melalui pengujian non-fungsional, tim pengembangan dapat memastikan bahwa perangkat lunak tidak hanya berfungsi dengan benar, tetapi juga memenuhi standar kinerja, keamanan, dan pengalaman pengguna yang diharapkan.

8.4 Pengujian Integrasi

Gambar 8.4. Icon Pengujian Integrasi

Pengujian integrasi merupakan langkah kritis dalam pengembangan perangkat lunak yang bertujuan untuk memastikan bahwa berbagai komponen atau modul perangkat lunak dapat berinteraksi dan bekerja bersama secara harmonis.

Proses ini melibatkan pengujian antarmuka antara komponen-komponen yang berbeda, serta validasi interaksi data dan aliran informasi di antara mereka. Pengujian integrasi memastikan bahwa integrasi teknis perangkat lunak berjalan lancar dan meminimalkan risiko masalah kompatibilitas atau inkonsistensi yang dapat muncul saat komponen-komponen dihubungkan. Dalam pengujian integrasi, skenario pengujian dirancang untuk menguji berbagai situasi dan jalur aliran data yang mungkin terjadi, sehingga memastikan bahwa integrasi antara komponen-komponen tersebut stabil, aman, dan sesuai dengan spesifikasi yang telah ditetapkan. (Myers, Badgett and Sandler, 2011)

Sebagai contoh, bayangkan pengembangan sebuah sistem e-commerce. Pengujian integrasi akan menguji bagaimana modul pembayaran berinteraksi dengan modul inventaris dan bagaimana data produk diteruskan dari modul katalog ke modul keranjang belanja. Selain itu, pengujian integrasi juga akan memeriksa apakah sistem pihak ketiga yang digunakan, seperti sistem pembayaran online, dapat terintegrasi dengan lancar dan berkomunikasi dengan sistem internal. Dalam kasus ini, pengujian integrasi akan mengidentifikasi masalah seperti kegagalan transfer data antara modul, perbedaan format data yang tidak kompatibel, atau keterlambatan respons sistem pihak ketiga. Dengan menjalani pengujian integrasi yang cermat, perangkat lunak dapat memastikan bahwa seluruh komponen bekerja secara sinergis, menciptakan pengalaman pengguna yang konsisten dan tanpa hambatan.

Pengujian integrasi melibatkan serangkaian langkah untuk memastikan bahwa berbagai komponen atau modul perangkat lunak dapat berinteraksi dan bekerja bersama tanpa masalah. Berikut adalah beberapa hal yang dilakukan dalam pengujian integrasi beserta contohnya:

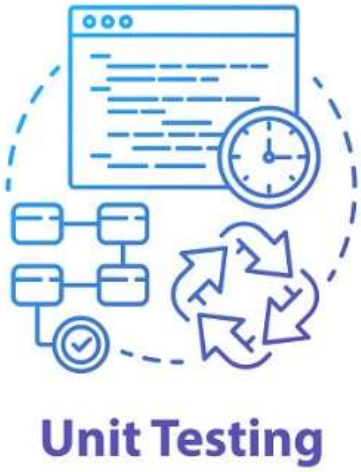
1. Pengujian Antarmuka (Interface Testing): Menguji interaksi antara antarmuka komponen atau modul yang berbeda untuk memastikan bahwa data dapat dikirimkan dan diterima dengan benar.
   Contoh: Memastikan bahwa data pesanan dari modul keranjang belanja diintegrasikan dengan modul pembayaran dan informasi pembayaran diteruskan dengan benar.
2. Pengujian Integrasi Fungsional: Menguji bagaimana berbagai komponen berinteraksi dan saling bergantung satu sama lain dalam menghasilkan fungsi yang lebih kompleks.
   Contoh: Menguji bagaimana komponen otentikasi dan otorisasi bekerja bersama untuk memberikan akses yang sesuai kepada pengguna.
3. Pengujian Integrasi Proses Bisnis: Menguji aliran kerja bisnis dari awal hingga akhir, menguji bagaimana komponen-komponen berinteraksi untuk menghasilkan hasil yang diharapkan.
   Contoh: Menguji proses pemesanan dari pengecekan ketersediaan produk hingga pengiriman dan pelacakan pengiriman.
4. Pengujian Integrasi Teknis: Menguji bagaimana komponen-komponen berinteraksi pada tingkat teknis, seperti integrasi antara sistem perangkat keras dan perangkat lunak.
   Contoh: Memastikan bahwa perangkat lunak komputer dapat berkomunikasi dengan perangkat keras seperti printer atau perangkat penyimpanan eksternal.
5. Pengujian Integrasi Layanan (*Service Integration Testing*): Menguji integrasi antara layanan atau komponen yang berjalan di dalam arsitektur berbasis layanan (SOA) atau sistem terdistribusi.

Contoh: Menguji bagaimana layanan pembayaran terintegrasi dengan layanan pihak ketiga yang menyediakan informasi tarif mata uang asing.

6. Pengujian Integrasi Pihak Ketiga: Menguji integrasi dengan sistem atau layanan pihak ketiga, seperti integrasi dengan API pihak ketiga.
   Contoh: Menguji bagaimana integrasi dengan layanan peta pihak ketiga berfungsi untuk menghitung rute pengiriman yang optimal.
7. Pengujian Integrasi Aplikasi Seluler: Menguji integrasi antara komponen aplikasi seluler, seperti antarmuka pengguna dan fitur-fitur internal.
   Contoh: Menguji bagaimana modul kamera dalam aplikasi seluler berinteraksi dengan modul pengeditan gambar.
8. Pengujian Integrasi Sistem Operasi: Menguji bagaimana perangkat lunak berinteraksi dengan sistem operasi dan memastikan kompatibilitasnya.
   Contoh: Menguji apakah aplikasi desktop berjalan dengan baik di berbagai sistem operasi, seperti Windows, macOS, atau Linux.

Melalui pengujian integrasi yang baik, tim pengembangan dapat mengidentifikasi masalah atau kesalahan dalam interaksi antara komponen-komponen, sehingga meminimalkan risiko terkait integrasi dan memastikan bahwa perangkat lunak berfungsi dengan baik sebagai keseluruhan.

8.5 Pengujian Unit



Gambar 8.5. Icon Pengujian Unit

Pengujian unit adalah tahap awal dan fundamental dalam proses pengujian perangkat lunak yang fokus pada pengujian terhadap unit-unit terkecil dari kode perangkat lunak, seperti fungsi, metode, atau modul. Tujuan utamanya adalah untuk memastikan bahwa setiap unit bekerja dengan benar secara terisolasi sebelum diintegrasikan dengan unit-unit lainnya. Dalam pengujian unit, pengembang membuat skenario pengujian yang mencakup berbagai input dan kondisi untuk menguji berbagai aspek dari unit tersebut. Setiap unit diuji untuk memastikan bahwa ia menghasilkan keluaran yang diharapkan dan berperilaku sesuai dengan spesifikasi fungsional. Pengujian unit membantu mengidentifikasi kesalahan atau cacat pada tahap awal pengembangan, sehingga memungkinkan pengembang untuk memperbaiki masalah sebelum tahap integrasi dan pengujian lebih lanjut.

Sebagai contoh, bayangkan sebuah fungsi matematika sederhana untuk menghitung total harga belanja berdasarkan harga dan jumlah barang yang dipilih. Dalam pengujian unit, pengembang akan membuat skenario pengujian dengan beberapa kasus uji yang berbeda, seperti harga dan jumlah barang yang bervariasi. Fungsi tersebut akan diuji untuk memastikan bahwa ia menghasilkan total harga yang benar sesuai dengan perhitungan matematika yang diharapkan. Jika ada kesalahan dalam kode, pengujian unit akan membantu mengidentifikasi masalah tersebut sehingga pengembang dapat melakukan perbaikan sebelum fungsi tersebut digunakan dalam komponen yang lebih besar. Dengan melakukan pengujian unit yang cermat, perangkat lunak memiliki dasar yang kuat untuk tahap integrasi dan pengujian selanjutnya.

Pengujian unit melibatkan sejumlah langkah untuk menguji unit-unit terkecil dari perangkat lunak secara terisolasi. Berikut adalah beberapa hal yang dilakukan dalam pengujian unit beserta contohnya:

1. Pengembangan Kasus Uji (*Test Case Development*): Pengembang membuat skenario pengujian yang mencakup berbagai kemungkinan input dan kondisi untuk menguji fungsionalitas unit.
   Contoh: Untuk fungsi penambahan, skenario pengujian dapat mencakup berbagai pasangan angka yang akan dijumlahkan.
2. Eksekusi Pengujian (*Test Execution*): Pengembang menjalankan skenario pengujian yang telah dibuat untuk menguji unit dan mendapatkan keluaran yang dihasilkan.
   Contoh: Untuk fungsi pengurangan, skenario pengujian akan menjalankan fungsi dengan input tertentu dan memeriksa apakah hasil pengurangannya benar.
3. Pengujian Kasus Uji Positif dan Negatif: Pengujian mencakup kasus uji yang menghasilkan output yang

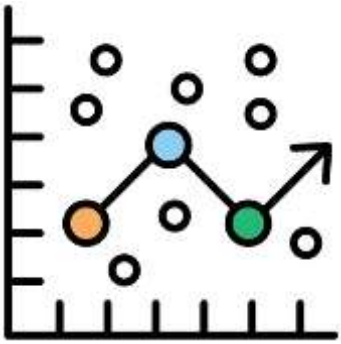
diharapkan (positif) dan kasus uji yang menghasilkan output yang tidak diharapkan (negatif).
Contoh positif: Pengujian unit untuk fungsi validasi email akan memeriksa apakah alamat email valid dikenali sebagai benar.
Contoh negatif: Pengujian unit yang mencoba memasukkan alamat email yang salah dan memastikan bahwa fungsi menolaknya.

4. Pengujian Batas (*Boundary Testing*): Mengujikan unit dengan input pada batas nilai maksimum atau minimum untuk melihat bagaimana unit merespons.
   Contoh: Pengujian unit untuk fungsi perhitungan usia dengan input 0 atau 150 tahun untuk memastikan bahwa hasilnya sesuai dengan harapan.
5. Pengujian *Exception* (*Exception Testing*): Pengujian unit untuk menguji respons unit terhadap kondisi yang dapat menyebabkan pengecualian atau kesalahan.
   Contoh: Pengujian unit untuk memastikan bahwa unit menangani dengan benar saat menerima input yang tidak valid.
6. Pengujian Kode Edge: Menguji unit dengan memeriksa titik-titik potensial di mana perubahan kecil dalam input dapat menghasilkan perubahan dalam output.
   Contoh: Pengujian unit pada fungsi perhitungan diskon yang memiliki berbagai aturan berdasarkan kisaran harga.
7. Pengujian Regresi: Setelah perubahan dilakukan pada kode, pengujian unit dapat dijalankan kembali untuk memastikan bahwa perubahan tersebut tidak memengaruhi fungsionalitas yang sudah ada sebelumnya.

Melalui serangkaian langkah ini, pengujian unit membantu mengidentifikasi masalah atau kesalahan pada tahap awal pengembangan, sehingga memungkinkan

pengembang untuk melakukan perbaikan sebelum tahap integrasi dan pengujian lebih lanjut.

8.6 Pengujian Regresi



Gambar 8.6. Icon Pengujian Regresi

Pengujian regresi merupakan pendekatan penting dalam pengujian perangkat lunak yang fokus pada memastikan bahwa perubahan atau penambahan fitur baru tidak mempengaruhi fungsionalitas yang sudah ada sebelumnya. Tujuannya adalah untuk mengidentifikasi dan mencegah timbulnya kesalahan regresi, yaitu kesalahan yang muncul ketika perubahan yang dilakukan mengganggu atau merusak bagian-bagian yang sudah berfungsi dengan baik. Dalam pengujian regresi, skenario pengujian yang telah ada sebelumnya dijalankan ulang untuk memastikan bahwa fungsionalitas inti perangkat lunak tetap utuh setelah perubahan dilakukan. Pengujian ini membantu mengamankan kemampuan perangkat lunak untuk tetap konsisten dan berfungsi dengan benar dalam setiap iterasi atau pengembangan lanjutan.

Sebagai contoh, bayangkan sebuah aplikasi manajemen proyek yang sedang dikembangkan. Setelah fitur baru ditambahkan untuk memungkinkan kolaborasi tim secara

online, pengujian regresi akan dijalankan untuk memeriksa apakah fitur-fitur yang sudah ada sebelumnya, seperti pembuatan tugas atau penjadwalan, masih berfungsi seperti yang seharusnya. Pengujian regresi akan menjalankan skenario pengujian sebelumnya untuk memastikan bahwa perubahan baru tidak merusak atau mempengaruhi fungsionalitas yang sudah ada sebelumnya. Jika ditemukan kesalahan regresi, tim pengembangan dapat segera melakukan perbaikan sebelum merilis versi yang diperbarui. Dengan melakukan pengujian regresi secara teratur, perangkat lunak dapat tetap konsisten, andal, dan sesuai dengan harapan pengguna, bahkan setelah perubahan-perubahan dilakukan.

Pengujian regresi melibatkan sejumlah langkah untuk memastikan bahwa perubahan atau penambahan baru pada perangkat lunak tidak mengakibatkan terjadinya kesalahan atau penurunan kualitas pada fitur-fitur yang sudah ada sebelumnya. Berikut adalah beberapa hal yang dilakukan dalam pengujian regresi beserta contohnya:

1. Pengujian Skenario Pengujian Lama: Menjalankan kembali skenario pengujian yang sudah ada sebelumnya untuk memastikan bahwa fitur-fitur yang sudah ada sebelumnya masih berfungsi dengan benar.

   Contoh: Pada aplikasi e-commerce, skenario pengujian pembuatan akun baru dan proses pemesanan yang telah diuji sebelumnya akan dijalankan ulang setelah pembaruan fitur lainnya.
2. Pengujian Penerimaan Pengguna: Melibatkan pengguna akhir atau pihak berkepentingan untuk menjalankan skenario pengujian yang relevan dan memberikan umpan balik terkait perubahan baru.

   Contoh: Setelah melakukan pembaruan pada aplikasi media sosial, pihak pengguna dapat diundang untuk menguji fungsionalitas baru dan memberikan masukan mengenai pengalaman pengguna.

3. Pengujian Integrasi Ulang: Menguji kembali integrasi antara komponen-komponen perangkat lunak setelah ada perubahan untuk memastikan bahwa interaksi antar komponen tetap berjalan baik.
   Contoh: Setelah mengubah modul pembayaran dalam aplikasi finansial, pengujian integrasi ulang akan memastikan bahwa data pembayaran tetap terkirim dengan benar ke modul keuangan.
4. Pengujian Kinerja Regresi: Menguji kinerja perangkat lunak setelah perubahan untuk memastikan bahwa kinerja umum dan responsivitas tidak terpengaruh.
   Contoh: Setelah pembaruan pada aplikasi streaming video, pengujian kinerja akan dilakukan untuk memastikan bahwa video tetap dimuat dengan cepat dan tanpa gangguan.
5. Pengujian Pengguna Akhir: Melibatkan pengguna akhir untuk menjalankan skenario pengujian yang mencakup aktivitas sehari-hari mereka dan memastikan bahwa perubahan baru tidak memengaruhi pengalaman pengguna.
   Contoh: Setelah melakukan perubahan pada aplikasi mobile bank, pengujian oleh pengguna akhir dapat membantu memverifikasi bahwa pengguna masih dapat dengan mudah melakukan transfer dana atau cek saldo.

Melalui pengujian regresi, tim pengembangan dapat dengan cepat mengidentifikasi dan mengatasi kesalahan yang mungkin muncul setelah perubahan dilakukan, sehingga memastikan bahwa perangkat lunak tetap berfungsi dengan baik dan memenuhi standar kualitas yang diharapkan.

8.7 Pengujian Uji Beban

Gambar 8.7. Icon Pengujian Uji Beban

Pengujian uji beban (*load testing*) merupakan metode penting dalam pengujian perangkat lunak yang bertujuan untuk mengukur kinerja perangkat lunak di bawah tekanan beban kerja yang tinggi. Tujuannya adalah untuk mengidentifikasi batas kinerja sistem, mengevaluasi responsivitas, dan memastikan bahwa perangkat lunak dapat berfungsi dengan baik dalam kondisi beban puncak. Dalam pengujian uji beban, simulasi dilakukan dengan menghasilkan lalu lintas pengguna atau permintaan yang tinggi pada perangkat lunak untuk melihat bagaimana perangkat lunak menangani situasi beban tinggi dan apakah kinerjanya tetap optimal. Pengujian ini membantu mengidentifikasi masalah kinerja, fluktuasi kinerja yang mungkin terjadi, dan batasan-batasan sistem dalam mengatasi permintaan yang besar.

Sebagai contoh, bayangkan sebuah situs web e-commerce yang ingin menjalani pengujian uji beban sebelum musim liburan untuk memastikan kinerja optimalnya. Dalam pengujian ini, akan dibuat skenario simulasi pengguna yang mencakup sejumlah besar pengguna yang melakukan

pencarian, menambahkan produk ke keranjang belanja, dan menyelesaikan pembayaran dalam waktu yang hampir bersamaan. Dengan menguji beban tinggi semacam ini, tim pengembangan dapat melihat apakah situs web masih responsif, berapa lama waktu respons yang diperlukan, apakah ada penurunan kualitas gambar atau tampilan, dan sejauh mana sistem tetap stabil. Hasil pengujian uji beban akan memberikan wawasan berharga tentang batasan sistem dan membantu mengambil langkah-langkah perbaikan atau peningkatan kinerja yang diperlukan sebelum situs web dihadapkan pada beban penggunaan yang sebenarnya saat liburan.

Pengujian uji beban melibatkan langkah-langkah untuk menguji kinerja perangkat lunak di bawah tekanan beban kerja yang tinggi. Berikut adalah beberapa hal yang dilakukan dalam pengujian uji beban beserta contohnya:

1. Perencanaan Skenario Beban: Pengembang merencanakan skenario pengujian yang mencakup beban kerja yang tinggi, seperti jumlah pengguna yang banyak atau permintaan transaksi yang berulang dalam waktu singkat.
   Contoh: Pada aplikasi *e-commerce*, pengujian mungkin mencakup skenario simulasi dengan 1000 pengguna secara bersamaan yang melakukan penelusuran produk dan penyelesaian transaksi.
2. Simulasi Beban Kerja: Menggunakan alat uji beban, pengembang akan menghasilkan lalu lintas yang tinggi pada perangkat lunak untuk menguji responsivitas dan kinerja sistem.
   Contoh: Dengan alat uji beban, pengujian dapat mensimulasikan ratusan pengguna yang mengakses situs web dalam waktu yang bersamaan dan melakukan berbagai tindakan.
3. Pengukuran Kinerja: Selama pengujian, pengukuran dilakukan untuk memantau parameter kinerja seperti

waktu respons, kecepatan transaksi, penggunaan sumber daya, dan lain-lain.
Contoh: Pengukuran dapat melibatkan memonitor berapa lama situs web merespons permintaan pengguna atau berapa banyak permintaan yang dapat diakomodasi dalam satu detik.

4. Analisis Hasil: Setelah pengujian, hasil yang diperoleh dianalisis untuk mengidentifikasi fluktuasi kinerja, titik batas beban, atau kemungkinan titik kegagalan sistem.
   Contoh: Hasil analisis mungkin menunjukkan bahwa waktu respons situs web meningkat secara signifikan saat jumlah pengguna mencapai 1000, mengindikasikan batasan beban.
5. Optimisasi dan Perbaikan: Berdasarkan hasil analisis, tim pengembangan dapat melakukan perbaikan atau optimisasi untuk meningkatkan kinerja perangkat lunak.
   Contoh: Tim pengembangan mungkin mengoptimalkan basis data atau mengimplementasikan caching untuk mengurangi waktu respons dan meningkatkan kapasitas.
6. Pengujian Skala Besar (*Stress Testing*): Pengujian juga dapat mencakup skenario ekstrim dengan beban melebihi kapasitas normal untuk melihat bagaimana perangkat lunak bertahan dalam kondisi puncak.
   Contoh: Pengujian skala besar dapat mencoba memuatkan sistem dengan 10.000 pengguna secara bersamaan untuk mengukur bagaimana sistem merespons beban yang ekstrem.

Melalui pengujian uji beban yang cermat, tim pengembangan dapat mengidentifikasi masalah kinerja, batasan sistem, dan melakukan perbaikan yang diperlukan untuk memastikan bahwa perangkat lunak dapat berjalan dengan baik dalam kondisi beban kerja yang tinggi.

8.8 Pengujian Keamanan

Gambar 8.8. Icon Pengujian Keamanan

Pengujian keamanan (*security testing*) merupakan aspek penting dalam pengembangan perangkat lunak yang bertujuan untuk mengidentifikasi dan mengatasi potensi kerentanan serta risiko keamanan yang dapat dieksploitasi oleh pihak yang tidak berwenang. Tujuan utamanya adalah untuk melindungi data sensitif, mencegah serangan peretas, dan menjaga integritas serta kerahasiaan sistem. Dalam pengujian keamanan, berbagai teknik, alat, dan skenario digunakan untuk mensimulasikan serangan yang mungkin terjadi, termasuk serangan peretas atau pencurian data. Pengujian ini membantu mengidentifikasi potensi masalah keamanan, seperti kerentanan perangkat lunak atau konfigurasi yang buruk, sehingga tim pengembangan dapat mengambil tindakan korektif sebelum perangkat lunak diimplementasikan secara luas.

Sebagai contoh, bayangkan sebuah aplikasi perbankan yang sedang dikembangkan. Pengujian keamanan akan mencakup upaya untuk mengidentifikasi celah keamanan yang dapat dimanfaatkan oleh peretas untuk mencuri informasi keuangan atau merusak transaksi. Dalam pengujian ini, teknik penetrasi dapat digunakan untuk mencoba meretas sistem,

serta simulasi serangan DDoS (*Distributed Denial of Service*) untuk menguji sejauh mana sistem dapat bertahan dari lonjakan lalu lintas yang tinggi. Selain itu, pengujian juga dapat melibatkan pemeriksaan terhadap kode perangkat lunak untuk menemukan potensi kerentanan atau celah keamanan. Dengan mengidentifikasi dan mengatasi masalah keamanan sebelum implementasi, pengujian keamanan membantu memastikan bahwa perangkat lunak beroperasi dengan aman dan data pengguna tetap terlindungi.

Jenis-jenis pengujian keamanan lainnya termasuk pengujian autentikasi (*authentication testing*) untuk memastikan bahwa hanya pengguna yang sah yang dapat mengakses sistem, pengujian otorisasi (authorization testing) untuk memeriksa apakah akses diberikan sesuai dengan hak pengguna, pengujian enkripsi (*encryption testing*) untuk melindungi data yang ditransmisikan, dan pengujian manajemen sesi (*session management testing*) untuk memastikan bahwa sesi pengguna dikelola dengan baik. Melalui pengujian keamanan yang komprehensif, perangkat lunak dapat memenuhi standar keamanan yang tinggi dan mengurangi risiko terkait ancaman keamanan yang mungkin muncul.

Pengujian keamanan melibatkan serangkaian langkah untuk mengidentifikasi dan menguji potensi kerentanan serta risiko keamanan dalam perangkat lunak. Berikut adalah beberapa hal yang dilakukan dalam pengujian keamanan beserta contohnya:

1. Pengujian Pemindaian Keamanan (*Security Scanning*): Pengujian ini melibatkan penggunaan alat otomatis untuk memeriksa perangkat lunak dan infrastruktur untuk mencari celah atau kerentanan yang dikenal.
   Contoh: Pengujian menggunakan alat pemindaian untuk mendeteksi kerentanan yang terkait dengan versi perangkat lunak yang sudah usang.

2. Pengujian Penetrasi (*Penetration Testing*): Tim pengujian keamanan akan mencoba secara aktif memanfaatkan celah keamanan dalam sistem, mencoba meretas dan mendapatkan akses yang tidak sah.
   Contoh: Seorang ahli keamanan mencoba meretas sistem perbankan dengan tujuan mendapatkan akses ke rekening bank tanpa izin.
3. Pengujian Serangan DDoS (*Distributed Denial of Service Testing*): Pengujian ini melibatkan simulasi serangan DDoS untuk melihat bagaimana sistem merespons lonjakan lalu lintas yang tinggi.
   Contoh: Pengujian ini akan mencoba membanjiri situs web dengan permintaan hingga menyebabkan sistem down atau tidak responsif.
4. Pengujian Injeksi SQL (SQL *Injection Testing*): Pengujian ini bertujuan untuk mengidentifikasi apakah perangkat lunak rentan terhadap serangan injeksi SQL, yang dapat memungkinkan peretas mengakses atau merusak basis data.
   Contoh: Pengujian mencoba memasukkan kode berbahaya ke dalam formulir input untuk melihat apakah perangkat lunak dapat melaksanakan perintah SQL yang tidak seharusnya.
5. Pengujian *Cross-Site Scripting* (XSS Testing): Pengujian ini menguji apakah perangkat lunak mampu mencegah serangan *cross-site scripting* yang dapat menyebabkan script jahat dieksekusi pada browser pengguna.
   Contoh: Pengujian mencoba menyisipkan skrip jahat ke dalam bidang input yang kemudian dieksekusi oleh peramban pengguna.
6. Pengujian Manajemen Otentikasi dan Otorisasi: Pengujian ini memastikan bahwa mekanisme autentikasi (pemastian identitas) dan otorisasi (pengendalian hak akses) berfungsi dengan benar.

Contoh: Pengujian dapat mencoba login dengan kombinasi nama pengguna dan kata sandi yang salah untuk melihat apakah sistem menghalangi akses.

7. Pengujian Enkripsi Data: Pengujian ini memastikan bahwa data yang ditransmisikan atau disimpan di dalam sistem dienkripsi dengan benar.
   Contoh: Pengujian akan mencoba mengakses data yang dienkripsi secara ilegal untuk melihat apakah data tetap terlindungi.

Melalui pengujian keamanan yang komprehensif seperti ini, tim pengembangan dapat mengidentifikasi dan mengatasi potensi kerentanan serta risiko keamanan sebelum perangkat lunak diimplementasikan secara luas, sehingga memastikan bahwa sistem beroperasi dengan tingkat keamanan yang tinggi.

8.9 Pengujian Penerimaan Pengguna

Gambar 8.9. Icon Pengujian Penerimaan Pengguna

Pengujian penerimaan pengguna (*user acceptance testing* atau UAT) adalah tahap penting dalam siklus

pengembangan perangkat lunak di mana pengguna akhir secara aktif terlibat untuk menguji perangkat lunak dan memastikan bahwa itu memenuhi kebutuhan dan harapan mereka. Tujuan utama pengujian ini adalah untuk memvalidasi bahwa perangkat lunak telah memenuhi persyaratan fungsional dan bisnis yang ditetapkan sebelumnya serta dapat digunakan secara efektif dalam lingkungan produksi. Dalam pengujian penerimaan pengguna, pengguna akhir akan menjalankan skenario pengujian yang mencerminkan situasi dunia nyata, mengevaluasi fungsionalitas, antarmuka, dan kinerja perangkat lunak. Hasil dari pengujian ini memberikan umpan balik berharga kepada tim pengembangan dan memungkinkan pengguna untuk merasa percaya diri bahwa perangkat lunak siap digunakan. (Padmini C., 2013)

Sebagai contoh, bayangkan pengembangan perangkat lunak manajemen inventaris untuk toko ritel. Dalam pengujian penerimaan pengguna, pengguna akan menggunakan perangkat lunak untuk mengelola stok produk, melakukan pemesanan, dan memantau penjualan sehari-hari. Pengguna akan menjalankan skenario di mana mereka menambahkan produk baru ke dalam sistem, mengatur tingkat stok minimum, dan menghasilkan laporan penjualan. Hasilnya akan dievaluasi untuk memastikan bahwa perangkat lunak dengan akurat merekam transaksi, memberikan laporan yang relevan, dan memudahkan pengguna dalam tugas-tugas mereka sehari-hari. Jika terdapat masalah atau ketidaksesuaian dengan harapan pengguna, tim pengembangan akan melakukan perbaikan yang diperlukan sebelum perangkat lunak diimplementasikan secara luas. Pengujian penerimaan pengguna membantu memastikan bahwa perangkat lunak sesuai dengan kebutuhan dan ekspektasi pengguna akhir, menciptakan pengalaman pengguna yang positif dan produktif.

Pengujian penerimaan pengguna (*User Acceptance Testing* atau UAT) melibatkan beberapa langkah penting untuk memvalidasi bahwa perangkat lunak telah memenuhi persyaratan dan kebutuhan pengguna akhir. Berikut adalah beberapa hal yang dilakukan dalam pengujian UAT beserta contohnya:

1. Perencanaan UAT: Tim pengembangan dan pengguna akhir bekerja sama untuk merencanakan skenario pengujian, membuat daftar persyaratan yang harus diverifikasi, dan menentukan kriteria penerimaan.
   Contoh: Pada proyek perangkat lunak sistem manajemen proyek, perencanaan UAT akan mencakup penentuan skenario pengujian untuk mengelola tugas, mengalokasikan sumber daya, dan memantau kemajuan.
2. Pengembangan Kasus Uji: Pengguna akhir atau pihak berkepentingan mengembangkan skenario pengujian yang mencakup berbagai situasi yang relevan dengan penggunaan sehari-hari.
   Contoh: Pada aplikasi e-learning, kasus uji dapat mencakup pengujian berbagai jenis materi pembelajaran, pengiriman tugas, dan interaksi antara siswa dan instruktur.
3. Eksekusi UAT: Pengguna akhir menjalankan skenario pengujian yang telah disusun, mengikuti langkah-langkah yang ditentukan dan mengamati hasil yang dihasilkan.
   Contoh: Dalam pengujian UAT untuk aplikasi pemesanan makanan online, pengguna akan mencoba memesan makanan dari berbagai restoran, memilih opsi pembayaran, dan memastikan pesanan tiba sesuai harapan.
4. Pengumpulan Umpan Balik: Pengguna akhir memberikan umpan balik terkait pengalaman mereka selama pengujian, mencatat masalah yang mungkin ditemui, dan memberikan saran perbaikan.

Contoh: Pengguna mungkin melaporkan masalah seperti kesalahan tampilan antarmuka atau kesulitan dalam mengakses fitur-fitur tertentu.

5. Pengujian Skala Penuh: Pengguna menjalankan skenario pengujian dalam skala penuh yang mencakup berbagai kasus penggunaan untuk memastikan bahwa perangkat lunak bekerja dengan baik dalam lingkungan produksi.
   Contoh: Pada sistem manajemen pergudangan, pengujian skala penuh akan melibatkan penggunaan data nyata dan mengelola persediaan yang sebenarnya.
6. Penerimaan atau Penolakan: Berdasarkan hasil pengujian, pengguna akhir memberikan keputusan apakah perangkat lunak dapat diterima atau ditolak untuk diimplementasikan.
   Contoh: Setelah menguji perangkat lunak manajemen konferensi, pengguna akhir dapat memberikan keputusan penerimaan jika perangkat lunak memenuhi persyaratan dan berfungsi dengan baik.

Melalui pengujian UAT yang teliti dan berkomitmen, pengguna akhir dapat merasa percaya diri bahwa perangkat lunak telah memenuhi harapan mereka dan siap digunakan dalam lingkungan produksi. Umpan balik yang diberikan oleh pengguna juga memberikan wawasan berharga bagi tim pengembangan untuk meningkatkan dan menyempurnakan perangkat lunak sebelum dirilis kepada publik.

8.10 Pengujian Pengguna Akhir

Gambar 8.10. Icon Pengujian Pengguna Akhir

Pengujian Pengguna Akhir adalah tahap penting dalam siklus pengembangan perangkat lunak di mana pengguna akhir secara langsung terlibat untuk menguji perangkat lunak dan memberikan umpan balik tentang pengalaman mereka dalam menggunakan aplikasi. Tujuan utamanya adalah untuk memastikan bahwa perangkat lunak memenuhi kebutuhan, harapan, dan preferensi pengguna, serta dapat digunakan dengan nyaman dan efektif dalam situasi penggunaan sehari-hari. Dalam pengujian ini, pengguna akhir akan menjalankan skenario pengujian yang mencerminkan aktivitas-aktivitas yang biasanya mereka lakukan dengan perangkat lunak, serta memberikan umpan balik tentang antarmuka, fungsionalitas, dan kinerja aplikasi. Pengujian Pengguna Akhir membantu mengidentifikasi masalah atau ketidaksesuaian dengan harapan pengguna, memastikan bahwa perangkat lunak memberikan pengalaman pengguna yang positif, dan meminimalkan risiko adopsi yang rendah akibat masalah yang tidak diantisipasi. (Padmini C., 2013)

Sebagai contoh, pertimbangkan pengembangan aplikasi seluler untuk pelacakan aktivitas kebugaran. Dalam pengujian

Pengguna Akhir, pengguna akan mengunduh dan menginstal aplikasi, mencoba mengatur tujuan kebugaran pribadi, merekam latihan, melihat statistik, dan mengaktifkan notifikasi. Pengguna akan memberikan umpan balik tentang apakah antarmuka mudah dinavigasi, apakah fitur-fitur seperti pemberitahuan berfungsi dengan baik, dan apakah laporan statistik memberikan informasi yang bermanfaat. Jika pengguna mengalami kesulitan atau menemukan ketidaksesuaian, tim pengembangan dapat melakukan perbaikan sebelum aplikasi dirilis ke publik. Pengujian Pengguna Akhir membantu memastikan bahwa perangkat lunak memenuhi kebutuhan dan ekspektasi pengguna akhir, sehingga menciptakan pengalaman pengguna yang positif dan mendorong adopsi yang sukses.

Selain itu, dalam pengujian Pengguna Akhir, pengguna akhir juga dapat memberikan masukan berharga tentang fitur-fitur yang mungkin belum terpikirkan oleh tim pengembangan. Hal ini dapat membantu perangkat lunak menjadi lebih inovatif dan sesuai dengan kebutuhan yang mungkin tidak teridentifikasi sebelumnya. Dengan melibatkan pengguna akhir dalam pengujian ini, tim pengembangan dapat memastikan bahwa perangkat lunak tidak hanya berfungsi dengan baik secara teknis, tetapi juga memberikan nilai tambah yang sesuai dengan harapan pengguna.

Pengujian Pengguna Akhir melibatkan beberapa langkah untuk mengumpulkan umpan balik dan evaluasi dari pengguna akhir yang menggunakan perangkat lunak. Berikut adalah beberapa hal yang dilakukan dalam pengujian Pengguna Akhir beserta contohnya:

1. Pengembangan Kasus Uji yang Relevan: Pengembang dan pengguna akhir bekerja sama untuk mengembangkan skenario pengujian yang mencakup aktivitas-aktivitas nyata yang akan dilakukan oleh pengguna dalam perangkat lunak.

Contoh: Dalam pengujian Pengguna Akhir untuk aplikasi e-commerce, kasus uji mungkin mencakup pencarian produk, menambahkan item ke keranjang belanja, dan menyelesaikan proses pembayaran.

2. Pengujian Berbasis Skenario: Pengguna akhir menjalankan skenario pengujian yang telah dirancang, mencoba berbagai fitur, interaksi, dan fungsi aplikasi sesuai dengan tujuan penggunaan sehari-hari.
   Contoh: Dalam pengujian Pengguna Akhir untuk aplikasi media sosial, pengguna akan mencoba memposting status, mengunggah foto, dan berinteraksi dengan postingan teman.
3. Pengamatan dan Pengumpulan Data: Tim pengujian mengamati dan mencatat bagaimana pengguna berinteraksi dengan perangkat lunak, menghadapi kendala, atau memberikan umpan balik selama pengujian.
   Contoh: Pengujian Pengguna Akhir pada aplikasi pengaturan jadwal mungkin mengamati apakah pengguna dengan mudah dapat membuat dan mengedit jadwal kegiatan.
4. Wawancara dan Diskusi: Tim pengujian dapat mengadakan wawancara atau diskusi dengan pengguna akhir untuk mendapatkan pandangan mendalam tentang pengalaman mereka dan mendengar masukan langsung.
   Contoh: Setelah menggunakan aplikasi perencanaan perjalanan, pengguna dapat berbicara tentang kemudahan navigasi, kualitas informasi, dan apakah aplikasi membantu mereka dalam merencanakan perjalanan.
5. Pengujian Dalam Kondisi Nyata: Pengguna akhir menjalankan perangkat lunak dalam lingkungan sebenarnya untuk mengamati bagaimana perangkat lunak berperilaku dalam kondisi penggunaan sehari-hari.
   Contoh: Dalam pengujian Pengguna Akhir untuk aplikasi manajemen email, pengguna akan memeriksa apakah

aplikasi berhasil mengelompokkan email sesuai dengan kategori dan mengirimkan pemberitahuan dengan benar.

6. Analisis dan Umpan Balik: Setelah pengujian selesai, umpan balik dan hasil pengamatan dievaluasi untuk mengidentifikasi masalah, kesulitan, atau peluang perbaikan.
   Contoh: Pengguna akhir mungkin memberikan umpan balik bahwa antarmuka tidak intuitif atau bahwa fitur pencarian tidak memberikan hasil yang relevan.

Melalui pengujian Pengguna Akhir, tim pengembangan dapat memahami perspektif dan kebutuhan pengguna secara lebih baik, mengidentifikasi masalah yang mungkin terlewatkan, dan melakukan perbaikan atau penyesuaian sebelum perangkat lunak diimplementasikan secara luas. Ini membantu memastikan bahwa perangkat lunak benar-benar bermanfaat dan sesuai dengan ekspektasi pengguna akhir.

8.11 Pengujian Kompatibilitas

Gambar 8.11. Icon Pengujian Kompatibilitas

Pengujian kompatibilitas adalah tahap penting dalam pengembangan perangkat lunak yang bertujuan untuk

memastikan bahwa aplikasi atau sistem dapat berfungsi dengan baik dan sesuai di berbagai lingkungan perangkat keras, perangkat lunak, dan konfigurasi yang berbeda. Tujuan utamanya adalah untuk memastikan bahwa perangkat lunak dapat berinteraksi dengan sistem operasi, perangkat, peramban web, atau lingkungan lain yang beragam, tanpa mengalami masalah atau gangguan yang signifikan. Dalam pengujian kompatibilitas, berbagai kombinasi perangkat keras, perangkat lunak, dan lingkungan disimulasikan untuk menguji apakah aplikasi tetap berjalan dengan benar, tampilan terlihat sesuai, dan fungsionalitas inti tetap utuh. Pengujian ini membantu memastikan bahwa pengguna dari berbagai latar belakang dan preferensi teknologi dapat mengakses dan menggunakan perangkat lunak dengan kenyamanan dan kepuasan.

Sebagai contoh, bayangkan sebuah aplikasi web yang dikembangkan untuk platform e-commerce. Pengujian kompatibilitas akan mencakup pengujian di berbagai peramban web yang populer, seperti Chrome, Firefox, dan Safari, untuk memastikan tampilan dan fungsionalitas tetap konsisten. Selain itu, aplikasi ini juga akan diuji di berbagai sistem operasi, seperti Windows, macOS, dan Linux, serta di berbagai perangkat, termasuk desktop, tablet, dan ponsel pintar. Tim pengembangan akan memastikan bahwa antarmuka masih terlihat baik dan fungsionalitas utama, seperti pencarian produk dan proses pembayaran, berjalan tanpa masalah di setiap kombinasi ini. Dengan melakukan pengujian kompatibilitas yang cermat, tim pengembangan dapat menghindari masalah yang mungkin terjadi saat perangkat lunak digunakan oleh pengguna dengan perangkat dan lingkungan yang berbeda-beda.

Selain itu, pengujian kompatibilitas juga mencakup pengujian integrasi dengan sistem atau aplikasi lain yang mungkin digunakan bersama dengan perangkat lunak yang

sedang dikembangkan. Misalnya, dalam kasus aplikasi manajemen keuangan, pengujian akan mencakup integrasi dengan sistem perbankan atau perangkat lunak pajak. Pengujian kompatibilitas membantu memastikan bahwa perangkat lunak dapat beroperasi dengan lancar dan berinteraksi dengan sistem lain tanpa adanya masalah kompatibilitas yang dapat menghambat fungsionalitas keseluruhan.

Pengujian kompatibilitas melibatkan serangkaian langkah untuk memastikan bahwa perangkat lunak dapat berfungsi dengan baik di berbagai lingkungan dan kondisi yang berbeda. Berikut adalah beberapa hal yang dilakukan dalam pengujian kompatibilitas beserta contohnya:

1. Pengujian Peramban Web: Aplikasi web diuji pada berbagai peramban web yang berbeda, termasuk Chrome, Firefox, Safari, dan Edge, untuk memastikan tampilan dan fungsionalitas yang konsisten.
   Contoh: Pengujian mengungkapkan bahwa tombol tampilan atau elemen antarmuka lainnya mungkin berperilaku berbeda di peramban tertentu, sehingga perlu disesuaikan.
2. Pengujian Sistem Operasi: Perangkat lunak diuji pada berbagai sistem operasi, seperti Windows, macOS, dan Linux, untuk memastikan kompatibilitas platform.
   Contoh: Pengujian mengidentifikasi bahwa perangkat lunak mengalami kesalahan saat dijalankan di sistem operasi tertentu, dan perlu ada perbaikan untuk memastikan kelancaran operasi.
3. Pengujian Perangkat: Aplikasi diuji di berbagai perangkat keras, termasuk desktop, laptop, tablet, dan ponsel pintar, untuk memastikan tampilan dan fungsionalitas yang sesuai.

Contoh: Pengujian menunjukkan bahwa tata letak antarmuka rusak atau tidak nyaman dilihat pada perangkat layar kecil, sehingga perlu dilakukan penyesuaian desain.

4. Pengujian Integrasi dengan Aplikasi Lain: Jika perangkat lunak berinteraksi dengan aplikasi lain, pengujian dilakukan untuk memastikan kompatibilitas dan integrasi yang baik.
   Contoh: Pengujian mengungkapkan bahwa perangkat lunak manajemen kepegawaian tidak berintegrasi dengan sistem penggajian, sehingga data tidak sinkron.
5. Pengujian Jaringan: Pengujian dilakukan di berbagai jenis jaringan, termasuk koneksi internet berkecepatan rendah atau tinggi, untuk memastikan responsivitas perangkat lunak.
   Contoh: Pengujian menunjukkan bahwa waktu muat laman web lebih lambat pada koneksi internet yang lambat, sehingga diperlukan optimisasi.
6. Pengujian Perangkat Tambahan: Perangkat lunak diuji dengan perangkat keras atau perangkat lunak tambahan yang mungkin digunakan oleh pengguna, seperti printer atau perangkat penyimpanan eksternal.
   Contoh: Pengujian mengidentifikasi bahwa perangkat lunak tidak dapat berkomunikasi dengan printer tertentu, sehingga perlu diberikan dukungan.

Melalui pengujian kompatibilitas yang cermat, tim pengembangan dapat memastikan bahwa perangkat lunak dapat berfungsi dengan baik di berbagai lingkungan dan kondisi, serta memberikan pengalaman pengguna yang konsisten dan memuaskan.

8.12 Pengujian Penyusupan

Gambar 8.12. Icon Pengujian Penyusupan

Pengujian penyusupan (*intrusion testing*) adalah proses penting dalam pengembangan perangkat lunak yang bertujuan untuk mengidentifikasi kerentanan keamanan yang dapat dieksploitasi oleh peretas atau penyusup. Tujuan utamanya adalah untuk mengevaluasi sejauh mana perangkat lunak tahan terhadap serangan yang bertujuan untuk mengakses informasi sensitif, mengganggu fungsionalitas, atau merusak integritas sistem. Dalam pengujian ini, teknik dan metode yang digunakan oleh peretas sebenarnya diuji pada perangkat lunak untuk menguji reaksi dan respons sistem terhadap ancaman potensial. Pengujian penyusupan membantu mengidentifikasi celah keamanan yang mungkin terlewatkan selama pengembangan dan memungkinkan tim pengembangan untuk mengambil tindakan korektif untuk meningkatkan keamanan perangkat lunak.

Contoh pengujian penyusupan meliputi serangan injeksi SQL, di mana perangkat lunak diserang dengan menginjeksikan perintah SQL berbahaya melalui input pengguna untuk mencoba mengakses atau merusak basis data. Metode lainnya adalah uji penetrasi, di mana pengujian secara aktif mencoba

memanfaatkan kerentanan yang ditemukan, seperti mencoba meretas kata sandi atau mengakses data tanpa izin. Pengujian penyusupan juga mencakup pengujian perusakan, di mana sistem diuji untuk melihat sejauh mana perangkat lunak mampu berfungsi setelah mengalami serangan atau kerusakan yang disengaja. Dengan melaksanakan pengujian penyusupan secara komprehensif, tim pengembangan dapat mengidentifikasi dan mengatasi kerentanan keamanan yang mungkin ada dalam perangkat lunak sebelum dirilis ke publik, menjaga integritas data, dan melindungi perangkat lunak dari ancaman yang mungkin timbul.

Pengujian penyusupan melibatkan upaya untuk secara aktif mengevaluasi perangkat lunak dengan cara mensimulasikan serangan yang dilakukan oleh peretas atau penyusup nyata. Berikut adalah beberapa tindakan yang dilakukan dalam pengujian penyusupan beserta contohnya:

1. Identifikasi Kerentanan: Tim pengujian akan menganalisis kode perangkat lunak, konfigurasi, dan komponen lainnya untuk mengidentifikasi potensi kerentanan keamanan yang dapat dimanfaatkan.
   Contoh: Dalam pengujian penyusupan pada aplikasi perbankan online, tim pengujian mungkin mengidentifikasi input yang tidak divalidasi dengan baik yang dapat menyebabkan serangan injeksi SQL.
2. Uji Injeksi: Pengujian akan mencoba menginjeksi data atau perintah berbahaya ke dalam perangkat lunak melalui input pengguna untuk melihat apakah perangkat lunak mengolahnya dengan benar.
   Contoh: Pengujian penyusupan mencoba menginjeksi kode skrip jahat ke dalam formulir input pada situs web, dengan tujuan mencuri informasi pengguna.
3. Pengujian Penetrasi: Teknik penetrasi digunakan untuk mencoba memanfaatkan kerentanan yang telah

diidentifikasi, seperti mencoba meretas kata sandi atau mendapatkan akses yang tidak sah.
Contoh: Pengujian ini mungkin mencoba menggunakan serangan brute force untuk menebak kata sandi akun pengguna dengan mencoba berbagai kombinasi.

4. Pengujian Serangan DDoS: Pengujian dapat mencoba simulasi serangan Distributed Denial of Service (DDoS) untuk menguji ketahanan perangkat lunak terhadap lonjakan lalu lintas yang tinggi.
   Contoh: Pengujian penyusupan akan mencoba membanjiri situs web dengan permintaan palsu hingga menyebabkan situs menjadi tidak responsif.
5. Pengujian Manajemen Sesi: Pengujian akan menguji manajemen sesi dan otorisasi untuk memastikan bahwa akses pengguna dikelola dengan benar dan hanya diberikan kepada pengguna yang sah.
   Contoh: Pengujian penyusupan mencoba memodifikasi ID sesi untuk mendapatkan akses yang tidak sah ke akun pengguna.
6. Pengujian Pemulihan Bencana: Pengujian juga dapat mencoba menguji kemampuan perangkat lunak dalam menghadapi serangan atau bencana, dan sejauh mana perangkat lunak dapat pulih setelah terjadi serangan.
   Contoh: Pengujian penyusupan mencoba merusak database dan menguji kemampuan perangkat lunak untuk mengembalikan data dari cadangan.

Melalui pengujian penyusupan yang hati-hati, tim pengembangan dapat mengidentifikasi dan mengatasi kerentanan keamanan yang ada dalam perangkat lunak sebelum perangkat lunak diimplementasikan secara luas, sehingga memastikan tingkat keamanan yang lebih tinggi dan mengurangi risiko terkait ancaman keamanan.

8.13 Pengujian Otomatis

Gambar 8.13. Icon Pengujian Otomatis

Pengujian otomatis adalah pendekatan yang melibatkan penggunaan alat dan skrip otomatis untuk menjalankan dan mengulangi serangkaian pengujian pada perangkat lunak. Tujuan utamanya adalah untuk mengotomatisasi proses pengujian guna meningkatkan efisiensi, konsistensi, dan akurasi pengujian. Dalam pengujian otomatis, skenario pengujian dibuat dalam bentuk kode atau skrip, yang dapat menjalankan serangkaian aksi dan verifikasi dengan cepat. Pengujian otomatis dapat mencakup pengujian fungsional, pengujian performa, pengujian regresi, dan banyak lagi. Keuntungan utama dari pengujian otomatis meliputi penghematan waktu dan sumber daya, memungkinkan pengujian yang lebih sering, dan dapat membantu mengidentifikasi masalah dengan cepat, sehingga memastikan kualitas perangkat lunak yang lebih baik.

Sebagai contoh, dalam pengujian otomatis pengujian fungsional, tim pengembangan dapat menggunakan alat otomatis untuk mengulangi serangkaian langkah, seperti mengisi formulir, mengklik tautan, dan memverifikasi hasil

yang diharapkan. Jika ada perubahan pada kode perangkat lunak, skrip otomatis dapat dijalankan kembali untuk memastikan bahwa perubahan tersebut tidak memengaruhi fungsi yang sudah ada sebelumnya. Pengujian otomatis ini membantu mengurangi risiko regresi dan mempercepat siklus pengembangan. Dalam pengujian performa, skenario pengujian yang kompleks dan repetitif dapat diotomatisasi untuk menghasilkan beban pada perangkat lunak dan mengamati kinerjanya dalam situasi yang berbeda. Dengan pengujian otomatis performa, tim pengembangan dapat mengidentifikasi batas kemampuan sistem dan mengoptimalkan performa untuk menghadapi situasi yang berat.

Pengujian otomatis melibatkan beberapa tindakan untuk mengotomatisasi proses pengujian perangkat lunak. Berikut adalah beberapa hal yang dilakukan dalam pengujian otomatis beserta contohnya:

Pemilihan Alat Otomasi: Tim pengujian memilih alat otomasi yang sesuai untuk kebutuhan pengujian, seperti Selenium untuk pengujian fungsional web, JUnit untuk pengujian unit, atau JMeter untuk pengujian performa.

1. Pengembangan Skrip Pengujian: Skrip pengujian otomatis dikembangkan dengan menggunakan bahasa pemrograman tertentu, yang mencakup langkah-langkah pengujian yang harus dijalankan dan verifikasi hasil yang diharapkan.

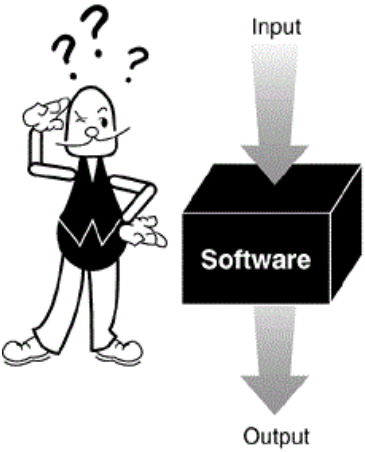
   Contoh: Dalam pengujian fungsional aplikasi e-commerce, skrip otomatis dapat mengulangi langkah-langkah pembelian produk, pengisian formulir, dan verifikasi total harga yang benar.
2. Eksekusi Skrip: Skrip otomatis dijalankan menggunakan alat otomasi yang dipilih, yang akan mengulangi serangkaian aksi pengujian sesuai dengan instruksi yang diberikan.

Contoh: Dalam pengujian regresi, skrip otomatis dijalankan untuk mengkonfirmasi bahwa fitur-fitur yang sebelumnya sudah berfungsi tetap berjalan dengan benar setelah perubahan kode.

3. Verifikasi Hasil: Setelah eksekusi skrip selesai, hasil pengujian otomatis akan diverifikasi dengan hasil yang diharapkan. Perbedaan antara hasil aktual dan yang diharapkan akan ditandai sebagai kegagalan.
   Contoh: Jika skrip otomatis untuk pengujian login menghasilkan pesan kesalahan yang tidak sesuai dengan yang diharapkan, itu akan dianggap sebagai kegagalan.
4. Pemantauan dan Laporan: Proses pengujian otomatis dapat dipantau secara berkala, dan laporan hasil pengujian otomatis dapat dihasilkan untuk memberikan wawasan tentang kinerja perangkat lunak.
   Contoh: Laporan pengujian otomatis dapat menunjukkan persentase skenario pengujian yang berhasil, gagal, atau memerlukan tindakan perbaikan.
5. Perbaikan dan Peningkatan Skrip: Skrip otomatis dapat diperbaiki dan ditingkatkan sesuai dengan hasil pengujian dan umpan balik. Perubahan pada perangkat lunak atau perubahan skenario pengujian juga dapat mengharuskan penyesuaian pada skrip otomatis.
   Contoh: Setelah memperbaiki masalah dalam perangkat lunak, skrip otomatis diperbarui untuk menguji kembali fitur yang telah diperbaiki.

Melalui pengujian otomatis yang tepat dan efektif, tim pengembangan dapat mengidentifikasi masalah dengan cepat, menjaga kualitas perangkat lunak, dan mempercepat siklus pengembangan secara keseluruhan.

8.14 Pengujian Blackbox



Gambar 8.14. Icon Pengujian Blackbox

Pengujian Black Box, juga dikenal sebagai pengujian fungsional atau pengujian berbasis spesifikasi, adalah metode pengujian perangkat lunak yang berfokus pada menguji fungsionalitas dan perilaku eksternal suatu aplikasi tanpa memperhatikan struktur internal atau logika kode. Dalam pengujian ini, penguji hanya memiliki pengetahuan tentang spesifikasi dan fitur yang diharapkan dari perangkat lunak, tanpa mengetahui bagaimana implementasinya dilakukan. Tujuan utamanya adalah untuk mengidentifikasi masalah dan kesalahan yang mungkin terjadi dalam fungsionalitas perangkat lunak dan memastikan bahwa perangkat lunak beroperasi sesuai dengan harapan. (Padmini C., 2013)

Dalam pengujian Black Box, penguji akan merancang dan menjalankan serangkaian skenario pengujian berdasarkan input yang mungkin diberikan oleh pengguna atau lingkungan operasional. Penguji kemudian akan membandingkan hasil yang dihasilkan oleh perangkat lunak dengan hasil yang

diharapkan sesuai dengan spesifikasi. Jika ada perbedaan antara hasil aktual dan hasil yang diharapkan, masalah atau cacat dalam fungsionalitas dapat diidentifikasi dan dilaporkan kepada tim pengembangan untuk diperbaiki.

Contohnya, pertimbangkan sebuah aplikasi pemesanan tiket pesawat online. Dalam pengujian Black Box, penguji akan menguji berbagai fitur seperti pencarian penerbangan, pemilihan tempat duduk, dan pembayaran. Mereka mungkin mencoba skenario berbeda, seperti penerbangan dengan tujuan dan tanggal yang berbeda, dan memeriksa apakah aplikasi memberikan hasil yang benar sesuai dengan permintaan mereka. Jika penguji menemukan bahwa pencarian penerbangan tidak menghasilkan hasil yang akurat atau pemilihan tempat duduk bermasalah, mereka akan melaporkan temuan tersebut kepada tim pengembangan untuk diperbaiki.

Pengujian Black Box sangat berguna dalam mengidentifikasi masalah yang mungkin tidak terdeteksi melalui pengujian lain, serta memberikan pandangan objektif tentang bagaimana pengguna akan berinteraksi dengan perangkat lunak. Meskipun tidak mengungkapkan detail internal perangkat lunak, metode ini membantu memastikan bahwa perangkat lunak memenuhi harapan pengguna dan memenuhi persyaratan fungsional yang telah ditetapkan.

Berikut adalah contoh skenario pengujian Black Box beserta contohnya untuk aplikasi pemesanan makanan online:

Skenario Pengujian: Pemesanan Menu

Deskripsi: Pengujian ini akan menguji kemampuan aplikasi untuk memproses pemesanan makanan oleh pengguna.

Langkah-langkah Pengujian:

1. Pengguna membuka aplikasi dan masuk ke akunnya.
2. Pengguna memilih restoran dari daftar yang tersedia.

3. Pengguna memilih beberapa item makanan dari menu dan menambahkannya ke keranjang belanja.
4. Pengguna melihat keranjang belanja dan memverifikasi item-item yang dipilih.
5. Pengguna melanjutkan ke halaman pembayaran, memasukkan rincian pembayaran, dan mengonfirmasi pemesanan.

Contoh Hasil yang Diinginkan:

1. Setelah langkah 3, item makanan yang dipilih muncul di keranjang belanja dengan jumlah yang benar.
2. Setelah langkah 4, pengguna melihat daftar item di keranjang belanja dengan harga total yang benar.
3. Setelah langkah 5, pengguna menerima konfirmasi pemesanan dengan nomor pesanan yang unik.

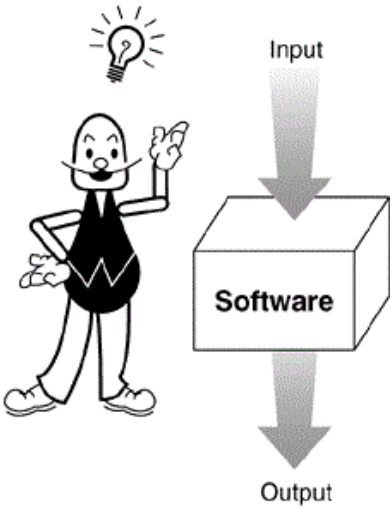
Contoh Hasil yang Tidak Diinginkan:

1. Setelah langkah 3, item makanan tidak muncul di keranjang belanja atau muncul dengan jumlah yang salah.
2. Setelah langkah 4, total harga di keranjang belanja tidak akurat.
3. Setelah langkah 5, pengguna tidak menerima konfirmasi pemesanan atau nomor pesanan tidak valid.

Dalam skenario pengujian di atas, penguji akan menjalankan serangkaian langkah sesuai dengan skenario pengujian untuk menguji apakah fungsionalitas pemesanan menu berjalan sesuai dengan yang diharapkan. Penguji akan membandingkan hasil aktual yang mereka lihat di aplikasi dengan hasil yang diharapkan sesuai dengan spesifikasi, dan jika ada perbedaan atau masalah, mereka akan melaporkannya kepada tim pengembangan untuk ditindaklanjuti. Metode pengujian Black Box seperti ini membantu memastikan bahwa

aplikasi pemesanan makanan online berfungsi dengan baik dan memberikan pengalaman yang positif bagi pengguna.

8.15 Pengujian Whitebox



Gambar 8.15. Icon Pengujian Whitebox

Pengujian White Box, juga dikenal sebagai pengujian struktural atau pengujian berbasis kode, adalah metode pengujian perangkat lunak yang berfokus pada pemeriksaan dan evaluasi struktur internal dari suatu aplikasi, termasuk kode sumber, logika, dan aliran eksekusi. Dalam pengujian ini, penguji memiliki pengetahuan tentang implementasi teknis dan rincian internal perangkat lunak, sehingga dapat merancang skenario pengujian yang memeriksa cakupan kode, pengujian jalur eksekusi, dan mengidentifikasi masalah dalam logika atau kerentanan yang mungkin dieksploitasi oleh penyusup. Pengujian White Box bertujuan untuk memastikan bahwa perangkat lunak beroperasi sesuai dengan desain dan spesifikasi, serta mengidentifikasi masalah yang mungkin

timbul dari struktur kode yang tidak benar atau tidak aman. (Padmini C., 2013)

Dalam pengujian White Box, penguji akan melakukan analisis mendalam pada kode sumber dan menggunakan teknik seperti pengujian unit, pengujian integrasi, dan analisis statis untuk mengidentifikasi masalah dan cacat dalam perangkat lunak. Mereka juga dapat melakukan pengujian jalur eksekusi yang melibatkan menjalankan berbagai kombinasi aliran kontrol di dalam kode. Tujuannya adalah untuk memastikan bahwa setiap bagian dari kode telah diuji secara cermat dan berfungsi dengan benar dalam berbagai skenario pengujian.

Contohnya, dalam pengujian White Box, penguji akan menganalisis kode sumber aplikasi e-commerce dan mengidentifikasi bagian kode yang memproses transaksi pembayaran. Mereka akan merancang skenario pengujian yang mencakup berbagai jenis pembayaran, seperti kartu kredit dan transfer bank, serta memeriksa bagaimana kode berinteraksi dengan sistem pembayaran eksternal. Penguji juga dapat mengidentifikasi cacat dalam kode yang dapat menyebabkan masalah dalam proses pembayaran atau mengungkapkan informasi sensitif pengguna.

Pengujian White Box sangat penting dalam mengidentifikasi masalah dalam struktur dan logika perangkat lunak yang mungkin terlewatkan oleh pengujian Black Box atau pengujian fungsional. Meskipun memerlukan pengetahuan teknis yang mendalam, metode ini membantu memastikan bahwa perangkat lunak memiliki kualitas yang tinggi, aman, dan sesuai dengan persyaratan teknis yang telah ditetapkan.

Berikut adalah contoh skenario pengujian White Box beserta contohnya untuk sebuah aplikasi pengelolaan inventaris:

Skenario Pengujian: Penghitungan Total Inventaris

Deskripsi: Pengujian ini akan memeriksa apakah perangkat lunak dapat menghitung total nilai inventaris dengan benar berdasarkan informasi barang dan harga.

Langkah-langkah Pengujian:

1. Penguji memeriksa bagian kode yang bertanggung jawab atas penghitungan total inventaris.
2. Penguji mengidentifikasi variabel atau fungsi yang terlibat dalam perhitungan.
3. Penguji merancang kasus uji yang mencakup beberapa item inventaris dengan harga yang berbeda.
4. Penguji menjalankan kasus uji dan mencatat hasil perhitungan.

Contoh Hasil yang Diinginkan:

1. Jika terdapat tiga item inventaris dengan harga masing-masing $10, $20, dan $30, maka total inventaris dihitung menjadi $60.
2. Jika terdapat lima item inventaris dengan harga masing-masing $15, $25, $35, $45, dan $55, maka total inventaris dihitung menjadi $175.

Contoh Hasil yang Tidak Diinginkan:

1. Jika perhitungan tidak memasukkan harga item inventaris yang salah atau mengabaikan beberapa item, total inventaris yang dihitung akan tidak akurat.
2. Jika ada kesalahan logika dalam kode yang mengakibatkan penghitungan yang salah, hasil perhitungan total inventaris dapat menjadi tidak valid.

Dalam skenario pengujian di atas, penguji akan menganalisis bagian kode yang bertanggung jawab atas

penghitungan total inventaris dan merancang kasus uji untuk menguji berbagai kemungkinan kondisi dan input. Mereka akan membandingkan hasil perhitungan yang dihasilkan oleh kode dengan hasil yang diharapkan sesuai dengan skenario pengujian. Jika ada perbedaan atau ketidaksesuaian, penguji akan melaporkannya kepada tim pengembangan untuk ditindaklanjuti dan diperbaiki.

Pengujian White Box seperti ini membantu memastikan bahwa logika dan perhitungan dalam kode perangkat lunak berfungsi dengan benar dan menghasilkan hasil yang konsisten dan akurat. Ini membantu mencegah terjadinya masalah dalam pengelolaan inventaris yang dapat mempengaruhi keputusan bisnis dan kehandalan sistem.

8.16 Pengujian Greybox

Gambar 8.16. Icon Pengujian Greybox

Pengujian Grey Box adalah metode pengujian perangkat lunak yang menggabungkan elemen-elemen dari pengujian Black Box dan White Box. Dalam pengujian ini, penguji memiliki sebagian pengetahuan tentang struktur internal dan logika kode perangkat lunak, namun tidak memiliki

pengetahuan yang lengkap seperti dalam pengujian White Box. Pengujian Grey Box berusaha untuk menggabungkan keuntungan dari kedua pendekatan tersebut, yaitu mengidentifikasi masalah fungsional dari luar seperti pada pengujian Black Box dan juga menggali masalah yang mungkin ada dalam struktur dan logika kode seperti pada pengujian White Box. (Padmini C., 2013)

Pada dasarnya, pengujian Grey Box dapat melibatkan analisis kode untuk memahami aliran eksekusi atau kerentanan potensial, sementara sebagian besar fungsionalitas diuji berdasarkan spesifikasi dan fitur yang diharapkan dari perangkat lunak. Teknik-teknik seperti pengujian penetrasi, pengujian integrasi, dan analisis jalur eksekusi dapat digunakan dalam pengujian Grey Box untuk mengidentifikasi masalah keamanan dan fungsionalitas.

Contohnya, bayangkan aplikasi jejaring sosial yang memiliki fitur pengiriman pesan pribadi. Dalam pengujian Grey Box, penguji mungkin memiliki pengetahuan tentang bagaimana pesan-pesan tersebut disimpan dan diolah di sisi server (informasi yang biasanya tidak diketahui dalam pengujian *Black Box*). Namun, mereka tidak memiliki akses penuh ke kode sumber atau detail teknis yang ada di dalamnya (seperti dalam pengujian *White Box*). Dalam pengujian ini, penguji dapat mengirimkan berbagai jenis pesan (teks, gambar, tautan) untuk mengidentifikasi masalah potensial dalam pengiriman, penyimpanan, atau tampilan pesan, sambil juga melihat bagaimana pesan-pesan tersebut dapat mempengaruhi sistem secara internal.

Pengujian Grey Box sangat berguna untuk menggali masalah yang mungkin terlewatkan dalam pengujian Black Box, sambil tetap mempertahankan pandangan yang lebih mirip dengan pengalaman pengguna akhir. Ini membantu dalam mengidentifikasi masalah fungsionalitas dan keamanan yang

mungkin ada dalam perangkat lunak sebelum perangkat lunak dirilis ke publik.

Berikut adalah contoh skenario pengujian Grey Box beserta contohnya untuk aplikasi manajemen keuangan pribadi:

Skenario Pengujian: Analisis Pengeluaran Bulanan

Deskripsi: Pengujian ini akan menguji kemampuan aplikasi untuk menganalisis pengeluaran bulanan pengguna berdasarkan data transaksi yang dimasukkan.

Langkah-langkah Pengujian:

1. Penguji memiliki akses terbatas ke struktur tabel database yang menyimpan data transaksi keuangan.
2. Penguji merancang serangkaian transaksi dengan jumlah dan kategori yang berbeda-beda.
3. Penguji memasukkan data transaksi ke dalam aplikasi.
4. Penguji menjalankan fitur analisis pengeluaran bulanan dan memeriksa hasil yang dihasilkan.

Contoh Hasil yang Diinginkan:

1. Jika terdapat transaksi sejumlah $100 dengan kategori "Makanan" dan $50 dengan kategori "Transportasi", maka aplikasi harus dapat menghitung total pengeluaran bulanan untuk makanan dan transportasi dengan benar.
2. Jika terdapat transaksi dengan tanggal yang berbeda-beda, aplikasi harus dapat mengelompokkan dan menganalisis pengeluaran berdasarkan bulan yang sesuai.

Contoh Hasil yang Tidak Diinginkan:

1. Jika aplikasi tidak dapat menghitung total pengeluaran dengan benar atau memberikan analisis yang salah, hasil yang dihasilkan tidak akan sesuai dengan harapan.
2. Jika aplikasi tidak mampu mengelompokkan atau menganalisis pengeluaran berdasarkan kriteria yang benar, laporan analisis pengeluaran bulanan dapat menjadi tidak akurat.

Dalam skenario pengujian di atas, penguji memiliki pengetahuan terbatas tentang struktur database yang menyimpan data transaksi, tetapi tidak mengetahui semua detail implementasinya. Penguji akan merancang skenario pengujian yang menguji fitur analisis pengeluaran bulanan dan memeriksa apakah hasil yang dihasilkan sesuai dengan harapan. Dengan pendekatan Grey Box, penguji dapat mengidentifikasi masalah fungsionalitas dan keakuratan analisis pengeluaran, sambil tetap mempertahankan batasan pengetahuan tentang struktur internal aplikasi.

DAFTAR PUSTAKA

Myers, G.J., Badgett, T. and Sandler, C. 2011. 'Software Testing Tutorial', pp. 2–7.

Padmini C. 2013. 'Beginners Guide To Software Testing', *Software Testing Class*, pp. 1–41.

BAB 9
EVOLUSI PERANGKAT LUNAK

Oleh Arief Yanto Rukmana

9.1 Pendahuluan

Rekayasa perangkat lunak, seni dan ilmu membangun sistem digital yang kompleks, telah menyaksikan kemajuan luar biasa selama bertahun-tahun (Romindo *et al.,* n.d.). Dari baris kode paling awal yang menghidupkan mesin primitif hingga aplikasi canggih yang kini menembus setiap aspek kehidupan kita, perangkat lunak telah menjadi bagian tak terpisahkan dari dunia modern (Acuña *et al*., 2012). Namun, perjalanan perangkat lunak tidak berakhir pada pembuatannya. Nyatanya, itu baru permulaan.

Selamat datang di ranah evolusi perangkat lunak, di mana kreasi digital dari kecerdikan manusia mengalami metamorfosis adaptasi, pertumbuhan, dan transformasi (Wakil et al., 2022). Evolusi perangkat lunak mewujudkan esensi entitas hidup, yang terus-menerus merespons tuntutan perubahan lingkungan dan pemangku kepentingannya. Ini adalah kisah inovasi, ketahanan, dan peningkatan berkelanjutan yang terus berkembang.

Dalam bab ini, akan mempelajari dunia evolusi perangkat lunak yang menawan, menjelajahi signifikansi, proses, tantangan, dan praktik terbaiknya (Wu *et al.,* 2016). Di mulai dari pencarian untuk mengungkap seluk-beluk memelihara dan meningkatkan sistem perangkat lunak untuk memastikan umur panjang dan relevansinya. Sama seperti organisme berevolusi untuk bertahan hidup dan berkembang, perangkat lunak mengalami proses serupa, beradaptasi dengan lanskap harapan pengguna yang terus berubah, kemajuan

teknologi, dan kebutuhan bisnis yang berkembang (Rukmana, 2023).

Kebutuhan akan evolusi perangkat lunak muncul dari banyak faktor. Seiring bertambahnya usia perangkat lunak, ia menjadi rentan terhadap masalah yang tidak terduga, apakah itu bug fungsional, kerentanan keamanan, atau hambatan kinerja. Selain itu, persyaratan pengguna dan pemangku kepentingan yang selalu berubah menuntut pembaruan dan peningkatan yang konstan untuk memberikan pengalaman pengguna yang optimal. Kemajuan pesat dalam teknologi semakin mendorong kebutuhan akan evolusi perangkat lunak, karena sistem yang ketinggalan zaman berisiko menjadi usang dalam menghadapi paradigma dan alat yang muncul (Adhy *et al.,* 2019).

Namun, jalur evolusi perangkat lunak bukannya tanpa tantangan. Kompleksitas tumbuh, ketergantungan menumpuk, dan risiko konsekuensi yang tidak diinginkan membayangi. Insinyur perangkat lunak harus menavigasi perairan berbahaya dari sistem warisan, hutang teknis, dan persyaratan yang saling bertentangan untuk memastikan evolusi yang berhasil. Keseimbangan halus antara inovasi dan stabilitas menjadi tali pengikat yang dilalui oleh para insinyur perangkat lunak, berusaha memperkenalkan fitur-fitur baru tanpa mengorbankan keandalan dan integritas sistem.

Untuk mengatasi tantangan ini, bab ini menyajikan eksplorasi yang komprehensif dari proses evolusi perangkat lunak. Kami mempelajari berbagai fase, termasuk analisis, perencanaan, implementasi, pengujian, dan penerapan, menyoroti teknik, alat, dan metodologi yang mendukung evolusi yang efisien dan efektif. Mengungkap seni pemeliharaan perangkat lunak, memeriksa berbagai jenis aktivitas pemeliharaan, dan mendiskusikan praktik terbaik untuk mempertahankan kesehatan perangkat lunak (Vassiliadis *et al.,* 2023).

Selain itu, penting juga mempelajari ranah pemfaktoran ulang perangkat lunak, teknik penting untuk merestrukturisasi dan meningkatkan desain internal sistem perangkat lunak tanpa mengubah perilaku eksternalnya. Tulisan ini menjelajahi manfaat dan tantangan pemfaktoran ulang dan mempelajari seluk-beluk antara mengadaptasi basis kode yang ada dan menjaga stabilitas perangkat lunak (Adhy *et al.,* 2019).

Perlunya mendalami bidang pengelolaan utang teknis, sebuah analogi untuk pertukaran yang dilakukan selama pengembangan perangkat lunak yang menimbulkan biaya di masa mendatang. Mengungkap strategi (Gunawan, 2021) untuk mengukur, memprioritaskan, dan melunasi hutang teknis, memastikan bahwa sistem perangkat lunak berkembang dengan risiko yang lebih rendah dan pemeliharaan yang lebih baik.

Signifikansi desain dan arsitektur evolusioner juga dieksplorasi, menekankan pentingnya struktur perangkat lunak yang fleksibel dan dapat diskalakan yang dapat dengan anggun mengakomodasi perubahan di masa depan. Mengulas dan membahas pola desain, gaya arsitektur, dan strategi untuk mengembangkan arsitektur perangkat lunak dari waktu ke waktu, memungkinkan sistem perangkat lunak beradaptasi dan berkembang dalam menghadapi perubahan (Romindo et al., n.d.).

Uraian tulisan ini mengeksplorasi peran penting dari continuous integration and deployment (CI/CD) dalam evolusi perangkat lunak, memungkinkan pengiriman pembaruan perangkat lunak yang cepat dan andal (N. Y. Setiawan, n.d.). Dalam bagian ini juga menyelidiki sistem kontrol versi, mengubah strategi manajemen, dan praktik pengujian dan jaminan kualitas yang diperlukan untuk memastikan evolusi sistem perangkat lunak yang mulus.

Sepanjang paparan bab ini, ditekankan pentingnya pemantauan dan umpan balik dalam evolusi perangkat lunak.

Dengan mengumpulkan dan menganalisis data tentang penggunaan dan kinerja perangkat lunak, para insinyur mendapatkan wawasan yang sangat berharga yang menginformasikan pengambilan keputusan dan mendorong peningkatan berulang (Yanto Rukmana et al., 2021). Penting untuk mengeksplorasi teknik untuk mengumpulkan umpan balik dan mendemonstrasikan bagaimana hal itu dapat dimanfaatkan untuk mengoptimalkan sistem perangkat lunak dan meningkatkan pengalaman pengguna.

Berdasarkan studi kasus dan contoh dunia nyata, tulisan ini menyajikan wawasan praktis dan pelajaran dari proyek evolusi perangkat lunak yang sukses (Mayefis et al., 2023a). Kisah-kisah ini menghidupkan konsep dan metodologi yang didiskusikan, memberikan inspirasi dan panduan bagi insinyur perangkat lunak yang menavigasi lanskap kompleks evolusi perangkat lunak.

Saat memulai perjalanan ini melalui dunia evolusi perangkat lunak yang menawan, sangat menarik untuk menggali dan mempelajari lebih dalam esensi perangkat lunak sebagai entitas yang dinamis, yang terus berkembang dan beradaptasi dengan kebutuhan penggunanya dan dunia tempatnya beroperasi. Bersama-sama, mari menggali lebih dalam serta membuka potensi evolusi perangkat lunak dan rangkul seni inovasi abadi, memastikan ketahanan dan umur panjang kreasi digital (Harto *et al.,* 2022).

9.2 Memahami Evolusi Perangkat Lunak

Sistem perangkat lunak, seperti organisme hidup, memiliki kemampuan luar biasa untuk beradaptasi, tumbuh, dan bertransformasi dari waktu ke waktu. Proses di mana perangkat lunak mengalami metamorfosis ini dikenal sebagai evolusi perangkat lunak. Pada bagian ini, kita menyelidiki kedalaman pemahaman evolusi perangkat lunak,

mengeksplorasi signifikansinya, prinsip-prinsip yang mendasarinya, dan berbagai faktor yang mendorong kebutuhannya (Chakroborti et al., 2023).

Evolusi perangkat lunak mencakup seluruh siklus hidup sistem perangkat lunak, dari awal hingga penghentiannya. Ini mengakui keniscayaan perubahan, merangkul gagasan bahwa perangkat lunak, setelah digunakan, akan terus berkembang untuk memenuhi kebutuhan pengguna dan pemangku kepentingan yang terus berkembang (Finandhita, 2018). Evolusi ini dapat bermanifestasi dalam berbagai bentuk, termasuk perbaikan bug, penyempurnaan, pengoptimalan kinerja, dan bahkan perombakan arsitektur lengkap (Mens et al., 2005).

Inti dari evolusi perangkat lunak terletak pada pengakuan bahwa perangkat lunak tidak statis. Sebaliknya, itu tunduk pada sifat dinamis lingkungannya, termasuk harapan pengguna yang berubah, kemajuan teknologi, dan persyaratan bisnis yang berkembang. Dengan demikian, sistem perangkat lunak harus terus beradaptasi agar tetap relevan, mempertahankan efisiensinya, dan memberikan nilai yang diinginkan kepada penggunanya (N. Y. Setiawan, n.d.).

Salah satu pendorong utama evolusi perangkat lunak adalah perubahan kebutuhan pengguna dan pemangku kepentingan. Ketika organisasi tumbuh, kebutuhan pengguna berkembang, dan pergeseran dinamika pasar, perangkat lunak harus dimodifikasi untuk mencerminkan perubahan ini. Fitur dan fungsionalitas baru mungkin diperlukan, sedangkan fitur yang ada mungkin memerlukan peningkatan atau perubahan. Memahami dan secara efektif menanggapi persyaratan yang berubah ini sangat penting untuk keberhasilan dan umur panjang sistem perangkat lunak.

Kemajuan teknologi juga memainkan peran penting dalam evolusi perangkat lunak. Saat alat, kerangka kerja, perpustakaan, dan paradigma pemrograman baru muncul,

perekayasa perangkat lunak harus beradaptasi dan merangkul inovasi ini agar tetap kompetitif (Adhy *et al.,* 2019). Mengabaikan kemajuan teknologi dapat menyebabkan perangkat lunak menjadi usang, kurang efisien, atau tidak kompatibel dengan platform atau perangkat baru. Oleh karena itu, mengikuti lanskap teknologi yang terus berkembang adalah hal terpenting untuk evolusi perangkat lunak yang berhasil.

Aspek kritis lain dari evolusi perangkat lunak adalah pengakuan bahwa sistem perangkat lunak tidak kebal terhadap penuaan dan degradasi (Romindo *et al.,* n.d.). Seiring waktu, perangkat lunak mungkin mengalami bug fungsional, hambatan kinerja, kerentanan keamanan, atau kualitas kode yang memburuk. Masalah ini dapat muncul karena akumulasi perubahan, pengenalan kode baru, atau degradasi kode yang sudah ada. Proses evolusi perangkat lunak melibatkan identifikasi dan perbaikan masalah tersebut, memastikan bahwa perangkat lunak tetap kuat, aman, dan dapat dipelihara.

Namun, memahami evolusi perangkat lunak lebih dari sekadar mengenali kebutuhannya. Ini membutuhkan pendekatan sistematis dan pemahaman mendalam tentang prinsip dan proses yang mendasari yang terlibat (Vassiliadis *et al.,* 2023). Evolusi perangkat lunak biasanya mengikuti pola siklus, yang terdiri dari beberapa fase seperti analisis, perencanaan, implementasi, pengujian, dan penyebaran. Setiap fase memainkan peran penting dalam memastikan evolusi yang sukses, mulai dari memahami keadaan sistem perangkat lunak saat ini hingga menyusun peta jalan untuk pengembangannya di masa mendatang (Z. Setiawan *et al*., 2023).

Sepanjang proses evolusi, berbagai teknik dan alat dapat membantu perekayasa perangkat lunak dalam mengelola dan mengatur perubahan. Ini mungkin termasuk sistem kontrol versi, kerangka kerja pengujian otomatis, praktik pemfaktoran ulang kode, dan metodologi pemeliharaan perangkat lunak. Alat dan teknik ini menyediakan infrastruktur yang diperlukan

untuk menerapkan dan mengelola evolusi perangkat lunak secara efektif (Rukmana *et al*., 2023).

Keutamaan memahami evolusi perangkat lunak juga memerlukan pengenalan tantangan dan risiko yang terkait dengan proses tersebut. Mengelola sistem perangkat lunak yang kompleks selama evolusi membutuhkan pertimbangan cermat terhadap ketergantungan, masalah kompatibilitas, dan potensi regresi. Menyeimbangkan kebutuhan akan inovasi (Akhmad Al Aidhi, 2023) dengan kebutuhan akan stabilitas dan keandalan adalah tugas rumit yang menuntut keahlian, disiplin, dan perencanaan yang cermat (Razali *et al.,* 2023).

Untuk mengatasi tantangan ini, perekayasa perangkat lunak menggunakan strategi seperti desain modular, kopling longgar, dan abstraksi. Prinsip-prinsip arsitektur ini memungkinkan sistem perangkat lunak untuk berkembang dengan anggun, mengisolasi perubahan pada modul atau komponen tertentu tanpa menyebabkan efek berjenjang di seluruh sistem. Selain itu, konsep hutang teknis menyoroti kebutuhan untuk mengatasi keputusan desain yang kurang optimal atau jalan pintas yang dibuat selama pengembangan, memastikan pemeliharaan dan skalabilitas jangka panjang (Finandhita, 2018).

Memahami evolusi perangkat lunak melampaui aspek teknis pengembangan. Ini memerlukan perspektif holistik yang mencakup pengalaman pengguna, konteks bisnis, dan tujuan dan sasaran keseluruhan dari sistem perangkat lunak (Akhmad Al Aidhi, 2023; Sudirjo *et al.,* 2023). Insinyur perangkat lunak harus menyadari nilai yang dimaksudkan yang diberikan perangkat lunak kepada pengguna dan pemangku kepentingannya, menyelaraskan proses evolusi dengan strategi bisnis menyeluruh (Chusumastuti *et al*., 2023; Sugiarto *et al.,* 2023).

Mendalami dan memahami evolusi perangkat lunak sangat penting untuk menavigasi lanskap pengembangan

perangkat lunak yang selalu berubah. Ini melibatkan pengakuan perubahan yang tak terhindarkan, memahami pendorong evolusi, dan mengadopsi pendekatan sistematis untuk mengelola perubahan secara efektif. Dengan merangkul prinsip dan praktik evolusi perangkat lunak, perekayasa perangkat lunak dapat memastikan umur panjang, kemampuan beradaptasi, dan kesuksesan berkelanjutan dari sistem perangkat lunak yang mereka buat.

9.3 Pemeliharaan Perangkat Lunak

9.3.1 Mempertahankan Kesehatan dan Kinerja Perangkat Lunak

Mempertahankan Kesehatan dan Kinerja Perangkat Lunak, Pemeliharaan perangkat lunak merupakan bagian integral dari siklus hidup pengembangan perangkat lunak, berfokus pada aktivitas yang diperlukan untuk mempertahankan kesehatan, kinerja, dan kegunaan sistem perangkat lunak selama masa operasionalnya (Jiménez-Ramírez *et al*., 2023). Ini mencakup berbagai tugas, mulai dari mengatasi bug dan cacat perangkat lunak hingga memperkenalkan perangkat tambahan dan beradaptasi dengan lingkungan yang berubah. Pada bagian ini, kami menjelajahi seluk-beluk pemeliharaan perangkat lunak, jenisnya, tantangannya, dan praktik terbaik yang digunakan untuk memastikan kelangsungan sistem perangkat lunak dalam jangka Panjang (Acuña et al., 2012).

1. ***Pemeliharaan Korektif***: Pemeliharaan korektif melibatkan identifikasi dan perbaikan cacat perangkat lunak, juga dikenal sebagai bug atau kesalahan. Jenis pemeliharaan ini bertujuan untuk mengembalikan perangkat lunak ke perilaku yang diharapkan dan memastikan keandalan dan stabilitasnya. Insinyur perangkat lunak menganalisis dan memecahkan masalah yang dilaporkan, menerapkan

perbaikan, dan memvalidasi resolusi melalui pengujian yang ketat. Tujuannya adalah untuk meminimalkan gangguan dan mengembalikan perangkat lunak ke keadaan fungsional.

2. ***Pemeliharaan Adaptif***: Pemeliharaan adaptif berfokus pada modifikasi sistem perangkat lunak untuk mengakomodasi perubahan di lingkungan eksternal. Ini mungkin termasuk beradaptasi dengan platform perangkat keras baru, sistem operasi, atau persyaratan peraturan. Insinyur perangkat lunak menilai dampak perubahan eksternal, melakukan modifikasi yang diperlukan, dan memastikan bahwa perangkat lunak tetap kompatibel dan fungsional dalam ekosistem yang berkembang. Pemeliharaan adaptif sangat penting untuk memastikan umur panjang dan relevansi sistem perangkat lunak.
3. ***Pemeliharaan Sempurna***: Pemeliharaan sempurna melibatkan peningkatan sistem perangkat lunak dengan memperkenalkan fitur-fitur baru, meningkatkan kinerja, atau meningkatkan pengalaman pengguna. Jenis pemeliharaan ini bertujuan untuk mengoptimalkan perangkat lunak, menjadikannya lebih efisien, intuitif, dan selaras dengan ekspektasi pengguna yang terus berkembang. Insinyur perangkat lunak mengumpulkan umpan balik pengguna, menganalisis metrik kinerja, dan menerapkan peningkatan yang menambah nilai dan meningkatkan kualitas perangkat lunak secara keseluruhan.
4. ***Pemeliharaan Preventif***: Pemeliharaan preventif adalah pendekatan proaktif yang ditujukan untuk mengidentifikasi dan menyelesaikan potensi masalah sebelum menjadi masalah. Ini melibatkan aktivitas seperti pemfaktoran ulang kode, penyetelan kinerja, dan penambalan keamanan. Dengan menangani hutang teknis, menghilangkan inefisiensi, dan memperkuat keamanan,

pemeliharaan preventif mengurangi kemungkinan masalah di masa mendatang dan meningkatkan pemeliharaan jangka panjang dan stabilitas sistem perangkat lunak.

9.3.2 Tantangan dalam Pemeliharaan Perangkat Lunak

Pemeliharaan perangkat lunak menimbulkan beberapa tantangan yang perlu ditangani secara efektif untuk memastikan keberhasilan pemeliharaan sistem perangkat lunak:

1. ***Sistem Warisan***: Mempertahankan sistem warisan menghadirkan tantangan unik, karena sering kekurangan dokumentasi, memiliki teknologi yang sudah ketinggalan zaman, dan mengakumulasi utang teknis dari waktu ke waktu. Migrasi atau modernisasi sistem lama mungkin diperlukan untuk mengatasi tantangan ini dan memungkinkan pemeliharaan yang lebih efisien.
2. ***Persyaratan yang Berubah***: Kebutuhan pengguna dan pemangku kepentingan yang berkembang membutuhkan sistem perangkat lunak untuk beradaptasi dan berkembang. Menyeimbangkan pengenalan fitur baru dengan stabilitas dan keandalan sistem yang ada merupakan tantangan yang membutuhkan perencanaan dan prioritas yang cermat.
3. ***Analisis Dampak***: Memahami dampak potensial dari perubahan dalam sistem perangkat lunak yang kompleks sangat penting untuk mencegah konsekuensi yang tidak diinginkan. Insinyur perangkat lunak harus menganalisis ketergantungan, mengevaluasi risiko, dan menyusun strategi untuk mengurangi dampak perubahan pada komponen sistem lainnya.
4. ***Kolaborasi dan Komunikasi***: Kolaborasi yang efektif antara tim pengembangan perangkat lunak, pemangku kepentingan, dan pengguna akhir sangat penting untuk keberhasilan pemeliharaan. Komunikasi yang jelas, berbagi

pengetahuan, dan dokumentasi memfasilitasi pemahaman bersama tentang sistem perangkat lunak dan persyaratan pemeliharaannya.

9.3.3 Implementasi Pemeliharaan Perangkat Lunak

Untuk mengatasi tantangan dan memastikan pemeliharaan perangkat lunak yang efisien, beberapa praktik terbaik diterapkan:

1. ***Tetapkan Strategi Pemeliharaan***: Tetapkan strategi pemeliharaan yang jelas dan komprehensif yang selaras dengan tujuan bisnis, harapan pengguna, dan standar industri. Strategi ini harus menguraikan peran, tanggung jawab, dan proses untuk mengelola aktivitas pemeliharaan.
2. ***Tekankan Dokumentasi***: Pertahankan dokumentasi terkini dan komprehensif yang menangkap desain sistem, arsitektur, dan fungsionalitas. Ini termasuk log perubahan, laporan bug, manual pengguna, dan spesifikasi teknis. Dokumentasi memfasilitasi transfer pengetahuan dan membantu dalam memahami sistem perangkat lunak.
3. ***Patuhi Standar Pengkodean dan Praktik Terbaik***: Ikuti standar pengkodean, gunakan praktik rekayasa perangkat lunak yang baik, dan manfaatkan pola desain untuk meningkatkan keterbacaan kode, pemeliharaan, dan penggunaan kembali. Praktek-praktek ini menyederhanakan pemeliharaan dan mengurangi kemungkinan memperkenalkan cacat baru.
4. ***Terapkan Integrasi dan Pengujian Berkelanjutan***: Manfaatkan integrasi berkelanjutan dan kerangka kerja pengujian otomatis untuk mendeteksi masalah di awal siklus pemeliharaan. Pengujian otomatis memastikan bahwa perubahan tidak menimbulkan regresi dan mempertahankan integritas sistem perangkat lunak.
5. ***Pantau dan Analisis Kinerja***: Terapkan mekanisme pemantauan untuk melacak kinerja perangkat lunak,

mengidentifikasi kemacetan, dan secara proaktif mengatasi masalah kinerja. Analisis rutin metrik kinerja memfasilitasi pengambilan keputusan yang tepat dan pengoptimalan yang ditargetkan.

6. ***Prioritaskan Keamanan***: Pertahankan postur keamanan yang kuat dengan terus memperbarui kerentanan keamanan, menerapkan tambalan, dan melakukan audit keamanan secara teratur. Terapkan praktik terbaik keamanan untuk melindungi dari potensi ancaman dan menjaga integritas data dan privasi pengguna.

Mempertahankan Kesehatan dan Kinerja Perangkat Lunak adalah dengan memperhatikan prioritas pentingnya pemeliharaan perangkat lunak yang merupakan aspek fundamental dari pengembangan perangkat lunak, memastikan umur panjang, kehandalan, dan kinerja sistem perangkat lunak (Priyambadha & Rochimah, 2014). Dengan menggunakan jenis pemeliharaan yang sesuai, mengatasi tantangan, dan mengikuti praktik terbaik, perekayasa perangkat lunak dapat secara efektif mempertahankan dan mengembangkan sistem perangkat lunak, memaksimalkan nilainya, dan memenuhi kebutuhan pengguna dan pemangku kepentingan yang selalu berubah.

9.4 Refactoring Perangkat Lunak

9.4.1 Meningkatkan Kualitas dan Pemeliharaan Kode

Refactoring perangkat lunak adalah proses restrukturisasi kode yang ada secara disiplin dan sistematis tanpa mengubah perilaku eksternalnya (Coelho *et al.,* 2020). Ini berfokus pada peningkatan struktur internal, desain, dan organisasi sistem perangkat lunak untuk meningkatkan kualitas kode, pemeliharaan, dan ekstensibilitas. Pada bagian ini, kami mempelajari seluk-beluk refactoring perangkat lunak, menjelajahi manfaatnya, teknik umum, dan praktik terbaik yang digunakan oleh insinyur perangkat lunak (Lima *et al.,* 2023).

9.4.2 Manfaat Pemfaktoran Ulang Perangkat Lunak

1. ***Peningkatan Kualitas Kode***: Refactoring bertujuan untuk meningkatkan kualitas kode dengan menghilangkan bau kode, mengurangi kerumitan, dan meningkatkan keterbacaan. Ini mengarah pada kode yang lebih bersih dan lebih mudah dipelihara yang lebih mudah dipahami, dimodifikasi, dan diperluas.
2. ***Peningkatan Pemeliharaan***: Kode yang direfaktor ulang lebih mudah dipelihara dan diperbarui. Ini mengurangi risiko memperkenalkan bug selama modifikasi dan memfasilitasi perbaikan bug lebih cepat. Kode yang terstruktur dengan baik juga memungkinkan pengenalan anggota tim baru yang lebih cepat, karena mereka dapat memahami dan bekerja dengan basis kode secara lebih efektif.
3. ***Ekstensibilitas dan Skalabilitas***: Refactoring membuat basis kode lebih dapat diperluas, memungkinkan integrasi fitur dan penyempurnaan baru dengan lebih mudah. Ini meningkatkan skalabilitas sistem perangkat lunak dengan memungkinkannya menangani peningkatan kompleksitas

dan basis kode yang lebih besar tanpa mengorbankan kinerja atau pemeliharaan.

4. ***Pencegahan Cacat***: Refactoring membantu mengidentifikasi dan mengatasi potensi cacat dan kerentanan di awal proses pengembangan. Dengan menghilangkan duplikasi, meningkatkan modularisasi, dan meningkatkan kejelasan kode, pemfaktoran ulang mengurangi kemungkinan munculnya bug baru dan membuat basis kode lebih tangguh.

9.4.3 Teknik Refactoring Umum

1. ***Metode Ekstrak***: Teknik ini melibatkan penggalian blok kode menjadi metode terpisah, meningkatkan keterbacaan kode, mempromosikan penggunaan kembali kode, dan mengurangi duplikasi.
2. ***Ganti nama***: Teknik ganti nama berfokus pada pemberian nama yang bermakna ke variabel, metode, dan kelas, meningkatkan pemahaman kode dan memfasilitasi pemeliharaan dan kolaborasi yang lebih mudah.
3. ***Hapus Duplikasi***: Duplikasi dalam kode dapat menyebabkan tantangan pemeliharaan dan ketidakkonsistenan. Menghapus duplikasi melibatkan identifikasi segmen kode berulang dan mengabstraksikannya menjadi fungsi atau modul yang dapat digunakan kembali.
4. ***Ganti Logika Bersyarat dengan Polimorfisme***: Dengan menerapkan polimorfisme, logika bersyarat dapat diganti dengan kode yang lebih terstruktur dan dapat dipelihara, meningkatkan fleksibilitas dan ekstensibilitas.
5. ***Sederhanakan Ekspresi Kompleks***: Teknik ini melibatkan penguraian ekspresi kompleks menjadi komponen yang lebih sederhana dan lebih mudah dikelola, meningkatkan kejelasan kode dan membuatnya lebih mudah untuk dipahami dan dipelihara.

9.4.4 Praktik Terbaik Pemfaktoran Ulang Perangkat Lunak

Praktik Terbaik dalam Pemfaktoran Ulang Perangkat Lunak, Untuk memastikan pemfaktoran ulang perangkat lunak yang efektif dan efisien, perekayasa perangkat lunak mengikuti beberapa praktik terbaik:

1. ***Pahami Kode yang Ada***: Pahami basis kode secara menyeluruh sebelum memulai pemfaktoran ulang. Menganalisis struktur, dependensi, dan area peningkatannya. Pemahaman ini memberikan dasar yang kuat untuk membuat keputusan refactoring yang terinformasi.
2. ***Refactoring Berbasis Tes***: Gunakan pendekatan pengembangan berbasis tes selama refactoring. Kembangkan rangkaian pengujian yang komprehensif untuk memvalidasi kebenaran perubahan kode dan memastikan bahwa pemfaktoran ulang tidak menimbulkan regresi.
3. ***Refactor dalam Langkah Kecil***: Memecah tugas refactoring menjadi langkah-langkah kecil yang dapat dikelola. Hal ini meminimalkan risiko munculnya kesalahan dan memudahkan untuk memvalidasi perubahan di setiap tahap. Langkah-langkah kecil juga memungkinkan pelacakan kemajuan dan pengembalian yang lebih mudah jika diperlukan.
4. ***Integrasi Berkelanjutan dan Kontrol Versi***: Memanfaatkan integrasi berkelanjutan dan sistem kontrol versi untuk melacak perubahan, berkolaborasi dengan anggota tim, dan memutar kembali jika perlu. Ini memastikan bahwa aktivitas refactoring terintegrasi dengan lancar ke dalam alur kerja pengembangan.
5. ***Keputusan Refactoring Dokumen***: Pertahankan dokumentasi yang menangkap alasan di balik keputusan refactoring, teknik yang diterapkan, dan hasil yang diharapkan. Dokumentasi ini berfungsi sebagai referensi

untuk pemeliharaan di masa mendatang dan memastikan bahwa upaya pemfaktoran ulang didokumentasikan dan dikomunikasikan dengan baik.

Pemfaktoran ulang perangkat lunak adalah praktik vital dalam rekayasa perangkat lunak, yang memungkinkan peningkatan kualitas kode, pemeliharaan, dan ekstensibilitas (den Besten *et al.,* 2006). Dengan menerapkan teknik pemfaktoran ulang yang sesuai, mengikuti praktik terbaik, dan memastikan pengujian komprehensif, perekayasa perangkat lunak dapat mengubah kode yang kompleks dan sulit dipelihara menjadi fondasi yang kuat dan mudah beradaptasi, membuka jalan bagi pengembangan dan evolusi sistem perangkat lunak yang efisien.

9.5 Desain dan Arsitektur Evolusioner

9.5.1 Memelihara Sistem Perangkat Lunak untuk Adaptasi dan Pertumbuhan Berkelanjutan

Desain dan arsitektur evolusioner adalah pendekatan dinamis yang merangkul perubahan sebagai aspek inheren dari pengembangan perangkat lunak (Wu *et al*., 2016). Ini mengakui bahwa sistem perangkat lunak perlu berevolusi dan beradaptasi dari waktu ke waktu untuk memenuhi kebutuhan pengguna yang terus berkembang, pemangku kepentingan, dan lanskap teknologi yang selalu berubah. Pendekatan ini berfokus pada pembuatan desain dan arsitektur yang fleksibel, dapat diperluas, dan dapat diskalakan yang dapat mengakomodasi kebutuhan masa depan dan memfasilitasi evolusi yang berkelanjutan. Pada bagian ini, kami menjelajahi seluk-beluk desain dan arsitektur evolusioner, prinsip, manfaat, dan praktik terbaiknya yang digunakan oleh insinyur perangkat lunak (Coelho et al., 2020).

9.5.2 Prinsip Desain dan Arsitektur Evolusioner

Modularitas dan Enkapsulasi: Desain evolusioner menekankan modularitas dan enkapsulasi, mempromosikan pemisahan perhatian dan memungkinkan pengembangan independen dan evolusi komponen sistem. Modul yang terdefinisi dengan baik dengan tanggung jawab yang jelas memfasilitasi fleksibilitas, penggunaan kembali, dan pemeliharaan.

1. ***Kopling Longgar dan Kohesi Tinggi***: Desain evolusioner mendorong kopling longgar antara komponen sistem, mengurangi ketergantungan, dan memungkinkan evolusi independen. Kohesi tinggi memastikan bahwa fungsionalitas terkait dikelompokkan bersama, membuat sistem lebih koheren dan lebih mudah dipahami dan dimodifikasi.
2. ***Abstraksi dan Enkapsulasi Perubahan***: Desain evolusioner memanfaatkan abstraksi dan enkapsulasi untuk mengisolasi dan mengelola perubahan dalam modul atau komponen tertentu. Dengan mengenkapsulasi perubahan, dampak pada keseluruhan sistem diminimalkan, mengurangi risiko pengenalan cacat dan menyederhanakan modifikasi di masa mendatang.
3. ***Desain untuk Perubahan***: Desain dan arsitektur sistem evolusi didorong oleh antisipasi perubahan di masa depan. Fleksibilitas, ekstensibilitas, dan kemampuan beradaptasi diprioritaskan, memungkinkan sistem merangkul persyaratan, teknologi, dan kebutuhan bisnis baru tanpa gangguan yang signifikan.

9.5.3 Manfaat Desain dan Arsitektur Evolusioner

1. ***Ketangkasan dan Kemampuan Beradaptasi***: Dengan mengadopsi pendekatan evolusioner, sistem perangkat lunak menjadi lebih gesit dan mudah beradaptasi. Mereka dapat dengan cepat menanggapi perubahan kebutuhan

pengguna, permintaan pasar, dan kemajuan teknologi, memperoleh keunggulan kompetitif dalam lingkungan yang berkembang pesat.

2. ***Kemudahan Pemeliharaan dan Evolusi***: Desain evolusioner mempromosikan kode yang lebih mudah dipahami, dimodifikasi, dan diperluas. Dengan mengikuti prinsip-prinsip seperti modularitas dan kopling longgar, upaya pemeliharaan dan evolusi disederhanakan, mengurangi risiko regresi dan meminimalkan waktu dan upaya yang diperlukan untuk penyempurnaan di masa mendatang.
3. ***Skalabilitas dan Performa***: Arsitektur evolusioner dapat diskalakan dengan anggun saat sistem tumbuh dalam kompleksitas dan penggunaan. Ini mendukung distribusi beban kerja yang efisien, integrasi fungsionalitas baru, dan optimalisasi kinerja, memastikan bahwa sistem tetap responsif dan berperforma tinggi saat tuntutan meningkat.
4. ***Mitigasi Risiko***: Desain evolusioner mengurangi risiko yang terkait dengan pengembangan perangkat lunak. Dengan menyempurnakan dan memvalidasi sistem secara terus-menerus melalui perubahan bertahap, potensi masalah dan cacat diidentifikasi dan ditangani lebih awal, meminimalkan kemungkinan kegagalan yang merugikan dan merugikan.

9.5.4 Praktik Terbaik Desain dan Arsitektur Evolusioner

Untuk memastikan keberhasilan penerapan desain dan arsitektur evolusioner, insinyur perangkat lunak mengikuti beberapa praktik terbaik:

1. ***Refactoring Berkelanjutan***: Secara teratur refactor basis kode untuk meningkatkan desainnya, menghilangkan bau kode, dan meningkatkan pemeliharaan. Refactoring harus menjadi kegiatan yang berkelanjutan, memungkinkan

sistem untuk terus berkembang dan beradaptasi dengan perubahan kebutuhan dan wawasan.

2. ***Integrasi dan Pengiriman Berkelanjutan***: Manfaatkan praktik integrasi dan pengiriman berkelanjutan untuk memfasilitasi integrasi perubahan yang sering dan otomatis. Ini memastikan bahwa komponen yang berkembang diuji secara menyeluruh, divalidasi, dan diintegrasikan dengan mulus ke dalam sistem yang berkembang.
3. ***Penggunaan Pola Desain***: Terapkan pola desain yang telah terbukti untuk mengatasi tantangan desain umum dan mempromosikan praktik arsitektur yang baik. Pola desain memberikan solusi yang dapat digunakan kembali untuk masalah berulang dan memfasilitasi evolusi sistem perangkat lunak dengan cara yang terstruktur dan dapat dipelihara.
4. ***Rangkul Desain yang Muncul***: Biarkan desain dan arsitektur sistem muncul secara bertahap saat persyaratan baru terungkap dan wawasan diperoleh. Pendekatan adaptif ini memungkinkan sistem untuk berkembang secara organik, memastikan bahwa arsitektur selaras dengan kebutuhan aktual dan bukan hanya asumsi awal.
5. ***Pengembangan Berbasis Umpan Balik***: Cari umpan balik dari pengguna, pemangku kepentingan, dan anggota tim selama proses pengembangan. Secara aktif mengintegrasikan umpan balik ke dalam proses evolusi untuk menyempurnakan desain, mengatasi masalah, dan memastikan bahwa sistem yang berkembang memenuhi kebutuhan dan harapan penggunanya.

Konsep desain dan arsitektur evolusioner adalah pola pikir dan pendekatan yang mendorong adaptasi, pertumbuhan, dan peningkatan berkelanjutan dari sistem perangkat lunak (Constantino *et al.,* 2023). Dengan merangkul perubahan,

mengikuti prinsip-prinsip desain, dan mengikuti praktik terbaik, insinyur perangkat lunak dapat menciptakan sistem yang fleksibel, dapat dipelihara, dan tangguh yang dapat berkembang dengan anggun, memungkinkan organisasi untuk memenuhi tuntutan lanskap perangkat lunak yang selalu berubah.

9.6 Integrasi dan Penerapan Berkelanjutan

9.6.1 Merampingkan Pengembangan dan Pengiriman Perangkat Lunak

Continuous Integration and Deployment (CI/CD) adalah praktik pengembangan perangkat lunak yang bertujuan untuk mengotomatisasi dan merampingkan proses pengintegrasian perubahan kode, membuat artefak perangkat lunak, dan menerapkannya ke lingkungan produksi (Mayefis *et al.,* 2023b). Ini mempromosikan kolaborasi, efisiensi, dan pengiriman pembaruan perangkat lunak yang cepat, memungkinkan tim untuk memberikan perangkat lunak berkualitas tinggi dengan intervensi manual minimal. Di bagian ini, kami mempelajari seluk-beluk integrasi dan penerapan berkelanjutan, prinsip, manfaat, dan praktik terbaiknya yang digunakan oleh tim rekayasa perangkat lunak (Lima *et al*., 2023).

9.6.2 Prinsip Integrasi dan Penerapan Berkelanjutan

1. ***Integrasi Berkelanjutan***: Integrasi Berkelanjutan berfokus pada pengintegrasian perubahan kode yang sering dan secara otomatis ke dalam repositori bersama. Praktik ini mendorong developer untuk menggabungkan perubahan mereka ke basis kode pusat beberapa kali sehari, memfasilitasi deteksi dini masalah integrasi dan mendorong kolaborasi.
2. ***Pembuatan dan Pengujian Otomatis***: CI/CD menekankan pada otomatisasi proses pembuatan dan pelaksanaan

pengujian otomatis komprehensif untuk memvalidasi kebenaran dan stabilitas perangkat lunak. Ini memastikan bahwa perubahan kode tidak menimbulkan regresi dan mempertahankan integritas sistem selama siklus pengembangan.

3. ***Penerapan Berkesinambungan***: Penerapan Berkesinambungan memperluas konsep integrasi berkelanjutan dengan menyebarkan artefak perangkat lunak secara otomatis ke produksi atau lingkungan seperti produksi. Ini memungkinkan rilis yang sering, mengurangi waktu ke pasar, dan memungkinkan umpan balik yang cepat dari pengguna akhir.
4. ***Infrastruktur sebagai Kode***: CI/CD memanfaatkan praktik infrastruktur sebagai kode (IaC) untuk mengotomatiskan penyediaan dan konfigurasi sumber daya infrastruktur. Dengan memperlakukan infrastruktur sebagai kode, lingkungan penerapan dapat dengan mudah dibuat ulang, memastikan konsistensi, dan mengurangi kemungkinan masalah terkait penerapan.

9.6.3 Manfaat Integrasi dan Penerapan Berkelanjutan

1. ***Deteksi Dini Masalah***: Dengan sering mengintegrasikan perubahan kode dan menjalankan pengujian otomatis, praktik CI/CD membantu mengidentifikasi masalah integrasi, bug, dan konflik di awal proses pengembangan. Hal ini memungkinkan resolusi yang cepat dan mengurangi akumulasi utang teknis.
2. ***Putaran Umpan Balik Lebih Cepat***: Pembuatan, pengujian, dan penerapan otomatis memfasilitasi putaran umpan balik yang lebih cepat, memungkinkan pengembang untuk menerima umpan balik langsung tentang dampak perubahan mereka. Ini mendorong pengulangan yang cepat, kolaborasi, dan penyelesaian masalah yang lebih cepat.

3. ***Pengurangan Upaya Manual:*** CI/CD mengurangi kebutuhan untuk tugas manual, seperti pembuatan manual, pengujian, dan prosedur penerapan. Otomasi merampingkan proses ini, membebaskan waktu yang berharga bagi pengembang untuk fokus pada pengkodean, inovasi, dan tugas bernilai lebih tinggi.
4. ***Peningkatan Kualitas dan Stabilitas***: Integrasi berkelanjutan dan pengujian otomatis memastikan bahwa perubahan kode menjalani validasi yang ketat, mengurangi kemungkinan memperkenalkan cacat ke dalam perangkat lunak. Ini menghasilkan perangkat lunak berkualitas lebih tinggi dan lebih stabil yang memenuhi ekspektasi pengguna dan meminimalkan waktu henti.

9.6.4 Implementasi Integrasi dan Penerapan Berkelanjutan

Praktik Terbaik dalam Integrasi dan Penerapan Berkelanjutan, Untuk memaksimalkan manfaat CI/CD, tim rekayasa perangkat lunak mengikuti beberapa praktik terbaik:

1. ***Kontrol Versi dan Strategi Percabangan***: Memanfaatkan sistem kontrol versi, seperti Git, untuk mengelola perubahan kode dan menerapkan strategi percabangan terstruktur. Hal ini memungkinkan pengembangan paralel, isolasi fitur, dan integrasi perubahan yang mulus.
2. ***Pembuatan dan Pengujian Otomatis***: Siapkan sistem pembangunan tangguh yang secara otomatis mengompilasi perubahan kode, menjalankan pengujian unit, dan melakukan pengujian integrasi. Pengujian otomatis memastikan stabilitas dan kebenaran perangkat lunak dan membantu menangkap masalah lebih awal.
3. ***Server Integrasi Berkelanjutan***: Memanfaatkan server atau platform integrasi berkelanjutan khusus yang memantau repositori kontrol versi, memicu pembuatan, dan menjalankan pengujian secara otomatis setelah

perubahan kode. Hal ini mendorong umpan balik berkelanjutan dan deteksi dini masalah integrasi.

4. ***Continuous Deployment Pipeline***: Terapkan pipeline penerapan yang terdefinisi dengan baik yang mengotomatiskan proses pengemasan dan penerapan artefak perangkat lunak ke lingkungan yang berbeda. Pipeline harus mencakup langkah-langkah untuk pengujian, staging, dan penerapan produksi, dengan pemeriksaan dan keseimbangan yang sesuai.
5. ***Infrastruktur sebagai Kode (IaC)***: Menerapkan infrastruktur sebagai praktik kode, menggunakan alat seperti Ansible, Terraform, atau CloudFormation, untuk mengotomatiskan penyediaan dan konfigurasi lingkungan penerapan. Hal ini memastikan konsistensi, reproduktifitas, dan mengurangi upaya manual dalam mengelola infrastruktur.
6. ***Pemantauan dan Umpan Balik Berkelanjutan***: Terapkan mekanisme pemantauan untuk mengumpulkan dan menganalisis metrik dari lingkungan produksi. Pantau kinerja aplikasi, kesalahan, dan umpan balik pengguna untuk mengidentifikasi area peningkatan dan menginformasikan iterasi pengembangan di masa mendatang.

Dalam praktik integrasi dan penerapan berkelanjutan merupakan hal mendasar dalam rekayasa perangkat lunak modern, yang memungkinkan tim mengirimkan perangkat lunak lebih cepat, dengan kualitas lebih tinggi, dan upaya manual yang lebih sedikit. Dengan merangkul otomatisasi, umpan balik awal, dan pengembangan kolaboratif, tim rekayasa perangkat lunak dapat merampingkan proses pengembangan mereka, meningkatkan produktivitas, dan meningkatkan siklus hidup pengiriman perangkat lunak secara keseluruhan.

9.7 Kontrol Versi dan Manajemen Perubahan

9.7.1 Mengaktifkan Kolaborasi yang Efisien dan Evolusi Perangkat Lunak

Kontrol versi dan manajemen perubahan adalah praktik mendasar dalam pengembangan perangkat lunak yang memfasilitasi kolaborasi yang efisien, melacak perubahan kode, dan mengelola evolusi sistem perangkat lunak. Mereka menyediakan mekanisme untuk melacak, menggabungkan, dan mengembalikan perubahan kode, memastikan integritas basis kode dan memungkinkan tim bekerja secara paralel dengan lancar. Di bagian ini, kami mempelajari seluk-beluk kontrol versi dan manajemen perubahan, menjelajahi prinsip, manfaat, dan praktik terbaiknya yang digunakan oleh tim rekayasa perangkat lunak (Priyambadha & Rochimah, 2014).

9.7.2 Prinsip Kontrol Versi dan Manajemen Perubahan

1. ***Pembuatan versi***: Sistem kontrol versi (VCS) memungkinkan pengembang untuk melacak perubahan pada file kode dan aset proyek lainnya dari waktu ke waktu. Setiap perubahan dicatat, membuat riwayat revisi dan memungkinkan pengambilan versi sebelumnya dengan mudah bila diperlukan.
2. ***Pengembangan Serentak***: Sistem kontrol versi mendukung pengembangan bersamaan dengan memungkinkan beberapa pengembang untuk bekerja di berbagai bagian basis kode secara bersamaan. Mereka menyediakan mekanisme untuk menggabungkan perubahan yang dilakukan oleh pengembang yang berbeda, memastikan bahwa basis kode akhir mencerminkan kontribusi semua anggota tim.
3. ***Ubah Pelacakan dan Audit***: Manajemen perubahan melibatkan pelacakan dan mendokumentasikan setiap perubahan kode, termasuk informasi tentang penulis,

stempel waktu, dan tujuan. Ini memungkinkan akuntabilitas, ketertelusuran, dan audit modifikasi kode.
4. ***Kembalikan dan Kembalikan***: Sistem kontrol versi menyediakan kemampuan untuk mengembalikan atau mengembalikan perubahan, memungkinkan tim membatalkan modifikasi yang menimbulkan masalah atau konflik. Ini membantu menjaga stabilitas dan keandalan sistem perangkat lunak.

9.7.3 Manfaat Kontrol Versi dan Manajemen Perubahan

1. ***Kolaborasi dan Kerja Tim***: Kontrol versi memungkinkan kolaborasi di antara anggota tim dengan menyediakan pusat penyimpanan untuk berbagi dan mengintegrasikan perubahan kode. Ini memungkinkan pengembang untuk mengerjakan basis kode yang sama secara bersamaan dan memfasilitasi komunikasi, koordinasi, dan kerja tim yang efektif.
2. ***Riwayat dan Pemulihan Kode***: Sistem kontrol versi menyimpan riwayat terperinci perubahan kode, menyediakan jaring pengaman jika terjadi kesalahan, bug, atau konsekuensi yang tidak diinginkan. Pengembang dapat dengan mudah kembali ke versi sebelumnya, memulihkan pekerjaan yang hilang, dan menganalisis evolusi basis kode dari waktu ke waktu.
3. ***Resolusi Konflik***: Pengembangan bersamaan dapat menyebabkan konflik ketika banyak pengembang memodifikasi kode yang sama secara bersamaan. Sistem kontrol versi menyediakan alat untuk mendeteksi dan menyelesaikan konflik, memastikan bahwa perubahan kode terintegrasi dengan lancar dan meminimalkan gangguan.
4. ***Integrasi dan Penerapan Berkelanjutan***: Sistem kontrol versi memainkan peran penting dalam memungkinkan praktik integrasi dan penerapan berkelanjutan. Mereka

memfasilitasi otomatisasi proses build, memicu pengujian dan penerapan pipeline, dan memastikan bahwa artefak perangkat lunak dibuat dari versi basis kode yang benar dan konsisten.

9.7.4 Implementasi - Kontrol Versi - Manajemen Perubahan

Praktik Terbaik dalam Kontrol Versi dan Manajemen Perubahan, Untuk mengoptimalkan kontrol versi dan mengubah proses manajemen, tim rekayasa perangkat lunak mengikuti beberapa praktik terbaik:

1. ***Gunakan Sistem Kontrol Versi***: Gunakan sistem kontrol versi terdistribusi, seperti Git atau Mercurial, untuk mengelola perubahan kode secara efektif. Siapkan repositori pusat sebagai sumber kebenaran dan tetapkan pedoman yang jelas untuk strategi percabangan dan penggabungan.
2. ***Lakukan Lebih Awal dan Sering***: Dorong pengembang untuk melakukan perubahan lebih awal dan sering. Komitmen kecil dan atomik memudahkan untuk melacak perubahan, memahami tujuannya, dan mengembalikannya jika perlu. Ini juga memfasilitasi pengembangan dan kolaborasi paralel.
3. ***Strategi Percabangan dan Penggabungan***: Tentukan strategi percabangan yang sesuai dengan alur kerja pengembangan dan ukuran tim. Strategi umum meliputi percabangan fitur, percabangan rilis, dan pengembangan berbasis trunk. Komunikasikan dengan jelas dan tegakkan strategi untuk meminimalkan konflik dan memastikan proses integrasi yang lancar.
4. ***Tinjauan Kode dan Jaminan Kualitas***: Manfaatkan praktik tinjauan kode untuk memastikan kualitas dan konsistensi perubahan kode. Dorong peer review, di mana anggota tim secara menyeluruh meninjau kode masing-

masing untuk kebenaran, kepatuhan terhadap standar pengkodean, dan praktik terbaik sebelum bergabung ke dalam basis kode utama.

5. ***Pengujian Otomatis dan Integrasi Berkelanjutan***: Integrasikan pengujian otomatis ke dalam kontrol versi dan proses manajemen perubahan. Siapkan sistem integrasi berkelanjutan yang secara otomatis membangun, menguji, dan memvalidasi perubahan kode saat melakukan, memberikan umpan balik awal tentang kebenaran dan stabilitas perangkat lunak.
6. ***Dokumentasi dan Changelogs***: Mendokumentasikan perubahan kode, perbaikan bug, dan fitur baru di changelogs atau catatan rilis. Ini membantu mempertahankan catatan modifikasi yang komprehensif dan memberikan kejelasan kepada pemangku kepentingan dan anggota tim tentang evolusi perangkat lunak.

Kesimpulannya, kontrol versi dan manajemen perubahan adalah komponen penting dari pengembangan perangkat lunak, memungkinkan kolaborasi yang efisien, melacak perubahan kode, dan memfasilitasi evolusi sistem perangkat lunak. Dengan mengikuti praktik terbaik, tim dapat memastikan integrasi perubahan kode yang lancar, kolaborasi yang efektif, dan riwayat modifikasi yang andal, yang pada akhirnya mengarah pada peningkatan produktivitas, kualitas kode, dan evolusi perangkat lunak.

9.8 Pengujian dan Jaminan Kualitas dalam Evolusi Perangkat Lunak

9.8.1 Memastikan Keandalan dan Peningkatan Berkesinambungan

Pengujian dan jaminan kualitas merupakan aspek integral dari proses evolusi perangkat lunak, memainkan peran

penting dalam memastikan keandalan, fungsionalitas, dan kinerja sistem perangkat lunak (Acuña *et al*., 2012). Mereka melibatkan pendekatan yang sistematis dan komprehensif untuk memverifikasi dan memvalidasi perangkat lunak, mendeteksi cacat, dan memberikan umpan balik untuk perbaikan berkelanjutan. Pada bagian ini, kami menyelidiki seluk-beluk pengujian dan jaminan kualitas dalam evolusi perangkat lunak, mengeksplorasi prinsip, manfaat, dan praktik terbaik yang digunakan oleh tim rekayasa perangkat lunak (Jiménez-Ramírez *et al.,* 2023).

9.8.2 Prinsip Pengujian dan Jaminan Kualitas

1. ***Cakupan Tes***: Pengujian dan jaminan kualitas berusaha untuk mencapai cakupan pengujian yang komprehensif dengan menangani berbagai dimensi kualitas perangkat lunak. Ini termasuk pengujian fungsional untuk memastikan perangkat lunak berperilaku sebagaimana dimaksud, pengujian kinerja untuk menilai daya tanggap dan skalabilitasnya, pengujian keamanan untuk mengidentifikasi kerentanan, dan pengujian kegunaan untuk mengevaluasi pengalaman pengguna.
2. ***Pengujian Awal dan Berkelanjutan***: Pengujian bukanlah peristiwa satu kali; itu harus dilakukan terus menerus selama proses evolusi perangkat lunak. Dengan melakukan pengujian lebih awal dan lebih sering, masalah dapat diidentifikasi dan diatasi pada tahap paling awal, sehingga mengurangi biaya dan dampak kerusakan.
3. ***Otomasi***: Otomasi memainkan peran penting dalam pengujian dan jaminan kualitas. Alat dan kerangka kerja pengujian otomatis memungkinkan pelaksanaan pengujian berulang, memberikan umpan balik yang lebih cepat, dan memfasilitasi pengujian regresi, memastikan fungsionalitas yang ada tetap utuh saat perangkat lunak berkembang.

4. ***Kolaborasi dan Putaran Umpan Balik***: Pengujian dan jaminan kualitas melibatkan kolaborasi antara pengembang, penguji, dan pemangku kepentingan. Umpan balik dari aktivitas pengujian membantu mengidentifikasi area untuk peningkatan, menginformasikan pengambilan keputusan, dan memandu evolusi perangkat lunak.

9.8.3 Manfaat Pengujian dan Jaminan Kualitas

1. ***Peningkatan Kualitas Perangkat Lunak***: Praktik pengujian dan jaminan kualitas membantu mengidentifikasi dan memperbaiki cacat, memastikan bahwa perangkat lunak memenuhi persyaratan fungsional dan non-fungsional. Ini mengarah pada peningkatan kualitas perangkat lunak, mengurangi kemungkinan kegagalan, ketidakpuasan pelanggan, dan pemeliharaan yang mahal.
2. ***Keandalan dan Stabilitas yang Ditingkatkan***: Pengujian menyeluruh membantu mengidentifikasi dan memperbaiki bug, memastikan keandalan dan stabilitas perangkat lunak. Dengan mendeteksi dan menangani masalah lebih awal, perekayasa perangkat lunak dapat memitigasi risiko dan menghadirkan produk tangguh yang bekerja seperti yang diharapkan.
3. ***Peningkatan Kepuasan Pengguna***: Pengujian yang efektif dan jaminan kualitas berkontribusi pada pengalaman pengguna yang lebih baik. Dengan memvalidasi fungsionalitas, kinerja, dan kegunaan perangkat lunak, tim dapat menghasilkan produk yang memenuhi harapan pengguna, meningkatkan kepuasan, dan membangun kepercayaan dengan basis pengguna.
4. ***Penghematan Biaya dan Waktu***: Mengidentifikasi dan mengatasi kerusakan di awal proses evolusi perangkat lunak secara signifikan mengurangi biaya dan upaya yang diperlukan untuk pengerjaan ulang, perbaikan bug, dan

pemeliharaan. Praktik pengujian dan penjaminan mutu yang efektif menghemat waktu dan sumber daya, memungkinkan tim untuk berfokus pada penyampaian fitur dan peningkatan baru.

9.8.4 Praktik Terbaik Pengujian dan Jaminan Kualitas

Untuk memaksimalkan manfaat pengujian dan jaminan kualitas, tim rekayasa perangkat lunak mengikuti beberapa praktik terbaik:

1. ***Strategi dan Perencanaan Pengujian***: Kembangkan strategi dan rencana pengujian komprehensif yang sejalan dengan tujuan, persyaratan, dan batasan proyek. Tetapkan tujuan pengujian, cakupan pengujian, dan teknik pengujian yang sesuai dengan karakteristik perangkat lunak dan tujuan penggunaan.
2. ***Otomasi Pengujian***: Manfaatkan kerangka kerja dan alat otomatisasi pengujian untuk mengotomatiskan aktivitas pengujian yang berulang dan memakan waktu. Ini termasuk pengujian unit, pengujian integrasi, pengujian regresi, dan pengujian kinerja. Pengujian otomatis memastikan eksekusi yang konsisten dan efisien, memungkinkan umpan balik yang lebih cepat dan meminimalkan kesalahan manusia.
3. ***Integrasi dan Pengujian Berkelanjutan***: Mengintegrasikan pengujian ke dalam proses integrasi dan penerapan berkelanjutan. Otomatiskan eksekusi pengujian sebagai bagian dari pipeline build untuk memastikan bahwa perubahan kode diuji secara menyeluruh sebelum diterapkan. Pengujian berkelanjutan memastikan bahwa perangkat lunak tetap berfungsi dan stabil seiring perkembangannya.
4. ***Pengujian Eksplorasi***: Lengkapi pengujian otomatis dengan pengujian eksplorasi untuk mengungkap masalah tak terduga dan memvalidasi pengalaman pengguna.

Pengujian eksplorasi melibatkan eksplorasi perangkat lunak secara manual, memungkinkan penguji untuk mengidentifikasi kasus tepi, masalah kegunaan, dan potensi cacat yang mungkin terlewatkan oleh pengujian otomatis.

5. ***Pengujian Kinerja***: Lakukan pengujian kinerja untuk menilai skalabilitas, daya tanggap, dan pemanfaatan sumber daya perangkat lunak. Ini memastikan bahwa sistem dapat menangani beban kerja yang diharapkan dan bekerja secara optimal, bahkan saat perangkat lunak berkembang.
6. ***Pelacakan dan Pelaporan Bug***: Menerapkan sistem pelacakan dan pelaporan bug yang kuat untuk mendokumentasikan dan memprioritaskan masalah yang teridentifikasi. Ini memfasilitasi komunikasi yang efektif, kolaborasi, dan resolusi cacat, memastikan bahwa mereka ditangani tepat waktu.
7. ***Peningkatan Berkesinambungan***: Terus mengevaluasi dan menyempurnakan proses pengujian dan penjaminan mutu berdasarkan umpan balik, metrik, dan pelajaran yang dipetik. Menumbuhkan budaya pembelajaran dan peningkatan, mendorong anggota tim untuk berbagi pengetahuan, mengadopsi praktik terbaik, dan memasukkan umpan balik ke dalam proses evolusi perangkat lunak.

Pengujian dan jaminan kualitas adalah komponen vital dari proses evolusi perangkat lunak, memastikan keandalan, fungsionalitas, dan kinerja sistem perangkat lunak. Dengan mengikuti praktik terbaik, tim dapat meningkatkan kualitas perangkat lunak, meningkatkan kepuasan pengguna, dan mendorong peningkatan berkelanjutan, yang pada akhirnya menghasilkan produk berkualitas tinggi yang memenuhi harapan pengguna dan beradaptasi dengan perubahan kebutuhan (Finandhita, 2018).

DAFTAR PUSTAKA

Acuña, S. T., Castro, J. W., Dieste, O., & Juristo, N. 2012. A systematic mapping study on the open source software development process. *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, 42–46.

Adhy, M., Priyambadha, B., & Pradana, F. 2019. Kuantifikasi Pengaruh Understandability dan Maintainability pada Evolusi Perangkat Lunak. *Jurnal Teknologi Informasi Dan Ilmu Komputer (JTIIK)*, *6*(3).

Akhmad Al Aidhi, M. A. K. H. A. Y. R. 2023. Peningkatan Daya Saing Ekonomi melalui peranan Inovasi. *Jurnal Multidisiplin West Science*. https://wnj.westscience-press.com/index.php/jmws/article/view/229/160

Chakroborti, D., Nath, S. S., Schneider, K. A., & Roy, C. K. 2023. Release conventions of open-source software: An exploratory study. *Journal of Software: Evolution and Process*, *35*(1), e2499.

Chusumastuti, D., Zulfikri, A., & Rukmana, A. Y. 2023. Pengaruh Digital Marketing dan Kompetensi Wirausaha Terhadap Kinerja Pemasaran (Studi ada UMKM di Jawa Barat). *Jurnal Bisnis Dan Manajemen West Science*, *2*(02), 22–32.

Coelho, J., Valente, M. T., Milen, L., & Silva, L. L. 2020. Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects. *Information and Software Technology*, *122*, 106274.

Constantino, K., Souza, M., Zhou, S., Figueiredo, E., & Kästner, C. 2023. Perceptions of open-source software developers on collaborations: An interview and survey study. *Journal of Software: Evolution and Process*, *35*(5), e2393.

Den Besten, M., Dalle, J.-M., & Galia, F. 2006. *Collaborative maintenance in large open-source projects*. Springer.

Finandhita, A. 2018. *Pemeliharaan Perangkat Lunak*.

Gunawan, H. 2021. PENGARUH IMPLEMENTASI DIGITAL MARKETING SEBAGAI ALTERNATIF STRATEGI PEMASARAN USAHA YANG EFEKTIF. *Jurnal Abdimastek (Pengabdian Masyarakat Berbasis Teknologi)*, *2*(2), 27–38.

Harto, B., Saymsu, Y. L., Rukmana, A. Y., Komalasari, R., & Dwijayanti, A. 2022. Bibliometric Analysis of Transforming Leadership Education with Artificial Intelligence. In *1st Virtual Workshop on Writing Scientific Article for International Publication Indexed SCOPUS* (pp. 385–390). Sciendo. https://doi.org/10.2478/9788366675827-067

Jiménez-Ramírez, A., Chacón-Montero, J., Wojdynsky, T., & Gonzalez Enriquez, J. 2023. Automated testing in robotic process automation projects. *Journal of Software: Evolution and Process*, *35*(3), e2259.

Lima, P., Melegati, J., Gomes, E., Pereira, N. S., Guerra, E., & Meirelles, P. 2023. Cadv: A software visualization approach for code annotations distribution. *Information and Software Technology*, *154*, 107089.

Mayefis, R., Yusnanto, T., Heryana, N., Aulia, A. P., Permana, A. A., Aisa, S., Pasaribu, J. S., Wahyuddin, S., & Sihombing, F. A. H. 2023a. *Rekayasa Perangkat Lunak*. Global Eksekutif Teknologi.

Mayefis, R., Yusnanto, T., Heryana, N., Aulia, A. P., Permana, A. A., Aisa, S., Pasaribu, J. S., Wahyuddin, S., & Sihombing, F. A. H. 2023b. *Rekayasa Perangkat Lunak*. Global Eksekutif Teknologi.

Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., & Jazayeri, M. 2005. Challenges in software evolution. *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, 13–22.

Priyambadha, B., & Rochimah, S. 2014. Kuantifikasi Pengaruh Kloning Dan Kompleksitas Kode Terhadap Cacat Pada Evolusi Perangkat Lunak. *JUTI J. Ilm. Teknol. Inf*, *11*(2), 19.

Razali, G., Akbarina, F., Arubusman, D. A., Rukmana, A. Y., & Yusuf, M. 2023. Loyalty and the Effects of Trust and Switching Barriers. *Jurnal Mirai Management*, *8*(1), 237–248.

Romindo, R. M., Yusnanto, T., Heryana, N., Jamaludin, A. P. A., Permana, A. A., Aisa, S., Pasaribu, J. S., Wahyuddin, S., & Sihombing, F. A. H. (n.d.). *REKAYASA PERANGKAT LUNAK*.

Rukmana, A. Y. 2023. BAB 3 TEKNOLOGI DIGITAL. *Digital Marketing Dan E-Commerce*, 27.

Rukmana, A. Y., Zebua, R. S. Y., Aryanto, D., Nur'Aini, I., Ardiansyah, W., Adhicandra, I., & Setiawan, Z. 2023. *DUNIA MULTIMEDIA: Pengenalan dan Penerapannya*. PT. Sonpedia Publishing Indonesia.

Setiawan, N. Y. (n.d.). Pengaruh Pola Hubungan Pengembang Pada Evolusi Perangkat Lunak. *Jurnal Ilmiah Teknologi Informasi*, *11*(2), 3–9.

Setiawan, Z., Jauhar, N., Putera, D. A., Santosa, A. D., Fenanlampir, K., Sembel, H. F., Harto, B., Roza, T. A., Dermawan, A. A., & Rukmana, A. Y. 2023. *Kewirausahaan Digital*. Global Eksekutif Teknologi.

Sudirjo, F., Rukmana, A. Y., Wandan, H., & Hakim, M. L. 2023. Pengaruh Kapabilitas Pemasaran, Digital Marketing Dalam Meningkatkan Kinerja Pemasaran UMKM Di Jawa Barat. *Jurnal Bisnisman: Riset Bisnis Dan Manajemen*, *5*(1), 55–69.

Sugiarto, I., Napu, F., Rukmana, A. Y., & Hastuti, P. 2023. Kesuksesan Wirausaha di Era Digital dari Perspektif Orientasi Kewirausahaan (Study Literature). *Sanskara Ekonomi Dan Kewirausahaan*, *1*(02), 81–96.

Vassiliadis, P., Shehaj, F., Kalampokis, G., & Zarras, A. 2023. Joint source and schema evolution: Insights from a study of 195 foss projects. *Proceedings of the 26th International Conference on Extending Database Technology, EDBT*, 28–31.

Wakil, A., Cahyani, R. R., Harto, B., Latif, A. S., Hidayatullah, D., Simanjuntak, P., Rukmana, A. Y., & Sihombing, F. A. H. 2022. *Transformasi Digital Dalam Dunia Bisnis*. Global Eksekutif Teknologi.

Wu, H., Shi, L., Chen, C., Wang, Q., & Boehm, B. 2016. Maintenance effort estimation for open source software: A systematic literature review. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 32–43.

Yanto Rukmana, A., Harto, B., & Gunawan, H. 2021. Analisis Urgensi Kewirausahaan Berbasis Teknologi (Technopreneurship) dan Peranan Society 5.0 dalam Perspektif Ilmu Pendidikan Kewirausahaan. In *Jurnal Sains Manajemen & Akuntansi)* (Vol. 13, Issue 1).

BAB 10
PENGGUNAAN ULANG PERANGKAT LUNAK

Oleh Rahman Abdillah

10.1 Pendahuluan

Rekayasa perangkat lunak *(software engineering)* berbasis *reuse* adalah strategi pengembangan perangkat lunak di mana proses pengembangan ditujukan untuk menggunakan kembali perangkat lunak yang sudah ada sebelumnya. Sampai dengan sekitar tahun 2000, penggunaan metode *reuse* perangkat lunak secara sistematis masih jarang terjadi, tetapi saat ini reuse digunakan secara luas dalam pengembangan sistem perangkat lunak yang baru, contohnya: opensource. Konsep pengembangan perangkat lunak berbasis reuse ini sebagai tanggapan terhadap tuntutan untuk dapat menurunkan biaya produksi dan pemeliharaan sistem perangkat lunak, pengiriman sistem *(deliverables)* yang lebih cepat, serta peningkatan kualitas perangkat lunak. Perusahaan pengembang *(developer)* melihat perangkat lunak mereka sebagai aset yang berharga. Perusahaan mendorong penggunaan kembali sistem-sistem yang sudah ada untuk meningkatkan pengembalian investasi dari perangkat lunak yang sudah ada sebelumnya. Beberapa perusahaan *(developer)* besar menyediakan berbagai komponen yang dapat digunakan kembali untuk pelanggan *(user/customer)* mereka. Perangkat lunak yang sudah berstandar, seperti standar pada situs layanan web memudahkan pengembangan perangkat lunak untuk dapat digunakan kembali pada berbagai aplikasi dan sistem informasi perangkat lunak yang lainnya.

Kategori pengembangan perangkat lunak yang dapat digunakan kembali bisa saja memiliki ukuran dalam rentang yang berbeda, dibawah ini adalah contoh kategori *reuse* pengembangan perangkat lunak dalam ukuran yang berbeda (Dennis, *et. al.,* 2015):

1. *System Reuse.* Suatu perangkat lunak bisa saja terdiri dari beberapa sistem perangkat lunak yang menjadi bagian yang lebih kecil, namun bagian kecil tersebut tetap berbentuk sistem. Penggunaan ulang sistem perangkat lunak, dimaksudkan untuk dapat menggunakan sistem yang lebih kecil tersebut sebagai bagian dari sistem perangkat lunak keseluruhan.
2. *Application Reuse.* Suatu aplikasi dapat digunakan kembali dengan menggabungkannya kedalam sistem lain yang lebih besar tanpa melakukan perubahan besar atau dengan mengkonfigurasikan aplikasi untuk pengguna *(user)* yang berbeda. Pada dasarnya, arsitektur perangkat lunak dari basis aplikasi atau garis produk perangkat lunak memiliki karakter yang umum, tetapi disesuaikan dengan kebutuhan individual pengguna *(user)*. Hal ini dapat dilakukan dengan menggunakan proses integrasi aplikasi perangkat lunak kedalam sistem perangkat lunak lain yang lebih besar.
3. *Component Reuse.* Sebuah aplikasi atau sistem perangkat lunak bisa terdiri dari komponen-komponen, mulai dari subsistem, modul hingga kelas *(class)* dan objek tunggal. Seluruh komponen perangkat lunak ini dapat digunakan kembali. Sebagai contoh, sebuah sistem perangkat lunak melakukan pencocokan pola *(pattern matching)* sistem pemrosesan teks sehingga dapat digunakan kembali, hal ini merupakan bagian dari sistem manajemen database. Komponen-komponen ini bisa saja terdapat pada cloud atau host, ataupun server pribadi dan dapat diakses melalui suatu layanan antarmuka pemrograman aplikasi (API).

4. *Object & Function Reuse.* Sistem perangkat lunak bisa terdiri dari penerapan fungsi-fungsi tunggal atau formula, seperti fungsi matematika, atau kelas objek dapat digunakan kembali. Para perancang perangkat perangkat lunak sudah merancang fungsi-fungsi untuk menyusun perangkat lunak dengan pola yang standar selama 40 tahun terakhir. Programmer dapat menemukan fungsi dan kelas yang tersedia dengan bebas penggunaannya dan menghubungkan dengan kode *(coding)* perangkat lunak yang baru dikembangkan. Pada bidang-bidang seperti algoritma matematika dan grafis, objek dan fungsi tidak banyak mengalami perubahan sehingga dapat digunakan kembali dengan alasan efisiensi, efektifitas dan tentu saja untuk penghematan biaya pengembangan perangkat lunak.

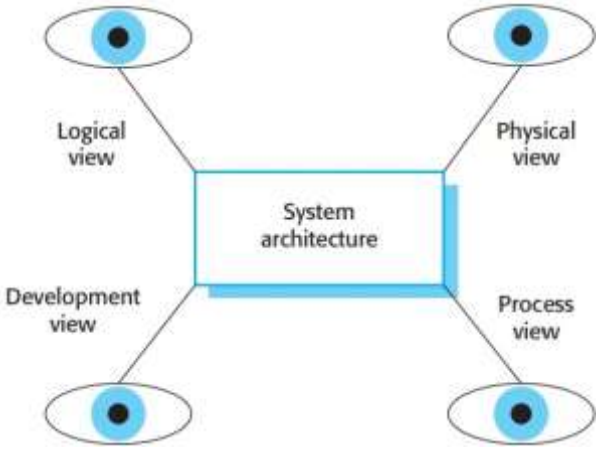
Semua sistem perangkat lunak dan komponen yang mencakup fungsi generik berpotensi dapat digunakan kembali. Namun, sistem atau komponen ini terkadang sangat spesifik sehingga cukup mahal untuk memodifikasinya hingga dapat diintegrasikan kedalam sistem perangkat lunak yang baru. Alih-alih menggunakan kembali kode *(coding)* pemrograman, kita sebenarnya juga dapat menggunakan kembali ide-ide atau konsep yang telah digunakan sebagai dasar/pondasi perancangan perangkat lunak sebelumnya, hal ini disebut sebagai konsep penggunaan ulang *(concept reuse)*.

Dalam penggunaan ulang konsep, kita tidak menggunakan kembali komponen-komponen perangkat lunak yang spesifik, namun sebaliknya kita dapat menggunakan kembali, ide, cara kerja, atau algoritma perangkat lunak yang sudah ada sebelumnya. Konsep yang kita gunakan kembali direpresentasikan kedalam notasi model, seperti pemodelan sistem, diagram atau tabel yang tidak mencakup detail implementasi. Oleh karena itu, penggunaan ulang konsep ini dapat dikonfigurasikan dan disesuaikan untuk berbagai situasi

kasus atau bisnis. Konsep *reuse* diimplementasikan dalam pendekatan seperti pola desain pada Bab.8 Desain dan Implementasi Perangkat Lunak, produk sistem perangkat lunak yang dapat dikonfigurasi, dan koding pembangkit program. Pada saat konsep *reuse* digunakan, proses *reuse* harus mencakup aktivitas dimana konsep dalam pemodelan dapat diwujudkan untuk membuat komponen *reuse* dapat dijalankan. Salah satu keuntungan yang jelas dari konsep *reuse* adalah bahwa biaya pengembangan perangkat lunak secara keseluruhan lebih rendah dan lebih sedikit komponen perangkat lunak yang harus dibuat dari awal, dirancang, diimplementasikan serta divalidasi.

10.2 Architectural Design

Untuk dapat memahami konsep *reuse* sistem perangkat lunak, ada baiknya kita mempelajari desain arsitektur dari sebuah sistem. Kita tidak dapat merepresentasikan seluruh informasi tentang arsitektur sistem perangkat lunak hanya dalam satu diagram, hal ini dikarenakan pemodelan secara grafis atau diagram hanya dapat digunakan untuk menampilkan satu pandangan atau aspek dari keseluruhan sistem perangkat lunak, contohnya: ERD digunakan untuk menggambarkan aspek basis data, Flowchart digunakan untuk menggambarkan algoritma, dsb. Diagram-diagram tersebut dapat digunakan untuk menunjukkan bagaimana sistem perangkat lunak diuraikan menjadi modul-modul bagian perangkat lunak, bagaimana proses dan waktu *(timeline)* eksekusi dan interaksi, atau bagaimana data informasi didistribusikan melalui jaringan sistem. Aspek-aspek sistem perangkat lunak ini bisa saja digunakan sebagai informasi atau referensi yang bermanfaat dalam waktu dan kondisi yang berbeda-beda, baik untuk implementasi desain maupun pada saat dokumentasi pelaporan.



Gambar 10.1. Architectural Views
(Sumber : Sommerville, 2016)

Dari gambar diatas, dapat kita lihat terdapat beberapa pandangan dari sebuah sistem arsitektur perangkat lunak. Menurut Krutchen (Sommerville, 2016) dalam model pandangan arsitektur *(architectural views)* yang terkenal bisa disebut sebagai model 4 +1. Gambar tersebut menggambarkan empat pandangan arsitektur yang fundamental yang dapat terhubung melalui skenario yang umum atau *use case*.

1. Pandangan logis (*logical view*), yang menunjukkan abstraksi utama dalam sistem sebagai objek atau kelas objek. Dalam pandangan ini, harus memungkinkan untuk mengaitkan persyaratan sistem dengan entitas dalam pandangan logis ini.
2. Pandangan proses (*process view*), yang menunjukkan bagaimana, saat runtime, sistem terdiri dari proses-proses yang saling berinteraksi. Pandangan ini berguna untuk membuat penilaian tentang karakteristik sistem non-fungsional seperti kinerja dan ketersediaan.
3. Pandangan pengembangan (*development view*), yang menunjukkan bagaimana perangkat lunak diuraikan untuk

pengembangan; yaitu, menampilkan pembagian perangkat lunak menjadi komponen-komponen yang diimplementasikan oleh satu pengembang atau tim pengembangan. Pandangan ini berguna bagi manajer perangkat lunak dan para programmer.

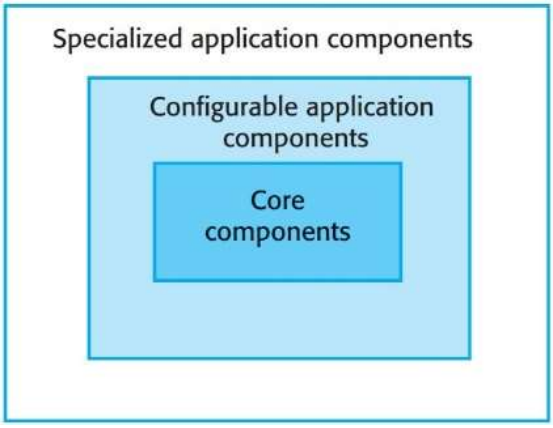
4. Pandangan fisik (*physical view*), yang menunjukkan perangkat keras sistem dan bagaimana komponen perangkat lunak didistribusikan di antara prosesor-prosesor dalam sistem. Pandangan ini berguna bagi insinyur sistem yang merencanakan implementasi sistem.

10.3 Garis Produk Perangkat Lunak

Ketika sebuah perusahaan perangkat lunak harus mendukung pengembangan sejumlah sistem yang mirip namun tidak identik, salah satu pendekatan yang paling efektif untuk penggunaan ulang adalah dengan menciptakan sebuah garis produk perangkat lunak (*Software Product Lines*). Sistem kontrol perangkat keras (*hardware*) seringkali dikembangkan menggunakan pendekatan ini untuk penggunaan ulang, begitu juga dengan aplikasi khusus domain di bidang seperti logistik atau sistem perangkat lunak untuk medis. Sebagai contoh, sebuah produsen printer harus mengembangkan perangkat lunak kontrol printer, di mana ada versi khusus produk untuk setiap jenis printer. Versi-versi perangkat lunak ini memiliki banyak kesamaan, sehingga cukup masuk akal untuk membuat produk perangkat lunak inti (garis produk) dan menyesuaikannya untuk setiap jenis printer.

Sebuah garis produk perangkat lunak adalah kumpulan aplikasi dengan arsitektur yang umum dan komponen yang dapat digunakan bersama, dengan setiap aplikasi yang disesuaikan untuk mencerminkan persyaratan pelanggan tertentu. Sistem inti ini dirancang sehingga dapat dikonfigurasikan dan disesuaikan untuk memenuhi kebutuhan

pelanggan dengan peralatan yang berbeda. Hal ini mungkin saja melibatkan konfigurasi beberapa komponen, implementasi komponen tambahan, dan modifikasi beberapa komponen untuk mencerminkan persyaratan baru. Pengembangan aplikasi dengan menyesuaikan versi generik dari aplikasi berarti bahwa sebagian besar kode aplikasi dapat digunakan kembali dalam setiap sistem. Tahap pengujian perangkat lunak dapat disederhanakan karena pengujian untuk bagian-bagian besar perangkat lunak juga dapat digunakan kembali, sehingga mengurangi waktu pengembangan perangkat lunak secara keseluruhan. Teknisi atau manajer pengembang produk dapat mempelajari domain perangkat lunak melalui garis produk perangkat lunak dan menjadi ahli yang dapat bekerja dengan cepat untuk dapat mengembangkan sistem perangkat lunak baru. Garis produk perangkat lunak biasanya muncul dari aplikasi yang sudah ada.



Gambar 10.2. Komponen Garis Produk Sistem
(Sumber : Sommerville, 2016)

Pada gambar diatas (gambar 10.2) dapat kita lihat komponen garis produk sistem perangkat lunak yang terdiri dari 3 aspek, yaitu:

1. Komponen inti (*core components*) yang menyediakan dukungan infrastruktur. Biasanya, komponen ini tidak dimodifikasi ketika mengembangkan fitur baru dari garis produk.
2. Komponen yang dapat dikonfigurasi atau dimodifikasi dan dikonfigurasi untuk dikhususkan pada pengembangan sistem perangkat lunak yang baru (*configurable application components*). Terkadang, bagian komponen ini dapat dikonfigurasi ulang tanpa harus mengubah *coding* yang asli, dengan cara menggunakan pemrograman hasil konfigurasi komponen bawaan.
3. Komponen khusus (*specialized application components*), beberapa komponen pada bagian ini mungkin saja diganti sebagian atau semua bagian ketika menciptakan sistem perangkat lunak yang baru dikarenakan ketidaksesuaian yang cukup signifikan.

Dengan memahami komponen-komponen garis produk sistem diatas, pengembang sistem perangkat lunak dapat melihat kesesuaian bagian klasifikasi komponen *reuse* yang dapat digunakan pada sistem perangkat lunak yang baru. Hal ini membutuhkan identifikasi fungsionalitas umum dari sistem perangkat lunak dan deskripsi yang jelas untuk pengembangan di masa depan.

10.4 Pengembangan berbasis Komponen

Dalam konteks teori rekayasa perangkat lunak, penggunaan ulang (*reuse*) adalah suatu gagasan yang ada sejak lama namun ada juga gagasan *reuse* yang baru. Para programmer telah menggunakan kembali gagasan, abstraksi,

dan proses sejak awal era komputasi, namun pendekatan awal terhadap penggunaan ulang masih digunakan dalam lingkup kecil untuk maksud tertentu. Saat ini, sistem berbasis komputer yang kompleks dan berkualitas tinggi harus dibangun dalam waktu yang singkat dan memerlukan pendekatan yang lebih terorganisir terhadap konsep penggunaan ulang. Rekayasa perangkat lunak berbasis komponen (*Component-based software engineering/CBSE*) adalah suatu proses yang menekankan pada desain dan konstruksi sistem berbasis komputer dengan menggunakan "komponen" perangkat lunak yang dapat digunakan kembali (Pressman, 2015). Adapun beberapa pertanyaan yang muncul tentang konsep penggunaan ulang komponen untuk pengembangan sistem perangkat lunak adalah sebagai berikut: Apakah memungkinkan untuk membangun sistem perangkat lunak yang kompleks dengan menggabungkan komponen-komponen yang berasal dari katalog komponen perangkat lunak sebelumnya? Apakah hal ini dapat dilakukan secara efisien dilihat dari segi biaya dan waktu? Dapatkah pemberian insentif yang tepat dibentuk untuk mendorong para programmer perangkat lunak dalam menggunakan ulang daripada membuat dari awal? Apakah manajemen bersedia menanggung biaya tambahan yang terkait dengan pembuatan komponen perangkat lunak yang dapat digunakan kembali? Dapatkah komponen-komponen yang sudah ada dalam jumlah yang sangat besar, ditemukan oleh mereka (*programmer*) yang memerlukannya?

Dari beberapa pertanyaan yang diajukan oleh pengembang perangkat lunak tersebut diatas, sehingga muncul konsep analisis komponen perangkat lunak dalam rekayasa domain (*domain engineering)*. Tujuan dari rekayasa domain adalah untuk mengidentifikasi, membangun, mencatat, dan menyebarkan sejumlah komponen perangkat lunak yang dapat diterapkan pada perangkat lunak yang sudah ada dan yang

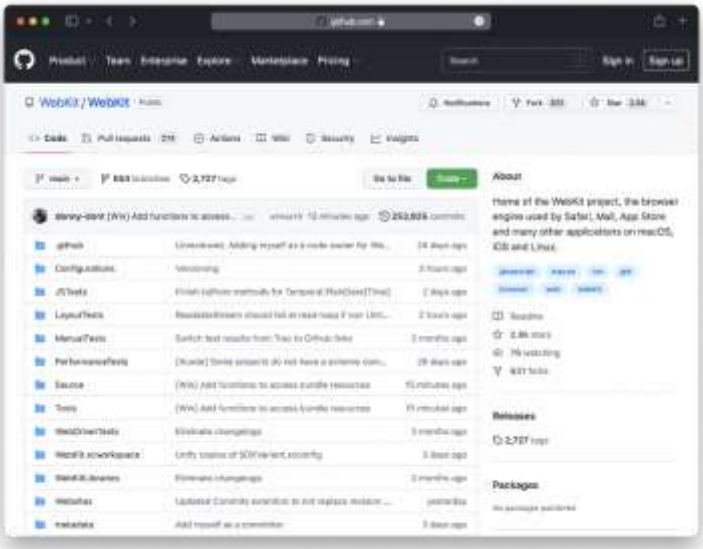
akan datang dalam suatu domain aplikasi tertentu. Tujuan keseluruhan *domain engineering* adalah untuk menetapkan mekanisme yang memungkinkan insinyur perangkat lunak (*programmer*) untuk berbagi komponen-komponen perangkat lunak dan menggunakannya kembali. Rekayasa domain melibatkan tiga kegiatan utama, yaitu: analisis, pembangunan, dan penyebaran.

Pendekatan keseluruhan terhadap analisis domain sering kali dikategorisasikan dalam konteks rekayasa perangkat lunak berorientasi objek. Langkah-langkah dalam proses ini adalah: (1) mendefinisikan domain yang akan diselidiki, (2) mengkategorikan item yang diekstraksi dari domain, (3) mengumpulkan sampel representatif dari aplikasi dalam domain tersebut, (4) menganalisis setiap aplikasi dalam sampel dan menentukan kelas analisis, dan (5) mengembangkan model persyaratan untuk kelas-kelas tersebut. Penting untuk dicatat bahwa analisis domain dapat diterapkan pada paradigma rekayasa perangkat lunak apa pun dan dapat digunakan untuk pengembangan konvensional maupun berorientasi objek (Pressman, 2015).

Rekayasa domain (*domain engineering*) menyediakan berbagai macam komponen yang dapat digunakan kembali untuk CBSE (*Component-based Software Engineering*). Beberapa dari komponen-komponen yang dapat digunakan kembali ini dikembangkan di dalam perusahaan, beberapa lainnya dapat diekstrak dari sistem perangkat lunak yang sudah ada, dan ada pula komponen yang diperoleh dari pihak ketiga. Sayangnya, keberadaan komponen-komponen yang dapat digunakan kembali ini tidak menjamin bahwa komponen-komponen tersebut dapat diintegrasikan dengan mudah atau efektif ke dalam arsitektur yang dipilih untuk aplikasi baru. Oleh karena itu, urutan tindakan pengembangan berbasis komponen perlu dilakukan ketika sebuah komponen direkomendasikan untuk digunakan kembali.

Domain untuk mencari komponen perangkat lunak dapat ditemukan di berbagai sumber. Beberapa domain sebagai tempat populer yang dapat digunakan untuk mencari komponen perangkat lunak adalah sebagai berikut:

1. GitHub: GitHub adalah platform hosting kode sumber terbesar di dunia. Kita dapat mencari proyek atau program open source yang relevan dengan domain tertentu menggunakan fitur pencarian pada GitHub.
2. SourceForge: SourceForge adalah repositori perangkat lunak open source dan platform pengembangan perangkat lunak. Kita dapat menemukan berbagai macam proyek open source pada domain ini.
3. Bitbucket: Bitbucket adalah platform pengembangan perangkat lunak yang mendukung repositori git. Kita juga dapat mencari perangkat lunak open source di domain ini.
4. Google Open Source: Google Open Source adalah situs yang menyediakan daftar proyek open source yang didukung oleh Google. Kita dapat menemukan proyek open source yang dibuat atau didukung oleh Google di domain ini.
5. *Open Source Directory*: Ada beberapa situs direktori perangkat lunak open source yang mengkategorikan proyek berdasarkan domain tertentu. Anda dapat mencari proyek open source yang sesuai dengan kebutuhan kita di situs-situs tersebut.
6. Reddit: Ada beberapa subreddit (sub-forum) di Reddit yang didedikasikan untuk diskusi dan berbagi perangkat lunak open source. Kita dapat mencari di sub-reddit tersebut untuk menemukan proyek open source yang relevan.
7. *Apache Software Foundation*: Apache Software Foundation adalah organisasi yang menyediakan banyak proyek perangkat lunak open source yang terkenal dan bermanfaat.



Gambar 10.3. Tampilan Github
(Sumber : webkit.org/blog/13140/webkit-on-github/)

10.5 Analisis dan Desain untuk Penggunaan Ulang

Ketersediaan komponen-komponen untuk dapat digunakan kembali yang telah kita bahas pada bagian garis produk perangkat lunak dan pengembangan berbasis komponen, dalam suatu proses pengembangan sistem perangkat lunak terkadang perlu dilakukan suatu langkah pencocokan spesifikasi. Jika komponen yang kita butuhkan telah sesuai dengan spesifikasi yang ada pada *domain engineering*, kita dapat mengambil komponen tersebut dari perpustakaan (*repository*) atau domain penggunaan ulang dan menggunakannya dalam perancangan sistem perangkat lunak kita yang baru. Namun, jika komponen-komponen yang kita butuhkan tidak dapat ditemukan, barulah kita membuat komponen yang baru. Pada saat inilah ketika kita mulai

membuat komponen yang baru, penerapan sebuah konsep yang dinamakan desain untuk penggunaan ulang (*Design for Reuse/DFR*) perlu dipertimbangkan.

Untuk dapat menerapkan konsep DFR, kita perlu menerapkan konsep dan desain perancangan perangkat lunak yang kokoh (*solid*). Namun, karakteristik dari domain perangkat lunak yang baru juga harus dipertimbangkan. Jika domain perangkat lunak memiliki struktur data global yang standar, komponen juga harus dirancang dengan menggunakan struktur data yang standar. Protokol antarmuka yang standar dalam domain perangkat lunak harus diadopsi, dan arsitektur yang sesuai untuk domain tersebut dapat digunakan sebagai template untuk desain arsitektur perangkat lunak yang baru. Setelah struktur data, antarmuka, dan template program yang standar telah ditetapkan, kita memiliki kerangka kerja untuk membuat desain yang solid. Komponen-komponen baru yang sesuai dengan kerangka kerja ini memiliki probabilitas yang lebih tinggi untuk penggunaan ulang (*reuse*) di masa mendatang.

Sebagai seorang developer perangkat lunak (*software engineer*), kita bisa membayangkan sebuah repositori komponen yang sangat besar. Ratusan atau bahkan ribuan komponen perangkat lunak yang tersedia didalamnya dapat kita gunakan kembali. Namun, bagaimana cara menemukan komponen yang kita butuhkan dengan tepat? Untuk dapat menjawab pertanyaan ini, muncul pertanyaan lain: Bagaimana kita dapat menggambarkan komponen perangkat lunak dengan istilah yang tepat, yang tidak dapat disalahartikan dan dapat diklasifikasikan pada repositori? Hal ini merupakan pertanyaan yang cukup sulit untuk dijawab, namun setidaknya kita harus paham betul terkait istilah yang akan kita gunakan untuk menemukan komponen pada repositori dan juga studi kasus sistem perangkat lunak yang mirip dengan sistem yang akan kita kembangkan.

Sebuah komponen perangkat lunak yang dapat digunakan kembali dapat digambarkan dengan berbagai cara, namun deskripsi istilah komponen perangkat lunak yang ideal mencakup apa yang disebut Tracz sebagai model 3C: konsep, isi, dan konteks (*concept, content and context*). Adanya deskripsi atau istilah tentang apa yang dapat dicapai oleh sebuah komponen, bagaimana pencapaian tersebut dilakukan, serta dimana komponen tersebut berada dalam domain implementasinya. Untuk mendukung implementasi penggunaan kembali (*reuse*) digunakan kembali, beberapa komponen perangkat lunak memiliki karakteristik lingkungan penggunaan ulang sebagai berikut (Pressman, 2015):

1. Sebuah database komponen yang mampu menyimpan komponen perangkat lunak dan informasi klasifikasi yang diperlukan untuk mengambilnya kembali.
2. Sistem manajemen perpustakaan (*library management system*) yang menyediakan akses ke database tersebut.
3. Sistem pengambilan komponen perangkat lunak (*software component retreival system*), misalnya: objek request broker yang memungkinkan aplikasi klien untuk mengambil komponen dan layanan dari server perpustakaan/repositori.
4. Alat-alat CBSE (*Component-Based Software Engineering tools*) yang mendukung integrasi komponen yang digunakan kembali ke dalam desain atau implementasi baru.

DAFTAR PUSTAKA

Dennis, A. Wixom, B. H., Tegarden, D. 2015. *Systems Analysis & Design: An Object-Oriented Approach with UML 5th Edition.* New Jersey: Wiley

Pressman, R. S., Maxim, B. R. 2015. *Software Engineering A Practitioner's Approach 8th Edition.* New York: McGraw-Hill Education

Sommerville, I. 2016. *Software Engineering 10th Edition*. London: Pearson

BAB 11
TREN DAN MASA DEPAN REKAYASA PERANGKAT LUNAK

Oleh Teguh Wahyono

11.1 Pendahuluan

Perkembangan teknologi yang begitu cepat dari waktu ke waktu, tentu mempengaruhi perkembangan ilmu rekayasa perangkat lunak. Disamping itu kebutuhan pengguna juga berkembang, mengikuti perkembangan teknologi dan peradaban manusia. Dengan demikian agar tetap relevan untuk memenuhi kebutuhan pengguna, rekayasa perangkat lunak juga terus berkembang dan berevolusi. Beberapa hal yang mendorong perubahan tersebut diantaranya adalah permintaan pasar, perkembangan teknologi, pengetahuan yang lebih baik, perubahan kebutuhan, metode baru, serta tantangan keamanan yang terus berkembang.

Permintaan akan perangkat lunak yang lebih baik dari versi yang ada, terus meningkat seiring dengan perkembangan teknologi dan kebutuhan pengguna. Pengguna mengharapkan perangkat lunak yang lebih bagus, aman, handal, dan efisien. Untuk memenuhi kebutuhan inilah perlu terus dilakukan pengembangan dan peningkatan teknik serta metode rekayasa perangkat lunak.

Perkembangan teknologi merupakan pemicu utama evolusi rekayasa perangkat lunak (Pressman, 2019). Adanya teknologi terkini seperti komputasi awan, kecerdasan buatan, big data, *Internet of Things* (IoT), dan lain-lain, mempengaruhi perkembangan rekayasa perangkat lunak dimana dapat menciptakan solusi yang lebih inovatif dan kompleks.

Seiring dengan pengalaman dan penelitian yang terus berlanjut, *software developer* juga tentunya semakin memahami praktik terbaik dalam melakukan rekayasa perangkat lunak. Pengalaman dari proyek-proyek sebelumnya memberikan wawasan tentang kegagalan dan keberhasilan, yang memungkinkan untuk terus dilakukan peningkatan proses pengembangan perangkat lunak (Simarmata, 2010; Sommervile, 2017).

Disamping hal-hal tersebut di atas kebutuhan dan persyaratan pelanggan juga terus berubah seiring waktu. Lingkungan bisnis yang dinamis dan tuntutan pasar yang berubah memerlukan adaptasi perangkat lunak yang fleksibel. Rekayasa perangkat lunak harus mampu mengikuti perubahan ini dengan mengembangkan fitur baru, meningkatkan performa, dan memperbaiki masalah yang teridentifikasi.

Hal lain yang mempengaruhi perkembangan rekayasa perangkat lunak adalah adanya penemuan dan pengembangan metode baru. Rekayasa perangkat lunak terus mengalami kemajuan melalui penemuan dan pengembangan metode baru. Misalnya, pengembangan metode Agile dan DevOps telah memberikan pendekatan baru dalam pengembangan perangkat lunak yang lebih adaptif, kolaboratif, dan terus-menerus (Almeida, 2022).

Masalah keamanan dan kerentanan juga menjadi penyebab perubahan ilmu rekayasa perangkat lunak. Ancaman keamanan komputer terus berkembang dan berubah seiring waktu. Hal ini mendorong perkembangan rekayasa perangkat lunak untuk memperkuat keamanan dan mengatasi kerentanan. Keamanan perangkat lunak adalah masalah yang kompleks, dan upaya terus dilakukan untuk meningkatkan keamanan perangkat lunak melalui pengembangan teknik dan praktik yang lebih baik.

Chapter ini akan membahas tentang beberapa trend rekayasa perangkat lunak yang berkembang saat ini dan ke

depan. Shivsharan (2023) menuliskan 10 trend rekayasa perangkat lunak di tahun 2023 dan perkembangannya ke depan. Kesepuluh trend tersebut diantaranya adalah: (1) *Artificial Intelligence* atau Kecerdasan Buatan, (2) *Cybersecurity* atau keamanan siber, (3) Adopsi *Blockchain,* (4) Standard Kualitas Perangkat lunak dengan ISO/IEC 9126, (5) *Low-code Development,* (6) Aplikasi Web *Progressive,* (7) Software Outsourcing, (8) Layanan Cloud Terdistribusi, (9) *Internet of Things* (IoT), dan (10) *Mixed Reality*. Selanjutnya beberapa dari trend tersebut akan dibahas secara lebih lengkap pada sub bagian berikut ini.

11.2 Kecerdasan Buatan

Kecerdasan Buatan (*Artificial Intelligence*, AI) memiliki pengaruh yang signifikan dalam rekayasa perangkat lunak ke depan. AI akan membuat analisis dan pengolahan data dalam pengembangan perangkat lunak dilakukan dengan cara yang lebih cerdas. Teknik *Machine Learning* dan *Deep Learning* dalam AI memungkinkan perangkat lunak untuk belajar dari data yang ada dan mengidentifikasi pola atau tren yang tidak terlihat sebelumnya. Hal ini dapat membantu pengembang dalam membuat keputusan yang lebih baik dan mengoptimalkan proses pengembangan perangkat lunak (Heryadi, dkk. 2020).

AI juga dapat meningkatkan kecerdasan aplikasi. Aplikasi perangkat lunak dapat menjadi lebih cerdas dan adaptif. Misalnya, aplikasi dapat menggunakan algoritma pemrosesan bahasa alami untuk memahami dan merespons perintah pengguna secara lebih intuitif. Aplikasi juga dapat belajar dari pengalaman pengguna dan memberikan rekomendasi yang lebih personalisasi. Selanjutnya AI dapat membantu otomatisasi beberapa aspek pengembangan perangkat lunak. Contohnya adalah generate kode otomatis, di

mana AI dapat menghasilkan kode berdasarkan pemahaman terhadap kebutuhan dan spesifikasi tertentu. Ini dapat menghemat waktu dan usaha pengembang serta mempercepat siklus pengembangan perangkat lunak.

Dengan AI, pengujian dan pemeliharaan perangkat lunak juga akan yang lebih efisien. AI dapat digunakan untuk meningkatkan pengujian perangkat lunak dengan mengidentifikasi skenario pengujian yang relevan, mengotomatiskan pengujian, dan menemukan bug atau masalah yang mungkin terlewatkan. Dalam pemeliharaan perangkat lunak, AI juga dapat digunakan untuk pemantauan kinerja dan deteksi anomali, yang memungkinkan respons yang lebih cepat terhadap masalah yang muncul.

Terkait dengan internaksi manusia dan mesin, AI dapat digunakan untuk meningkatkan interaksi melalui penggunaan teknik seperti pemrosesan bahasa alami, pengenalan suara, atau deteksi emosi. AI dapat membuat antarmuka pengguna lebih intuitif dan memungkinkan pengguna berinteraksi dengan perangkat lunak menggunakan cara yang lebih alami. Selanjutnya AI juga dapat membantu mengoptimalkan performa perangkat lunak dengan mengidentifikasi *bottleneck* atau area yang membutuhkan perbaikan. Selain itu, AI juga dapat membantu dalam pengelolaan sumber daya dengan mengatur penggunaan sumber daya secara cerdas, sehingga meningkatkan efisiensi dan skalabilitas perangkat lunak.

Dengan demikian secara keseluruhan, kecerdasan buatan memiliki potensi untuk mengubah cara mengembangkan, menguji, dan memelihara perangkat lunak. Dengan memanfaatkan kecerdasan dan kemampuan analitis AI, rekayasa perangkat lunak dapat menjadi lebih efisien, adaptif, dan interaktif dengan pengguna.

11.3 Keamanan Siber dan Digital Trust

Dalam era digital yang membuat dunia terhubung satu dengan yang kain, ancaman siber semakin meningkat dan semakin kompleks. Perangkat lunak sering menjadi target para peretas untuk mencuri data sensitif, merusak sistem, atau mencari celah keamanan lainnya. Karena itu, keamanan siber menjadi aspek yang sangat penting untuk melindungi informasi dan infrastruktur dari ancaman-ancaman tersebut (Prakosa dkk, 2019).

Dengan mengingat pentingnya keamanan siber, maka dalam pengembangan perangkat lunak akan berlaku pendekatan *security by design* atau rancangan keamanan sistem dari awal. Artinya, keamanan akan menjadi pertimbangan utama sejak tahap perencanaan hingga implementasi. Pengembang akan mempertimbangkan potensi ancaman dan risiko keamanan dari awal, dan melibatkan praktik keamanan terbaik untuk menghindari atau meminimalkan kerentanan.

Uji keamanan atau *security testing* juga akan menjadi bagian integral dari proses pengembangan perangkat lunak. Pengujian keamanan akan dilakukan secara menyeluruh dan mendalam untuk mengidentifikasi kerentanan, celah keamanan, dan potensi risiko yang mungkin muncul. Dengan melakukan tes keamanan secara teratur, perangkat lunak dapat diperbaiki dan ditingkatkan untuk menghadapi ancaman baru yang muncul (Chotimah. 2019).

Untuk melawan ancaman siber yang semakin kompleks, pengembang perangkat lunak akan menggunakan teknologi keamanan yang lebih canggih. Contohnya adalah penggunaan enkripsi yang lebih kuat, penggunaan kecerdasan buatan dalam deteksi ancaman, dan penggunaan teknologi otentikasi yang lebih aman seperti biometrik.

Software Development juga akan fokus pada *Digital Trust* dimana kepercayaan menjadi hal yang sangat penting.

Pengembang perangkat lunak harus berfokus pada membangun kepercayaan digital dengan para pengguna. Ini melibatkan transparansi dalam penggunaan data, perlindungan privasi, dan kebijakan yang jelas tentang bagaimana data pengguna dikelola dan digunakan. Selanjutnya dengan meningkatnya kekhawatiran tentang keamanan siber, pemerintah dan badan regulasi mungkin akan mengenakan persyaratan lebih ketat terkait keamanan perangkat lunak. Pengembang perangkat lunak harus mematuhi regulasi ini dan memastikan perangkat lunak mereka memenuhi standar keamanan yang ditetapkan.

Dengan demikian secara keseluruhan, keamanan siber dan *digital trust* akan menjadi komponen krusial dalam rekayasa perangkat lunak ke depan. Pengembang perangkat lunak harus menghadapi tantangan keamanan yang semakin kompleks dengan fokus pada keamanan by design, pengujian keamanan yang intensif, dan penggunaan teknologi keamanan yang lebih canggih. Selain itu, membangun kepercayaan digital dengan pengguna akan menjadi faktor kunci untuk kesuksesan perangkat lunak di masa depan.

11.4 Komputasi Awan

Perkembangan komputasi awan atau *cloud computing* yang cukup pesat, memiliki pengaruh yang besar dalam rekayasa perangkat lunak ke depan. Hal itu dikarenakan kom putasi awan menyediakan lingkungan pengembangan yang fleksibel, skalabel, dan mudah diakses (Ercana. 2010). *Cloud computing* memungkinkan pengembang untuk dengan mudah meningkatkan atau menurunkan kapasitas komputasi dan sumber daya yang dibutuhkan untuk proyek pengembangan perangkat lunak. Ini memungkinkan tim pengembang untuk beradaptasi dengan kebutuhan yang berubah-ubah selama

siklus pengembangan dan memastikan performa aplikasi yang konsisten.

Dalam proyek rekayasa perangkat lunak, kolaborasi dan akses bersama menjadi hal yang penting, terutama jika tim pengembang berlokasi di berbagai tempat atau memiliki anggota tim yang bekerja secara remote. *Cloud computing* memungkinkan semua anggota tim untuk mengakses lingkungan pengembangan dan sumber daya perangkat lunak dari mana saja dengan koneksi internet.

Cloud computing akan membuat pengelolaan infrastrukturmenjadi lebih efisien. Dengan menggunakan *cloud computing*, pengembang perangkat lunak tidak perlu mengelola infrastruktur fisik atau membeli perangkat keras. Alih-alih, mereka dapat menggunakan infrastruktur cloud yang disediakan oleh penyedia layanan cloud, yang mengurangi biaya dan kompleksitas dalam pengelolaan infrastruktur. *Cloud computing* juga memungkinkan perangkat lunak dapat diakses dan digunakan secara global tanpa batasan geografis. Pengembang dapat dengan mudah mendistribusikan aplikasi dan layanan mereka ke berbagai wilayah dan negara, memperluas jangkauan bisnis mereka dengan lebih efisien.

Selanjutnya dalam rekayasa perangkat lunak, waktu pengujian dan pengembangan yang cepat menjadi kunci untuk menciptakan produk yang kompetitif. *Cloud computing* memungkinkan pengembang untuk dengan mudah menciptakan lingkungan pengujian dan pengembangan, mempercepat siklus pengembangan, dan memungkinkan penerapan perangkat lunak dengan cepat.

Kombinasi dari fleksibilitas, akses bersama, manajemen sumber daya yang mudah, kecepatan pengujian, dan penyediaan layanan yang kaya membuat *cloud computing* menjadi elemen yang kritis dalam rekayasa perangkat lunak modern. Semakin banyak perusahaan dan pengembang yang beralih ke lingkungan *cloud computing* untuk memaksimalkan

efisiensi, inovasi, dan kemampuan mereka dalam mengembangkan perangkat lunak yang canggih dan andal.

11.5 Internet of Things

Internet of Things (IoT) adalah konsep yang mengacu pada jaringan perangkat fisik yang terhubung ke internet dan dapat saling berkomunikasi dan bertukar data tanpa adanya interaksi manusia. Dalam IoT, berbagai perangkat seperti sensor, peralatan rumah tangga, kendaraan, perangkat wearable, dan banyak lagi, dilengkapi dengan teknologi yang memungkinkan mereka terhubung ke internet dan berbagi informasi dengan perangkat lainnya. IoT memiliki pengaruh yang besar dalam rekayasa perangkat lunak karena memperluas kemampuan perangkat lunak untuk berkomunikasi, berinteraksi, dan mengintegrasikan dengan berbagai perangkat fisik dan sensor (Susanto, 2022).

IoT memungkinkan pengembangan aplikasi yang terhubung dengan berbagai perangkat fisik, seperti sensor, perangkat pintar, dan peralatan rumah tangga. Rekayasa perangkat lunak untuk IoT melibatkan pembuatan aplikasi yang dapat mengakses, mengumpulkan, dan menganalisis data dari perangkat IoT untuk memberikan pengalaman yang lebih cerdas dan lebih terintegrasi bagi pengguna. Dalam konteks IoT, interaksi manusia dengan perangkat lunak bisa menjadi lebih kompleks karena melibatkan berbagai perangkat fisik. Pengembang perangkat lunak harus merancang antarmuka pengguna yang intuitif dan mudah digunakan untuk mengontrol dan memantau perangkat IoT dengan efisien.

Selanjutnya IoT menghasilkan jumlah data yang besar dari berbagai perangkat terhubung. Pengembang perangkat lunak harus merancang sistem yang dapat mengolah dan menganalisis data secara efisien untuk mengambil informasi yang berharga dari data yang dihasilkan oleh perangkat. IoT

juga menyajikan tantangan keamanan tambahan karena melibatkan berbagai perangkat dan konektivitas yang kompleks. Pengembang perangkat lunak harus memperhatikan keamanan dan privasi data dengan cermat untuk melindungi data sensitif dan mencegah potensi serangan atau penyalahgunaan.

IoT sering terkait dengan komputasi awan karena memungkinkan akses dan analisis data dari perangkat IoT secara terpusat. Rekayasa perangkat lunak dalam konteks IoT sering melibatkan pengembangan aplikasi berbasis cloud untuk mengelola dan menganalisis data secara efisien.

Dengan demikian secara keseluruhan, IoT memperluas cakupan dan kompleksitas rekayasa perangkat lunak karena melibatkan berbagai perangkat fisik yang saling terhubung. Pengembang perangkat lunak harus mempertimbangkan aspek unik dari IoT dalam merancang, mengembangkan, dan mengelola aplikasi IoT agar dapat berfungsi secara efisien dan aman. Dengan potensi untuk menghubungkan miliaran perangkat di seluruh dunia, IoT menawarkan peluang yang menarik dan tantangan menantang bagi para pengembang perangkat lunak.

11.6 Standard Kualitas Perangkat Lunak

Standar kualitas perangkat lunak adalah standar yang memberikan panduan tentang aspek-aspek yang harus diperhatikan dalam perencanaan pengembangan perangkat lunak. Salah satu standard yang digunakan adalah ISO/IEC 9126 Standard tersebut merupakan standar internasional yang menggambarkan model kualitas perangkat lunak. Model ini memberikan panduan dan kriteria yang digunakan untuk mengukur dan menilai kualitas perangkat lunak dalam berbagai aspek.

Dengan mengacu pada kriteria kualitas yang ditetapkan dalam ISO/IEC 9126, tim pengembang dapat merumuskan tujuan yang lebih jelas dan spesifik untuk mencapai tingkat kualitas yang diharapkan dalam perangkat lunak. Standar ISO/IEC 9126 juga akan membantu meningkatkan kualitas perangkat lunak dengan memberikan panduan tentang karakteristik kualitas yang perlu ditingkatkan. Penggunaan kriteria ini membantu tim pengembang untuk fokus pada aspek-aspek penting dari perangkat lunak, seperti keandalan, efisiensi, usabilitas, dan lain-lain.

Standar ini juga akan memberikan kerangka untuk mengukur dan menilai kualitas Perangkat Lunak. ISO/IEC 9126 menyediakan kerangka kerja dengan metode yang lebih terstruktur dan obyektif. Dengan menggunakan metrik kualitas yang ditetapkan dalam standar ini, tim pengembang dapat melakukan evaluasi yang lebih menyeluruh terhadap perangkat lunak dan mengidentifikasi area yang memerlukan perbaikan.

Adopsi ISO/IEC 9126 sebagai standar internasional, praktik pengukuran kualitas perangkat lunak juga menjadi lebih terstandarisasi dan seragam di seluruh industri perangkat lunak. Hal ini memungkinkan pembandingan yang lebih mudah antara berbagai produk dan proyek perangkat lunak. Kualitas perangkat lunak yang baik adalah faktor kunci untuk membangun kepercayaan pengguna. Dengan mengacu pada standar ISO/IEC 9126, pengembang dapat meningkatkan kualitas perangkat lunak mereka dan memastikan bahwa produk yang dihasilkan memenuhi standar kualitas yang diakui secara internasional.

Standar ini juga memasukkan aspek-usabilitas perangkat lunak, yang berarti pengembang perangkat lunak harus memprioritaskan pengalaman pengguna dalam proses pengembangan. Hal ini mendorong pembuatan antarmuka yang lebih intuitif dan ramah pengguna. Secara keseluruhan, ISO/IEC 9126 memiliki dampak positif pada perkembangan

rekayasa perangkat lunak ke depannya dengan memberikan panduan, kriteria, dan metrik yang membantu meningkatkan kualitas perangkat lunak. Dengan adopsi standar ini, diharapkan perangkat lunak yang dihasilkan akan menjadi lebih andal, efisien, dan sesuai dengan harapan pengguna.

11.7 Trend Metodologi: Agile, Scrum dan Devops

Terkait dengan metodologi dalam pengembangan perangkat lunak, *Agile, Scrum*, dan *DevOps* telah menjadi tren metodologi rekayasa perangkat lunak dan semakin populer saat ini (Almeida, 2022). Metodologi tersebut mengatasi tantangan dalam pengembangan perangkat lunak dengan memberikan pendekatan yang lebih adaptif, kolaboratif, dan terus-menerus. Dengan mengadopsi pendekatan ini, tim pengembang dapat lebih cepat menanggapi perubahan kebutuhan bisnis, memberikan produk yang lebih berkualitas, dan meningkatkan efisiensi proses pengembangan secara keseluruhan. Karena keberhasilan dan manfaat yang terbukti, ketiga metodologi ini menjadi tren yang kuat dalam rekayasa perangkat lunak dan akan terus memainkan peran penting dalam perkembangan perangkat lunak di masa depan

Agile adalah pendekatan pengembangan perangkat lunak yang berfokus pada kolaborasi tim, fleksibilitas, dan respons cepat terhadap perubahan kebutuhan. Metodologi ini menempatkan pengembangan perangkat lunak sebagai proses yang iteratif dan inkremental, dengan fokus pada pengiriman perangkat lunak yang berfungsi dengan cepat dan berulang-ulang selama siklus pengembangan. Agile menekankan komunikasi yang kuat antara tim pengembang dan pemangku kepentingan (stakeholders), serta keterlibatan aktif pengguna dalam proses pengembangan.

Scrum adalah kerangka kerja khusus yang beroperasi dalam paradigma Agile. Ini adalah salah satu pendekatan yang

paling populer dalam pengembangan perangkat lunak. Scrum berfokus pada pembagian proyek menjadi iterasi pendek yang disebut *sprint*, yang biasanya berlangsung selama 1 hingga 4 minggu. Setiap sprint menghasilkan penambahan fungsionalitas yang dapat digunakan. Scrum memperkuat kerjasama antara tim dengan menyelenggarakan pertemuan reguler seperti *Daily Stand-up*, *Sprint Planning*, *Sprint Review*, dan *Sprint Retrospective*.

Slanjutnya DevOps adalah pendekatan yang mengintegrasikan pengembangan perangkat lunak (*Development*) dengan operasi (*Operations*) untuk meningkatkan efisiensi dan kualitas proses pengembangan dan penerapan perangkat lunak. DevOps menekankan kerjasama yang erat antara tim pengembang dan tim operasi untuk memfasilitasi pengiriman perangkat lunak yang lebih cepat dan lebih andal. Automatisasi dan alat-alat yang dapat diulang adalah elemen penting dalam DevOps untuk menyederhanakan dan mempercepat pengiriman perangkat lunak.

DAFTAR PUSTAKA

Aji, M. Prakoso, Jerry Indrawan. 2019. *Cyberpolitics*: *Perspektif Baru Memahami Politik Era Siber*. Depok: Rajawali Pers.

Fernando Almeida. 2022. Exploring the Benefits of Combining DevOps and Agile, *Future Internet*, vol 14, no 2.

Fredy Susanto, dll. 2022. Implementasi IoT dalam Kehidupan Sehari hari, Jurnal Imagine, Vol. 2 No 1.

Hidayat Chusnul Chotimah. 2019. Tata Kelola Keamanan Siber dan Diplomasi Siber Indonesia di Bawah Kelembagaan Badan Siber dan Sandi Negara, *Jurnal Politica*, No. 2 (November 2019).

Ian Sommervile. 2017. *Software Engineering, 10th Edition.* Boston: Pearson Education, Inc.

Janner Simarmata. 2010. *Rekayasa Perangkat Lunak.* Yogyakarta: Penerbit Andi.

Made Sudarma. 2012. *Manajemen Proyek Teknologi Informasi*. Udayana University.

Pratik Shivsharan. 2023. T*op 10 Software Development Trends For 2023* [online] https://www.ailoitte.com/blog/ software-development-trends

Roger Pressman. 2019. *Software Engineering A Practitioner's Approach*. 9th Edition. Mc GrawHill.

Tuncay Ercana. 2010. Effective Use of Cloud Computing in Educational Institutions. *Procedia Social and Behavioral Sciences 2*: p. 938–942

Yaya Heryadi, Teguh Wahyono. 2020. *Machine Learning: Konsep dan Implementasi*, Penerbit Gava Media, Yogyakarta.

BIODATA PENULIS

Ade Maulana, S.Kom., M.T.I., CITSM
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer Universitas Pelita Harapan
Kampus Kota Medan

Ade Maulana merupakan dosen Program Studi Sistem Informasi Universitas Pelita Harapan kampus kota Medan. Ia tercatat sebagai lulusan Magister Teknologi Informasi Universitas Indonesia. Sebelum mengajar, dia bekerja dibeberapa perusahaan pada bidang teknologi informasi.

BIODATA PENULIS

Nono Heryana, M. Kom.
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer, Universitas Singaperbangsa Karawang

Penulis lahir di Lampung Barat tanggal 03 Maret 1989. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika di Universitas Singaperbangsa Karawang lulus tahun 2012 dan melanjutkan S2 pada Jurusan Ilmu Komputer di Universitas Budi Luhur lulus tahun 2016. Selain menempuh pendidikan formal penulis juga memiliki beberapa sertifikasi profesi, baik lokal maupun internasional. Seperti: *Google Certified Educator Level 1*, *Certified International of Internal Quality Audit (CIIQA)*, *Scrum Master Accredited Certification™ (SMAC), Scrum Foundation Professional Certificate (SFPC), Certified of Editor (C. Ed), Certified Business Operations Associate (CBOA®)* dan *Remote Worker Professional Certificate (RWPC)*. Penulis adalah dosen tetap pada Program Studi Sistem Informasi, Fakultas Ilmu Komputer, Universitas Singaperbangsa Karawang, sejak tahun 2016 sampai dengan sekarang.

BIODATA PENULIS

Ir. Johni S. Pasaribu, MT.
Dosen Program Studi Sistem Informasi
Fakultas IT Politeknik Piksi Ganesha

Penulis lahir di Medan tanggal 1 Juni 1966 dari pasangan HA Pasaribu (alm) dan Taruli Tambunan (alm). Penulis adalah anak keempat dari enam bersaudara. Pendidikan yang ditempuh yaitu SD Methodist Medan lulus tahun 1979, SMPN 88 Jakarta lulus tahun 1982 dan SMAN 3 Jakarta lulus tahun 1985. Kemudian melanjutkan studi di ITB jurusan Teknik Fisika spesialisasi Instrumentasi dan Kontrol. Penulis melakukan penelitian untuk tugas akhir di PT IPTN dengan judul Perancangan Sistem Kendali Fly By Wire Gerak Longitudinal Pada Engineering Flight Simulator Pesawat N250-100 dan menyelesaikan gelar sarjana teknik Insinyur (Ir) tahun 1992.

Kemudian penulis bekerja di PT. IPTN (sekarang PT Dirgantara Indonesia) di Departemen Iron Bird kemudian berpindah di Departemen Engineering Flight Simulator (EFS). Penulis terlibat dalam Operation and Maintenance untuk EFS sehingga simulator N250-100 ini dapat digunakan saat dilakukan pengujian-pengujian pesawat melalui simulator

sebelum pengujian langsung pesawat di udara. Tahun 1998 penulis melanjutkan studi S2 di ITB pada jurusan Teknik Informatika untuk spesialisasi Software Engineering, dimana ilmu yang diperoleh dapat diaplikasikan dalam pekerjaan di lapangan. Tahun 2001 penulis menyelesaikan pendidikan S2 dengan gelar MT mengambil penelitian yang berjudul Pengembangan Perangkat Lunak Simulasi Sistem Perasa Kendali Elevator Pesawat N250-100 dimana penelitian ini juga dilakukan di Engineering Flight Simulator (EFS) N250-100. Tahun 2004 penulis mulai aktif mengajar di Perguruan Tinggi di Bandung diantaranya pernah mengajar di Unikom, ITHB, STT Telkom (kini TelU). Kini penulis penuh mengajar dan menjadi dosen tetap di Politeknik Piksi Ganesha. Bidang keahlian mengajar dan penelitian penulis adalah Artificial Intelligence (AI), Decision Support System (DSS), Software Engineering dan Software Testing. Penulis juga aktif melakukan bimbingan mahasiswa S1 dan publikasi di jurnal-jurnal nasional maupun internasional.

BIODATA PENULIS

Addin Aditya, S.Kom., M.Kom.
Dosen Program Studi Sistem Informasi
Sekolah Tinggi Informatika dan Komputer Indonesia Malang

Penulis lahir di Malang tanggal 2 Juni 1991. Penulis adalah dosen tetap pada Program Studi Sistem Informasi, Sekolah Tinggi Informatika & Komputer Indonesia (STIKI) Malang. Penulis Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika di STIKI Malang dan melanjutkan S2 pada Jurusan Sistem Informasi di Institut Teknologi Sepuluh Nopember. Penulis menekuni bidang Sistem Informasi Manajemen dan Rekayasa Perangkat Lunak. Adapun penulis dapat dihubungi melalui email di addin@stiki.ac.id

BIODATA PENULIS

Elisawati, S.Kom., M.Kom.
Dosen Program Studi Sistem Informasi
Universitas Dumai

Penulis lulus dari program sarjana ilmu komputer STIKI Malang pada tahun 2003 dan memperoleh gelar master dalam bidang ilmu komputer dari Universitas Putra Indonesia (UPI) Padang, Indonesia pada tahun 2010 dengan fokus pada ilmu komputer kosentrasi Teknologi Informasi. penulis bekerja sebagai dosen tetap di Universitas Dumai dan aktif mengajar di beberapa universitas Dumai. Saat ini penulis telah menulis beberapa buku berjudul Machine Learning dan Digital Artificial Intelligence.

BIODATA PENULIS

Rudiansyah, S.Kom., M.Kom
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer Universitas Sjakhyakirti Palembang

Penulis lahir di Palembang tanggal 20 November 1979. Penulis adalah dosen tetap pada Program Studi Sistem Informasi Fakultas Ilmu Komputer, Universitas Sjakhyakirti Palembang. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Teknik Informatika. Penulis menekuni bidang Menulis.

BIODATA PENULIS

Drs. Amna, MIT

Memulai pendidikan dasar dan menengah di kota kelahirannya Takengon Aceh Tengah dan kemudian melanjutkan pendidikan tinggi di bidang matematika di USK Banda Aceh pada tahun 1988 dan memperoleh gelar sarjana pada 1992. Tahun 1993 melanjutkan program Pra S2 Matematika di ITB Bandung. Program S2 pula di tempuh pada Mathematics Department Fakulti Sains matematik dan komputer Universiti Kebangsaan Malaysia (UKM). Akhir tahun 1994 menukar jurusan ke Sains Komputer seiring dengan terbentuknya Fakulti Teknologi dan Sains Maklumat UKM. Selama melanjutkan studi S2 sangat aktif sebagai Tutor sambilan untuk mata kuliah matematik, statistik dan ilmu komputer, juga sebagai Asisten Research untuk bidang ilmu komputer dan rekayasa perangkat lunak. Setelah menamatkan S2 tahun 1997 dunia Dosenpun digeluti dengan memulai sebagai Dosen program degree sains sistem di kolej Politek MARA Bangi untuk franchise program UKM-KYPM. Kemudian sebagai dosen D3 sains komputer prancise dengan UiTM Shah Alam, UPM Serdang juga program degree software engineering dengan MMU Cyberjaya di KYPM Kuala Lumpur. Tahun 2001

kembali ke Kampus UKM Bangi sebagai dosen sains sistem. Tahun 2003 bergabung dengan Kolej SAL Kuala Lumpur sebagai dosen program prancise degree in software engineering dan degree in networking dengan Edith Cowan University Western Australia. Tahun 2004 bergabung dengan Universiti Tenaga Nasional (UNITEN) sebagai asisten research bidang parallel processing dan dosen matematik di college Of engineering sambil melanjutkan studi S3. Tahun 2008 bergabung dengan College of IT UNITEN sebagai dosen system and network departement. Tahun 2010 hingga sekarang bergabung dengan Universitas Gajah Putih sebagai dosen teknik informatika. Banyak jurnal dan proseding baik internasional dan nasional yang telah di hasilkan dalam jangka waktu tersebut. Juga mengisi berbagai macam konferensi, simposium, dan webinar. Sejak 2017 hingga sekarang sebagai dosen tamu prodi teknik informatika di STTP Paya kumbuh. Tigabelas book chapter yang telah di hasilkan: *The art of digital marketing* strategi pemasaran generasi milanial, *Fintech innovation essence, posisition & strategy*, Sistem operasi dan Tahapan rekayasa perangkat lunak, big data dan lain-lain. Hal terbaru dari riset pada 27 Desember 2021 mendapat tempat harapan 1 lomba Innovasi LISIK 2021menerusi judul riset Kamus Bahasa Gayo Berbasis android.

Lahir dari ayah Abdurrahman AS dan ibu Fatimah dan menikah dengan Dr. Anna Permatasari Kamarudin, S.Tp, MBA sejak tahun 2000 dan dikaruniakan dengan 3 putri diberi nama Adiba Imani Salsabila, Najla Raihana Kamila dan Khayla Rahadatul Fakhira.

Email Penulis: amnaa98@hotmail.com

BIODATA PENULIS

Angga Aditya Permana
Dosen Informatika
Universitas Multimedia Nusantara

Angga Aditya Permana (lahir pada Desember 1989 di Jakarta) seorang dosen full time di bidang Computer Science pada Universitas Multimedia Nusantara sejak tahun 2021, fokus penelitian pada kriptografi, steganografi serta keamanan computer, namun pada tahun 2021 sedang mendalami topik bioinformatika dan network science. Angga juga memiliki hobi yaitu touring dan juga bermain bulutangkis, saat ini sedang menjadi mahasiswa program doctoral pada IPB University. Memulai karir pertama kali sebagai Network Enginner tahun 2011 dan memulai profesinya sebagai dosen pada 2013 di kampus swasta yang ada di Jakarta, pernah juga mengajar di kampus negri yang berada di Jakarta dan Tangerang, juga pernah di undang menjadi dosen tamu untuk mengenalkan Bioinformatika di kampus negeri di Jakarta. Terimakasih..

BIODATA PENULIS

Arief Yanto Rukmana, S.T., M.M.
Mahasiswa Doktoral Universitas Pendidikan Indonesia
Dosen Perguruan Tinggi Indonesia Mandiri Bandung
Dosen Universitas Nurtanio Bandung

Penulis sebelumnya telah memegang posisi penting di sejumlah bisnis yang beroperasi pada skala nasional dan dunia. penulis saat ini berprofesi sebagai praktisi bisnis (Pemilik Perusahaan di Bidang Fashion, Kuliner dan Bisnis Digital juga sebagai seorang akademisi. Penulis mengajar di Universitas Nurtanio Bandung, dan dosen tetap di perguruan tinggi indonesia mandiri (STMIK IM & STIE STAN IM). alumni dari Program Studi Informatika (S1) Sekolah Tinggi Manajemen Informatika dan Komputer Indonesia Mandiri (STMIK IM), Program Magister Manajemen (S2) di Pasca Sarjana Universitas Widyatama dan sedang melanjutkan studi Pendidikan (S3) Program Doktoral Pendidikan Teknologi dan Vokasional di Universitas Pendidikan Indonesia. Penulis aktif sebagai pembicara dan narasumber kewirausahaan digital juga berdedikasi sebagai praktisi bisnis / owner / pemilik beberapa perusahaan yang bergerak pada industri food, fashion and fun. serta aktif dalam organisasi nasional maupun internasional.

Penulis juga seorang profesional dalam bidang public speaking & sebagai asesor kompetensi digital dan instruktur berpengalaman untuk sertifikasi kompetensi SKKNI BNSP Digital Marketing, Social Media Marketing dan Metodologi Pelatihan Level III, Graphic Design serta Sertifikasi Kompetensi beberapa lainnya yang diselenggarakan dinas tenaga kerja kota bandung, PT Rice INTI dan Balai Besar Pengembangan Latihan Kerja (BBPLK) Bandung. Selain itu pula penulis produktif sebagai pendamping UMKM dan pengelola Inkubator Bisnis Perguruan Tinggi Indonesia Mandiri dengan membina dan melatih / coaching, mentoring serta melakukan pendampingan bisnis UMKM / Start UP untuk mahasiswa yang berminat menjadi seorang Entrepreneur / Technopreneurship tangguh yang mampu berdayasaing pada kancah internasional.

BIODATA PENULIS

Rahman Abdillah, M.Tech
Dosen Program Studi Teknik Informatika
Fakultas Teknik dan Ilmu Komputer, Universitas Indraprasta PGRI

Penulis lahir di Malang tanggal 23 September 1985. Penulis adalah dosen tetap pada Program Studi Teknik Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Indraprasta PGRI, Jakarta. Menyelesaikan pendidikan jenjang D4 pada Jurusan Mekatronika, PENS-ITS Surabaya lulus tahun 2006 dan melanjutkan pendidikan S2 pada bidang keahlian Software Engineering, University of Canberra, Australia lulus tahun 2009.

BIODATA PENULIS

Dr. Teguh Wahyono, S.Kom, M.Cs
Dosen Program Studi S1 Bisnis Digital Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana

Penulis saat ini adalah dosen di Program Studi S1 Bisnis Digital, Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana. Menyelesaikan pendidikan S1 Teknik Informatika Universitas Kristen Duta Wacana Yogyakarta tahun 1998, S2 Ilmu Komputer di Universitas Gadjah Mada Yogyakarta tahun 2011 dan S3 Ilmu Komputer di Universitas Bina Nusantara pada tahun 2021.

Bidang peminatannya dalam akademik adalah bidang *Data Science*, *Machine Learning* dan *Artificial Inteligence.* Di bidang tersebut telah mendapatkan sertifikasi *Deep Learning Specialist* dari *International Institute of Applied Informatics* (IIAI) Japan serta *Azure AI Fundamental Certified* dari Microsoft. Disamping itu yang bersangkutan juga mendapatkan Sertifikat Kompentensi dalam Bidang *Software Development* dari Badan Nasional Sertifikasi Profesi (BNSP) Republik Indonesia. Saat ini penulis mengampu beberapa mata kuliah yang diantaranya adalah *Software Engineering*, *Artificial Intelligence*, *Web and mobile programming* dan Teknologi Bisnis Digital.