PENGEMBANGAN PERANGKAT LUNAK:
PROSES, METODE, DAN PRAKTIK TERBAIK

Penulis :
Purwita Sari
Lucky Indra Kesuma
Ali Impron
Safira Fegi Nisrina
Reska Mayefis

ISBN 978-623-125-597-4
9 786231 255976

PENGEMBANGAN PERANGKAT LUNAK: PROSES, METODE, DAN PRAKTIK TERBAIK

Purwita Sari
Lucky Indra Kesuma
Ali Impron
Safira Fegi Nisrina
Reska Mayefis

GET PRESS INDONESIA

PENGEMBANGAN PERANGKAT LUNAK: PROSES, METODE, DAN PRAKTIK TERBAIK

Penulis :
Purwita Sari
Lucky Indra Kesuma
Ali Impron
Safira Fegi Nisrina
Reska Mayefis

ISBN : 978-623-125-597-6

Editor : Ari Yanto M.Pd
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, Januari 2025

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 Pengembangan Perangkat Lunak: Proses, Metode, Dan Praktik Terbaik ini.

Buku Ini Membahas Pendahuluan Tentang Pengembangan Perangkat Lunak, Proses Pengembangan Perangkat Lunak, Metode Pengembangan Perangkat Lunak, Pengelolaan Tim Pengembang Perangkat Lunak, Tantangan Dan Isu-isu Terkini Dalam Pengembangan Perangkat Lunak.

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, Desember 2024

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1
PENDAHULUAN TENTANG PENGEMBANGAN PERANGKAT LUNAK

1.1 Konsep Dasar Pengembangan Perangkat Lunak

Pengembangan perangkat lunak merujuk pada proses perancangan, pengkodean, pengujian, dan pemeliharaan perangkat lunak untuk memenuhi kebutuhan tertentu. Proses ini dimulai dari analisis kebutuhan pengguna, desain perangkat lunak, hingga implementasi dan pemeliharaan. Tujuan utama dari pengembangan perangkat lunak adalah untuk menciptakan perangkat lunak yang efektif, efisien, dan sesuai dengan kebutuhan pengguna. Pada dasarnya, pengembangan perangkat lunak melibatkan kolaborasi antara tim pengembang, pemangku kepentingan, dan pengguna akhir untuk memastikan produk yang dihasilkan berkualitas tinggi dan memenuhi tujuan yang telah ditetapkan (González & García, 2020).

Pengembangan perangkat lunak (*software development*) adalah proses untuk merancang, membuat, menguji, dan memelihara aplikasi atau sistem perangkat lunak. Proses ini bertujuan untuk menciptakan perangkat lunak yang memenuhi kebutuhan pengguna, baik itu untuk penggunaan pribadi, organisasi, atau tujuan komersial. Pengembangan perangkat lunak mencakup berbagai aktivitas mulai dari analisis kebutuhan, desain, pengkodean, pengujian, hingga pemeliharaan setelah perangkat lunak dirilis.

Proses pengembangan perangkat lunak tidak hanya melibatkan aspek teknis, tetapi juga melibatkan komunikasi yang erat antara pemangku kepentingan, termasuk pengguna akhir, pengembang, dan tim pengelola proyek. Hal ini penting untuk memastikan bahwa perangkat lunak yang dikembangkan benar-benar dapat memenuhi kebutuhan yang telah ditetapkan dan dapat digunakan dengan mudah oleh pengguna (Sommerville, 2021).

1.2 Sejarah Pengembangan Perangkat Lunak

Pengembangan perangkat lunak dimulai pada awal abad ke-20 dengan perangkat lunak yang sederhana dan terbatas. Pada masa itu, perangkat lunak lebih banyak berupa program yang ditulis untuk mendukung komputer mekanis dan komputer pertama, seperti ENIAC (*Electronic Numerical Integrator and Computer*), yang pertama kali digunakan pada tahun 1945. Pemrograman pada masa itu masih sangat terkait dengan pengendalian perangkat keras, dan kode yang digunakan sangat mendasar, seringkali langsung terkait dengan mesin. Seiring dengan berjalannya waktu, bahasa pemrograman tingkat tinggi seperti Fortran (1957) dan COBOL (1959) diperkenalkan, membuat pengembangan perangkat lunak lebih mudah dan lebih efisien (Sommer et al., 2020).

Pada dekade 1960-an dan 1970-an, dengan semakin berkembangnya perangkat keras, kebutuhan akan perangkat lunak yang lebih kompleks dan canggih muncul. Salah satu perkembangan besar dalam sejarah pengembangan perangkat lunak adalah penciptaan sistem operasi. Sistem operasi memungkinkan perangkat keras untuk lebih efisien dan efektif digunakan, dan juga menyediakan antarmuka yang lebih ramah pengguna bagi pengembang dan operator. Sistem operasi pertama yang sukses adalah UNIX, yang dikembangkan pada 1969 oleh AT&T Bell Labs. UNIX kemudian menjadi dasar bagi banyak sistem operasi modern, seperti Linux dan macOS (Stallman, 2021).

Pada 1980-an dan 1990-an, dengan munculnya *personal computer* (PC), pengembangan perangkat lunak berkembang pesat. Komputer pribadi seperti Apple II dan IBM PC membawa perangkat lunak ke tangan pengguna umum. Pada periode ini, perangkat lunak aplikasi seperti pengolah kata (misalnya Microsoft Word) dan perangkat lunak grafis (seperti Adobe Photoshop) mulai dikenal luas. Pengenalan antarmuka grafis pengguna (GUI) oleh Apple dengan Macintosh (1984) dan Microsoft Windows pada 1985 mempercepat perkembangan perangkat lunak dengan menyediakan cara yang lebih intuitif bagi pengguna untuk berinteraksi dengan komputer (Wright & Ahmed, 2020).

Era 2000-an ditandai dengan revolusi internet, yang membawa dampak besar terhadap pengembangan perangkat lunak. Aplikasi berbasis web dan perangkat lunak berbasis *cloud* mulai berkembang pesat. Pengguna tidak lagi harus mengunduh perangkat lunak untuk menggunakannya; perangkat lunak sekarang dapat diakses melalui *browser web,* yang memudahkan kolaborasi dan akses jarak jauh. Pengenalan SaaS (*Software as a Service*) dan pengembangan platform berbasis cloud seperti Google Drive, Dropbox, dan Microsoft 365 membuat perangkat lunak lebih fleksibel dan terjangkau. Aplikasi berbasis cloud ini memungkinkan pengguna mengakses perangkat lunak secara *online*, menggantikan kebutuhan perangkat lunak lokal yang rumit dan mahal (Boehm, 1976).

Pada awal abad ke-21, perangkat lunak *open source* menjadi salah satu pendorong utama dalam pengembangan perangkat lunak. Proyek seperti Linux, yang dikembangkan oleh Linus Torvalds pada 1991, mengubah cara perangkat lunak dikembangkan dan didistribusikan. Perangkat lunak *open source* memungkinkan siapa saja untuk melihat, mengubah, dan mendistribusikan perangkat lunak tanpa biaya lisensi. Ini mendorong kolaborasi global di kalangan pengembang dan mengurangi ketergantungan pada

perangkat lunak *proprietary* dari perusahaan besar. Selain itu, pengembangan perangkat lunak berbasis komunitas ini memperkenalkan model baru dalam pengembangan perangkat lunak, di mana kontribusi dapat dilakukan oleh siapa saja, menghasilkan inovasi yang lebih cepat dan lebih luas (Sharma et al., 2022).

Sejak 2010-an, kecerdasan buatan (AI) dan pembelajaran mesin (*machine learning*) menjadi fokus utama dalam pengembangan perangkat lunak. Teknologi ini digunakan dalam berbagai bidang, mulai dari analisis data besar hingga asisten virtual seperti Siri dan Alexa. Pengembangan perangkat lunak AI berfokus pada algoritma yang memungkinkan komputer untuk mempelajari pola dan melakukan tugas tanpa pemrograman eksplisit. Ini juga mencakup pengembangan perangkat lunak untuk kendaraan otonom dan diagnosa medis berbasis AI. AI kini mengubah cara perangkat lunak berinteraksi dengan pengguna dan menyelesaikan masalah yang lebih kompleks secara otomatis (Li et al., 2021).

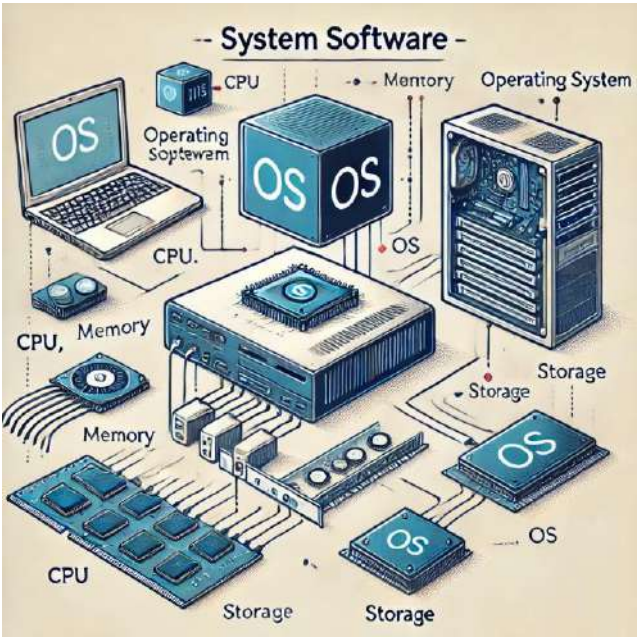
1.3 Jenis Perangkat Lunak

Berikut adalah penjelasan lengkap mengenai jenis perangkat lunak, dilengkapi dengan diagram yang menggambarkan berbagai jenis perangkat lunak. Setiap jenis perangkat lunak memiliki peran khusus dalam ekosistem teknologi informasi.

1.3.1 Perangkat Lunak Sistem

Perangkat lunak sistem mengacu pada perangkat lunak yang bertanggung jawab untuk mengelola perangkat keras komputer dan menyediakan platform bagi perangkat lunak aplikasi lainnya. Sistem operasi seperti Windows, Linux, dan macOS adalah contoh utama perangkat lunak sistem. Selain itu, perangkat lunak utilitas yang membantu memelihara dan mengoptimalkan sistem juga termasuk dalam kategori ini. Sistem operasi bertugas untuk mengelola sumber daya perangkat keras, memberikan antarmuka pengguna,

serta menjalankan aplikasi perangkat lunak lainnya (Bourgeois & Niazi, 2020).



Gambar 1.1. Perangkat Lunak Sistem
(Sumber : (DALL·E, OpenAI, 2024))

Gambar di atas menunjukkan bagaimana sistem operasi (OS) berfungsi sebagai lapisan inti yang mengelola perangkat keras seperti CPU, memori, dan penyimpanan, serta menyediakan antarmuka bagi aplikasi pengguna. Gambar ini memberikan gambaran yang jelas mengenai hubungan antara perangkat keras dan perangkat lunak sistem.

1.3.2 Perangkat Lunak Aplikasi

Perangkat lunak aplikasi adalah program yang dirancang untuk memenuhi kebutuhan pengguna dalam berbagai tugas spesifik, seperti pengolahan data, desain grafis, pengelolaan proyek, dan lain-lain. Contoh perangkat lunak aplikasi termasuk Microsoft Office untuk pengolahan kata, Adobe Photoshop untuk desain grafis,

dan QuickBooks untuk akuntansi. Perangkat lunak ini biasanya dibangun untuk meningkatkan produktivitas dan efisiensi dalam pekerjaan sehari-hari (Rathore et al., 2021).

Gambar 1.2. Perangkat Lunak Aplikasi
(Sumber : (DALL·E, OpenAI, 2024))

Gambar di atas menggambarkan Perangkat Lunak Aplikasi dalam konteks perangkat lunak yang digunakan untuk tugas spesifik oleh pengguna. Dalam gambar tersebut, terdapat beberapa ikon yang mewakili berbagai jenis perangkat lunak aplikasi yang umum digunakan dalam kehidupan sehari-hari seperti Microsoft Word, Adobe Photoshop, Microsoft Excel, dan Google Chrome.

1.3.3 Perangkat Lunak Pengembangan

Perangkat lunak pengembangan adalah alat yang digunakan oleh pengembang untuk menulis, menguji, dan memelihara kode aplikasi. Ini termasuk lingkungan pengembangan terintegrasi (IDE) seperti Visual Studio dan Eclipse, serta alat pengelolaan versi seperti

Git. Perangkat lunak pengembangan memungkinkan programmer untuk menciptakan perangkat lunak dengan cara yang lebih efisien dan terstruktur (Lo et al., 2020).

1.3.4 Perangkat Lunak *Middleware*

Middleware adalah perangkat lunak yang berfungsi sebagai perantara antara perangkat lunak aplikasi dan sistem operasi atau perangkat keras. Middleware memfasilitasi komunikasi dan manajemen data antara aplikasi atau antara aplikasi dan database. Misalnya, *server web* seperti *Apache* atau perangkat lunak pengelolaan *database* digunakan untuk memfasilitasi interaksi antara aplikasi dan *server* (Chen et al., 2021).

1.3.5 Perangkat Lunak ERP (*Enterprise Resource Planning*)

Perangkat lunak ERP digunakan oleh organisasi untuk mengintegrasikan dan mengelola berbagai fungsi bisnis, seperti akuntansi, pengelolaan sumber daya manusia, dan manajemen inventaris. ERP memungkinkan perusahaan untuk mengkoordinasikan dan memonitor aktivitas operasional secara efisien dalam satu sistem terpusat. Contoh perangkat lunak ERP populer termasuk SAP dan Oracle ERP (Wu et al., 2021).

Gambar 1.3. Perangkat Lunak ERP
(Sumber : (Microsoft, 2024))

Berikut adalah ilustrasi tambahan dari perangkat lunak ERP (*Enterprise Resource Planning*), yang menggambarkan sistem sebagai hub pusat dengan berbagai modul yang terhubung, seperti keuangan, inventaris, manufaktur, dan manajemen pelanggan. Ilustrasi ini menggunakan ikon digital futuristik dengan garis bercahaya yang menghubungkan setiap modul, mencerminkan integrasi dan efisiensi.

1.3.6 Perangkat Lunak Keamanan

Perangkat lunak keamanan dirancang untuk melindungi sistem komputer dan jaringan dari ancaman seperti virus, malware, dan peretasan. Ini termasuk antivirus, firewall, dan perangkat lunak enkripsi. Dalam dunia yang semakin terhubung secara digital,

perangkat lunak keamanan sangat penting untuk menjaga data dan sistem tetap aman dari serangan (Sharma et al., 2022).

1.3.7 Perangkat Lunak Berbasis *Cloud*

Perangkat lunak berbasis cloud memungkinkan pengguna untuk mengakses aplikasi dan data melalui internet tanpa perlu menginstalnya di perangkat lokal. Layanan seperti Google Workspace dan Microsoft 365 adalah contoh perangkat lunak berbasis cloud yang digunakan untuk kolaborasi dan penyimpanan data. *Cloud computing* memberikan fleksibilitas, skalabilitas, dan aksesibilitas dari mana saja dan kapan saja (Zhao et al., 2022).

1.4 Tahapan Pengembangan

Pengembangan perangkat lunak umumnya terbagi menjadi beberapa tahap, dan berikut ini akan dijelaskan tahap-tahap dalam pengembangan perangkat lunak.

1.4.1 Perencanaan (*Planning*)

Perencanaan dalam konteks pengembangan perangkat lunak adalah tahap awal yang sangat penting untuk memastikan proyek perangkat lunak berjalan dengan lancar. Pada tahap ini, tujuan dan ruang lingkup proyek ditentukan, serta sumber daya yang diperlukan, seperti tim pengembang, anggaran, dan jadwal. Perencanaan yang baik membantu meminimalkan risiko dan memastikan bahwa proyek dapat diselesaikan tepat waktu dan sesuai anggaran. Tanpa perencanaan yang jelas, proyek perangkat lunak dapat dengan mudah mengalami kegagalan dalam hal waktu dan biaya (Sommerville, 2021).

1.4.2 Analisis Kebutuhan (*Requirement Analysis*)

Pada tahap analisis kebutuhan, tim pengembang bekerja dengan pengguna untuk mengidentifikasi dan mendokumentasikan kebutuhan dan harapan yang harus dipenuhi oleh perangkat lunak.

Ini termasuk mengumpulkan informasi tentang fungsi yang diinginkan, kebutuhan performa, dan batasan sistem. Kebutuhan ini kemudian dijadikan dasar untuk mendesain dan membangun perangkat lunak. Analisis kebutuhan yang jelas dan tepat sangat penting untuk menghindari miskomunikasi dan memastikan bahwa perangkat lunak yang dikembangkan sesuai dengan kebutuhan pengguna (Pressman, 2020).

1.4.3 Desain (*Design*)

Desain adalah proses merencanakan struktur perangkat lunak dan memetakan kebutuhan pengguna ke dalam solusi teknis. Pada tahap ini, arsitektur perangkat lunak, antarmuka pengguna, dan struktur database dirancang. Desain yang baik harus mempertimbangkan kinerja, keamanan, dan skalabilitas sistem. Selain itu, desain juga mencakup pengembangan spesifikasi rinci yang akan digunakan oleh tim pengembang dalam tahap implementasi. Desain yang matang akan mempermudah proses pengembangan dan mengurangi kemungkinan kesalahan di kemudian hari (Chavez & García, 2020).

1.4.4 Implementasi (*Implementation*)

Pada tahap implementasi, desain yang telah dibuat diterjemahkan menjadi kode sumber menggunakan bahasa pemrograman yang sesuai. Tahap ini melibatkan penulisan kode untuk setiap komponen perangkat lunak dan pengujian unit untuk memastikan setiap bagian berfungsi dengan benar. Implementasi yang sukses memerlukan keterampilan teknis yang baik serta pemahaman yang jelas tentang desain dan kebutuhan perangkat lunak. Selain itu, pengujian unit juga dilakukan untuk memastikan bahwa setiap modul dapat berjalan dengan baik sebelum integrasi (Wright & Ahmed, 2020).

1.4.5 Pengujian (*Testing*)

Setelah perangkat lunak diimplementasikan, tahap selanjutnya adalah pengujian. Pengujian perangkat lunak bertujuan untuk memastikan bahwa sistem bekerja sesuai dengan spesifikasi yang telah ditetapkan dan bebas dari bug. Pengujian dilakukan dengan berbagai jenis, mulai dari pengujian unit, pengujian integrasi, hingga pengujian sistem secara keseluruhan. Proses ini penting untuk mendeteksi masalah yang tidak terlihat pada tahap pengembangan dan memastikan kualitas perangkat lunak sebelum diluncurkan ke pengguna (Rai & Yadav, 2021).

1.4.6 Pemeliharaan (*Maintenance*)

Pemeliharaan perangkat lunak adalah proses yang dilakukan setelah perangkat lunak dirilis dan berfungsi, dengan tujuan untuk memperbaiki masalah, meningkatkan kinerja, atau menambah fitur baru berdasarkan umpan balik pengguna. Pemeliharaan ini penting untuk memastikan perangkat lunak tetap relevan, efisien, dan bebas dari kesalahan seiring berjalannya waktu. Pemeliharaan perangkat lunak dapat melibatkan perbaikan *bug*, pembaruan sistem, atau peningkatan keamanan untuk mengatasi masalah yang muncul setelah penggunaan perangkat lunak dalam jangka panjang (Sommerville, 2021).

DAFTAR PUSTAKA

Bourgeois, D., & Niazi, M. (2020). Software System Architecture: Working with Stakeholders Using Viewpoints and Perspectives. CRC Press.

Chavez, L., & García, E. (2020). Software Testing and Quality Assurance in Agile Development. Wiley.

Chen, X., Wang, Y., & Zhang, L. (2021). Middleware for Communications: Performance, Scalability, and Security. Wiley-IEEE Press.

González, A., & García, M. (2020). Software Development Fundamentals: Concepts and Methodologies. Springer.

Li, X., Zhang, Z., & Liu, C. (2021). Artificial Intelligence and Machine Learning in Software Engineering. Elsevier.

Lo, D., Chen, W., & Xu, M. (2020). Software Engineering for Modern Web Applications. Springer.

Microsoft (2024). Microsoft Dynamics 365 ERP. Tersedia di: https://dynamics.microsoft.com (Diakses: 12 Desember 2024).

OpenAI, 2024. Generative Models for Image Synthesis. Available at: https://openai.com/dall-e [Accessed 12 December 2024].

Pressman, R. S. (2020). Software Engineering: A Practitioner's Approach. McGraw-Hill Education.

Rai, H., & Yadav, P. (2021). Managing Software Development Projects: A Practical Approach. Elsevier.

Rathore, M. M., Park, J. H., & Kim, D. S. (2021). Smart Software Systems and Applications. Springer.

Sharma, R., Singh, P., & Agarwal, A. (2022). Cybersecurity in the Age of Cloud Computing. CRC Press.

Sharma, R., Singh, P., & Agarwal, A. (2022). Open Source Software: Collaborating for the Future. Springer.

Sommer, C., Keuper, J., & Müller, A. (2020). History of Software Engineering: Foundations, Practices, and Future Directions. Springer.

Sommerville, I. (2021). Software Engineering: A Practitioner's Approach. McGraw-Hill Education.

Stallman, R. (2021). The UNIX Programming Environment. Prentice Hall.

Wright, J., & Ahmed, M. (2020). Personal Computing: A Historical Perspective. Wiley.

Wright, J., & Ahmed, M. (2020). Personal Computing: A Historical Perspective. Wiley.

Wu, Y., Wang, Z., & Chen, Y. (2021). Cloud Computing and Software as a Service: Trends, Issues, and Challenges. Springer.

Wu, Y., Wang, Z., & Chen, Y. (2021). Enterprise Resource Planning: Integrated Management of Core Business Processes. Wiley.

Zhao, W., Zhang, X., & Li, H. (2022). Cloud Computing and Big Data: Technologies, Applications, and Future Directions. Springer.

BAB 2
PROSES PENGEMBANGAN PERANGKAT LUNAK

2.1 Definisi Proses Pengembangan Perangkat Lunak

1. Konsep Dasar

   Proses pengembangan perangkat lunak didefinisikan sebagai kerangka sistematis yang mencakup serangkaian tahapan, metode, dan praktik untuk merancang, membuat, menguji, dan memelihara perangkat lunak komputer (Sommerville, 2015). Menurut Pressman dan Maxim (2020), proses ini tidak sekadar aktivitas teknis, melainkan pendekatan komprehensif yang menggabungkan aspek manajemen, rekayasa, dan kolaborasi tim.

2. Definisi Komprehensif

   a. Perspektif Akademik

      IEEE *Computer Society* (2014) mendefinisikan proses pengembangan perangkat lunak sebagai:

      Kerangka kerja terstruktur untuk transformasi kebutuhan pengguna menjadi produk perangkat lunak

      Serangkaian aktivitas terencana untuk menghasilkan sistem perangkat lunak berkualitas

      Metodologi sistematis untuk mengelola kompleksitas pengembangan teknologi informasi

   b. Perspektif Praktis

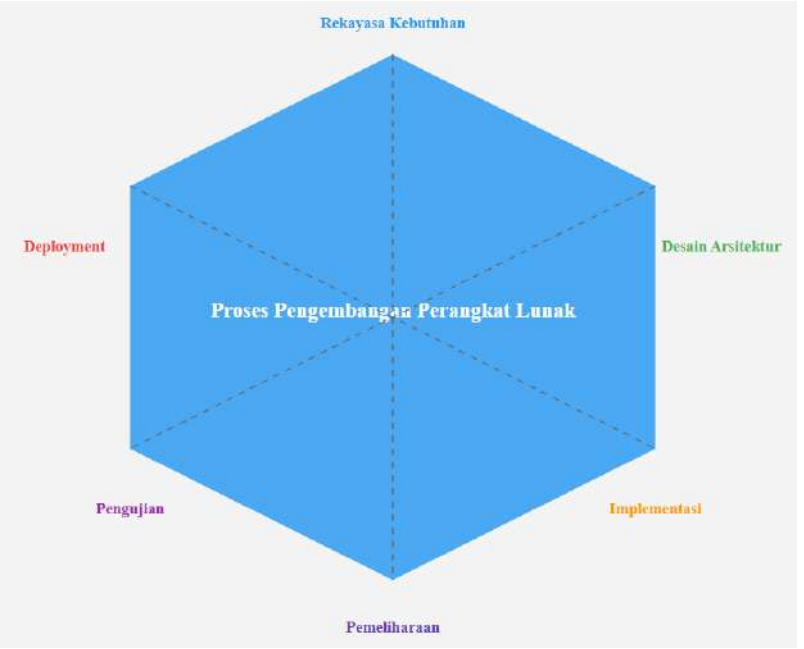
      Boehm dan Turner (2004) menjelaskan bahwa proses pengembangan perangkat lunak meliputi:

      1) Koordinasi aktivitas manusia

2) Pengelolaan sumber daya teknologi
3) Manajemen kompleksitas teknis
4) Pemenuhan kebutuhan bisnis

2.2 Komponen Utama Proses Pengembangan Perangkat Lunak

Proses pengembangan perangkat lunak terdiri dari sejumlah komponen kunci yang saling terkait dan memiliki peran strategis dalam menghasilkan produk perangkat lunak berkualitas. Menurut Sommerville (2015), komponen-komponen ini membentuk kerangka kerja sistematis yang mengarahkan tim pengembang dari tahap konseptualisasi hingga implementasi dan pemeliharaan.



Gambar 2.1. Komponen Utama Proses Perangkat Lunak (Sumber : Sommerville, 2015)

1. Rekayasa Kebutuhan (*Requirements Engineering*)
    a. Definisi dan Ruang Lingkup
       Rekayasa kebutuhan merupakan proses sistematis untuk mengidentifikasi, menganalisis, mendokumentasikan, dan memvalidasi kebutuhan sistem perangkat lunak (Wiegers & Beatty, 2013).
    b. Aktivitas Kunci
        1) Elisitasi Kebutuhan
            a) Wawancara dengan pemangku kepentingan
            b) Observasi proses bisnis
            c) Analisis dokumen eksisting
        2) Dokumentasi Persyaratan
            a) Pembuatan Dokumen Spesifikasi Persyaratan Perangkat Lunak (SRS)
            b) Pendefinisian kebutuhan fungsional dan non-fungsional
            c) Pemodelan use case
        3) Validasi Kebutuhan
            a) Peninjauan dengan pemangku kepentingan
            b) Prototyping
            c) Analisis kelayakan

2. Desain Arsitektur dan Sistem
    a. Konsep Fundamental
       Desain arsitektur merupakan tahap perancangan struktur konseptual dan teknis perangkat lunak (Bass et al., 2012). Fokus utamanya adalah menciptakan kerangka yang kokoh dan skalabel.
    b. Komponen Desain
        1) Arsitektur Perangkat Lunak
            a) Pemilihan pola arsitektur (misalnya, microservices, monolitik)

b) Perancangan komponen sistem
c) Definisi antarmuka antarmodul

2) Desain Antarmuka Pengguna (UI/UX)
a) Perancangan alur interaksi pengguna
b) Desain wireframe
c) Prototipe antarmuka

3) Desain Basis Data
a) Pemodelan struktur data
b) Perancangan skema basis data
c) Normalisasi dan optimasi

3. Implementasi dan Pengkodean
a. Karakteristik Proses
Implementasi adalah tahap konversi desain menjadi kode program yang dapat dieksekusi (Martin, 2009). Melibatkan penerjemahan spesifikasi teknis ke dalam bahasa pemrograman.
b. Praktik Pengkodean
1) Standar Pengkodean
a) Penggunaan konvensi penamaan
b) Konsistensi format kode
c) Dokumentasi kode internal
2) Teknik Pengembangan
a) Pemrograman berorientasi objek
b) Modularitas
c) Penggunaan kerangka kerja dan pustaka
3) Manajemen Konfigurasi
a) Kontrol versi (Git)
b) Manajemen cabang (*branching*)
c) Integrasi berkelanjutan

4. Pengujian Perangkat Lunak
    a. Pendekatan Komprehensif
       Pengujian adalah proses sistematis untuk mengevaluasi kualitas, fungsionalitas, dan performa perangkat lunak (Myers et al., 2011).
    b. Tingkat Pengujian
        1) Pengujian Unit
            a) Verifikasi komponen individual
            b) Pengujian fungsi dan metode
            c) Cakupan kode
        2) Pengujian Integrasi
            a) Pengujian interaksi antarmodul
            b) Verifikasi aliran data
            c) Pengujian antarmuka
        3) Pengujian Sistem
            a) Evaluasi sistem secara keseluruhan
            b) Pengujian skenario end-to-end
            c) Validasi persyarat
        4) Pengujian Penerimaan Pengguna
            a) Validasi dengan pengguna akhir
            b) Pengujian skenario nyata
            c) Konfirmasi kesesuaian kebutuhan

5. Deployment dan Implementasi
    a. Strategi Implementasi
       Deployment mencakup proses instalasi, konfigurasi, dan persiapan perangkat lunak untuk penggunaan produksi (Humble & Farley, 2010).
    b. Aktivitas Kunci
        1) Persiapan Lingkungan
            a) Konfigurasi infrastruktur
            b) Migrasi data

c) Instalasi prasyarat sistem

2) Strategi Rilis
   a) Rilis bertahap
   b) Canary deployment
   c) Blue-green deployment
3) Dokumentasi Pengguna
   a) Panduan penggunaan
   b) Dokumentasi teknis
   c) Materi pelatihan

6. Pemeliharaan dan Evolusi
   a. Konsep Fundamental
      Pemeliharaan adalah fase berkelanjutan untuk menjaga dan meningkatkan kualitas perangkat lunak setelah deployment (Pigoski, 1997).
   b. Aktivitas Pemeliharaan
      1) Perbaikan Bug
         a) Identifikasi dan koreksi kesalahan
         b) Manajemen tiket
         c) Prioritasi perbaikan
      2) Peningkatan Fungsional
         a) Penambahan fitur baru
         b) Optimasi performa
         c) Penyesuaian dengan kebutuhan bisnis
      3) Dukungan Teknis
         a) Manajemen insiden
         b) Pemecahan masalah
         c) Bimbingan pengguna

Komponen utama proses pengembangan perangkat lunak membentuk ekosistem kompleks yang membutuhkan pendekatan sistematis, kolaboratif, dan berkelanjutan. Keberhasilan

implementasi bergantung pada integrasi efektif antar komponennya.

2.3 Karakteristik Proses

Dalam pengembangan perangkat lunak, terdapat beberapa karakteristik penting dari proses pengembangan yang perlu dipahami:

1. Iteratif dan Bertahap

   Proses pengembangan perangkat lunak memiliki sifat iteratif yang kompleks. Menurut Sommerville (2015), ini berarti setiap tahapan dalam pengembangan tidak bersifat linier murni, melainkan dapat diulang dan disempurnakan secara berkelanjutan. Model-model seperti spiral, prototype, dan agile menggambarkan sifat iteratif ini dengan jelas.

   Karakteristik iteratif meliputi:

   a. Kemampuan untuk kembali ke tahap sebelumnya
   b. Perbaikan berkelanjutan
   c. Fleksibilitas dalam menghadapi perubahan kebutuhan
   d. Pengujian dan validasi berulang

2. Kompleksitas dan Dinamis

   Kompleksitas dalam pengembangan perangkat lunak mencakup beberapa dimensi:

   a. Kompleksitas Teknis
      1) Integrasi berbagai teknologi
      2) Arsitektur sistem yang rumit
      3) Kebutuhan akan algoritma dan struktur data canggih
   b. Kompleksitas Manajerial
      1) Koordinasi tim multidisiplin
      2) Manajemen sumber daya
      3) Pengelolaan waktu dan anggaran

c. Kompleksitas Manusia
    1) Komunikasi antaranggota tim
    2) Perbedaan keahlian dan pemahaman
    3) Dinamika kelompok dan kerja sama

Menurut Pressman (2014), setiap proyek perangkat lunak memiliki keunikan tersendiri yang memerlukan pendekatan khusus dan fleksibel.

3. Manajemen Risiko
Proses manajemen risiko dalam pengembangan perangkat lunak meliputi:
    a. Identifikasi Risiko
        1) Risiko teknis
        2) Risiko proyek
        3) Risiko bisnis
        4) Risiko eksternal
    b. Analisis Risiko
        1) Penilaian probabilitas terjadinya risiko
        2) Evaluasi dampak potensial
        3) Prioritisasi risiko
    c. Mitigasi Risiko
        1) Strategi pencegahan
        2) Rencana kontingensi
        3) Mekanisme respons cepat

Schwalbe (2016) menekankan bahwa manajemen risiko yang efektif dapat mengurangi kemungkinan kegagalan proyek hingga 60%.

4. Kolaborasi dan Komunikasi
Karakteristik kolaboratif dalam pengembangan perangkat lunak mencakup:

a. Komunikasi Internal
    1) Briefing rutin
    2) Dokumentasi bersama
    3) Alat kolaborasi digital
    4) Rapat koordinasi
b. Komunikasi Eksternal
    1) Interaksi dengan pemangku kepentingan
    2) Umpan balik pengguna
    3) Negosiasi persyaratan
c. Kerangka Kerja Kolaboratif
    1) Metode agile
    2) Scrum
    3) Extreme Programming (XP)

Hughes & Cotterell (2009) menyoroti bahwa kualitas komunikasi memiliki korelasi langsung dengan kesuksesan proyek perangkat lunak.

5. Berorientasi pada Kualitas
   Fokus pada kualitas meliputi:
   a. Tahapan Pengujian
       1) Unit testing
       2) Integrasi testing
       3) Sistem testing
       4) Penerimaan pengguna (UAT)
   b. Metrik Kualitas
       1) Keandalan
       2) Efisiensi
       3) Usabilitas
       4) Keamanan
   c. Standar Kualitas
       1) ISO/IEC 25010

2) CMMI (*Capability Maturity Model Integration*)
3) Six Sigma

Royce (2013) menggarisbawahi bahwa kualitas bukanlah tahapan akhir, melainkan proses berkelanjutan yang terintegrasi dalam setiap fase pengembangan.

2.4 Faktor Keberhasilan

1. Manajemen Proyek yang Efektif
   a. Perencanaan strategis yang komprehensif
   b. Penetapan tujuan dan milestone yang jelas
   c. Alokasi sumber daya yang tepat
   d. Penggunaan metodologi manajemen proyek yang sesuai

   Menurut Kerzner (2017), manajemen proyek yang efektif dapat meningkatkan probabilitas kesuksesan hingga 70% dalam pengembangan perangkat lunak.

2. Kompetensi Tim Pengembang
   a. Keterampilan teknis tinggi
   b. Kemampuan adaptasi teknologi
   c. Pemahaman mendalam tentang domain permasalahan
   d. Keterampilan komunikasi dan kolaborasi

   Sommerville (2015) menekankan bahwa kualitas sumber daya manusia merupakan faktor kunci dalam keberhasilan proyek perangkat lunak.

3. Komunikasi yang Transparan
   a. Dokumentasi yang jelas
   b. Mekanisme umpan balik berkelanjutan

c. Keterbukaan antara tim pengembang dan pemangku kepentingan
d. Manajemen ekspektasi

Hughes & Cotterell (2009) mengidentifikasi komunikasi sebagai pondasi utama keberhasilan proyek perangkat lunak.

4. Pemahaman Kebutuhan Pengguna
   a. Analisis persyaratan yang komprehensif
   b. Keterlibatan pengguna dalam proses desain
   c. Pengujian berkelanjutan
   d. Fleksibilitas terhadap perubahan kebutuhan

Pressman (2014) menyatakan bahwa kesalahan dalam memahami kebutuhan pengguna dapat menyebabkan kegagalan proyek hingga 50%.

5. Manajemen Risiko
   a. Identifikasi risiko dini
   b. Strategi mitigasi yang terencana
   c. Pemantauan dan evaluasi berkelanjutan
   d. Rencana kontinjensi

2.5 Tantangan

1. Kompleksitas Teknologi
   a. Keragaman teknologi dan platform
   b. Kecepatan perubahan teknologi
   c. Integrasi sistem yang rumit
   d. Kebutuhan akan pembaruan berkelanjutan

2. Keterbatasan Sumber Daya
    a. Kendala anggaran
    b. Keterbatasan tenaga ahli
    c. Infrastruktur yang tidak memadai
    d. Kompetisi sumber daya internal

3. Dinamika Perubahan Kebutuhan
    a. Perubahan persyaratan yang cepat
    b. Kesulitan dalam pelacakan perubahan
    c. Biaya adaptasi yang tinggi
    d. Kompleksitas manajemen lingkup proyek

4. Faktor Manusia
    a. Perbedaan persepsi
    b. Resistensi terhadap perubahan
    c. Konflik antaranggota tim
    d. Masalah motivasi dan produktivitas

5. Kendala Eksternal
    a. Regulasi yang ketat
    b. Pembatasan teknologi
    c. Kondisi pasar yang tidak stabil
    d. Kompetisi global

2.6 Strategi Mengatasi Tantangan

1. Penggunaan Metodologi Modern
    a. Agile
    b. Scrum
    c. DevOps
    d. *Continuous Integration/Continuous Deployment* (CI/CD)

2. Investasi Pengembangan SDM
    a. Pelatihan berkelanjutan
    b. Sertifikasi profesional
    c. Program pengembangan keahlian
    d. Kultur learning organization

3. Penerapan Teknologi Pendukung
    a. Alat manajemen proyek
    b. Sistem kontrol versi
    c. Platform kolaborasi
    d. Teknologi cloud

DAFTAR PUSTAKA

Boehm, B. & Turner, R., 2004. Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Professional.

IEEE Computer Society, 2014. Guide to the Software Engineering Body of Knowledge. IEEE Computer Society Press.

Pressman, R.S. & Maxim, B.R., 2020. Software Engineering: A Practitioner's Approach. McGraw-Hill Education.

Sommerville, I., 2015. Software Engineering. 10th ed. Pearson Education Limited.

Bass, L., Clements, P. & Kazman, R., 2012. Software Architecture in Practice. Addison-Wesley Professional.

Humble, J. & Farley, D., 2010. Continuous Delivery. Pearson Education.

Martin, R.C., 2009. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

Myers, G.J., Sandler, C. & Badgett, T., 2011. The Art of Software Testing. John Wiley & Sons.

Pigoski, T.M., 1997. Practical Software Maintenance. John Wiley & Sons.

Wiegers, K.E. & Beatty, J., 2013. Software Requirements. Microsoft Press.

Royce (2013) menggarisbawahi bahwa kualitas bukanlah tahapan akhir, melainkan proses berkelanjutan yang terintegrasi dalam setiap fase pengembangan.

Hughes, B. & Cotterell, M., 2009. Software Project Management, 5th Edition. McGraw-Hill Education.

Pressman, R.S., 2014. Software Engineering: A Practitioner's Approach. 8th ed. McGraw-Hill Education.

Schwalbe, K., 2016. Information Technology Project Management. 8th ed. Cengage Learning.\

Hughes, B. & Cotterell, M., 2009. Software Project Management

Kerzner, H., 2017. Project Management: A Systems Approach

BAB 3
METODE PENGEMBANGAN PERANGKAT LUNAK

3.1 Pendahuluan

Pengembangan perangkat lunak adalah proses yang kompleks yang melibatkan berbagai tahapan, mulai dari perencanaan, analisis kebutuhan, desain, implementasi, hingga pemeliharaan (Budake, Bhoite and Kharade, 2023; Kepkep, Durmusoglu and Dereli, 2024) . Setiap tahap memerlukan pendekatan yang terstruktur dan metode yang tepat untuk mencapai hasil yang optimal. Metode pengembangan perangkat lunak bertujuan untuk memberikan pedoman yang jelas, memastikan kualitas perangkat lunak, serta meminimalkan risiko kegagalan dalam proyek.

Metode yang digunakan dalam pengembangan perangkat lunak sangat bergantung pada berbagai faktor, termasuk skala proyek, tingkat kompleksitas, anggaran, dan waktu yang tersedia (Hanafiah and Kasirun, 2007; Klünder *et al.*, 2020). Oleh karena itu, pemilihan metode yang tepat menjadi krusial untuk keberhasilan proyek. Dalam bab ini, kita akan membahas berbagai metode pengembangan perangkat lunak yang sering digunakan, mulai dari model tradisional hingga pendekatan modern yang lebih fleksibel dan adaptif.

3.2 Model Air Terjun (*Waterfall Model*)

Waterfall adalah salah satu metode pengembangan perangkat lunak yang paling awal dan paling terkenal. Model ini mengasumsikan bahwa pengembangan perangkat lunak dilakukan dalam urutan linier yang terstruktur, dengan setiap tahap harus

diselesaikan sepenuhnya sebelum melanjutkan ke tahap berikutnya. Secara umum, model ini mencakup beberapa tahap, yaitu:

1. Analisis Kebutuhan: Pada tahap ini, seluruh kebutuhan sistem dikumpulkan dan dianalisis dengan rinci.
2. Desain Sistem: Setelah kebutuhan dikumpulkan, desain perangkat lunak yang memenuhi spesifikasi tersebut dibuat.
3. Pengkodean: Tahap di mana desain perangkat lunak diterjemahkan menjadi kode yang dapat dijalankan.
4. Pengujian: Setelah perangkat lunak selesai dibangun, pengujian dilakukan untuk memastikan bahwa perangkat lunak berfungsi sesuai dengan kebutuhan.
5. Pemeliharaan: Setelah perangkat lunak digunakan, pemeliharaan dilakukan untuk memperbaiki masalah atau memperbarui sistem.

Keuntungan Model Waterfall

1. Struktur yang jelas: Setiap tahapan dalam proses pengembangan perangkat lunak ditetapkan dengan jelas, memudahkan manajemen proyek.
2. Dokumentasi yang lengkap: Model ini menuntut dokumentasi yang komprehensif, yang berguna untuk referensi di masa depan.
3. Mudah diimplementasikan pada proyek kecil: Karena sifatnya yang terstruktur, model ini sering digunakan dalam proyek dengan skala kecil dan jelas.

Kelemahan Model Waterfall

1. Kurang fleksibel: Setelah satu tahap selesai, kembali ke tahap sebelumnya bisa sangat sulit dan mahal. Hal ini membuat model ini kurang cocok untuk proyek dengan kebutuhan yang bisa berubah.

2. Tidak cocok untuk proyek besar dan kompleks: Pada proyek yang lebih besar dan lebih kompleks, model Waterfall sering kali tidak mampu menangani perubahan yang diperlukan sepanjang siklus hidup proyek.

Tabel 3.1. Proses Waterfall

Tahap	Deskripsi
Analisis Kebutuhan	Mengumpulkan dan menganalisis kebutuhan sistem secara rinci.
Desain Sistem	Merancang sistem berdasarkan analisis kebutuhan yang telah dilakukan.
Pengkodean	Menerjemahkan desain menjadi kode perangkat lunak.
Pengujian	Memastikan bahwa sistem berfungsi dengan benar sesuai dengan spesifikasi.
Pemeliharaan	Melakukan pembaruan atau perbaikan setelah perangkat lunak digunakan.

Dalam tabel di atas, Kita dapat melihat bagaimana proses pengembangan perangkat lunak dalam model Waterfall berjalan secara linier. Setiap tahap harus diselesaikan sebelum bergerak ke tahap berikutnya.

3.3 Model Iteratif dan Inkremental (*Iterative and Incremental Model*)

Model Iteratif dan Inkremental adalah pendekatan pengembangan perangkat lunak yang lebih fleksibel dibandingkan dengan model Waterfall. Dalam model ini, pengembangan perangkat lunak dilakukan dalam siklus berulang (iterasi), dengan setiap iterasi menghasilkan versi perangkat lunak yang lebih lengkap dan berkembang seiring waktu.

1. **Model Iteratif**: Dalam model ini, proses pengembangan dilakukan dalam beberapa putaran (*iteration*). Setiap iterasi melibatkan analisis, desain, pengkodean, dan pengujian. Setiap siklus iterasi bertujuan untuk meningkatkan dan memperbaiki perangkat lunak berdasarkan umpan balik dari iterasi sebelumnya.
2. **Model Inkremental**: Dalam model Inkremental, perangkat lunak dibangun secara bertahap dengan menambahkan fungsionalitas baru pada setiap inkrementasi. Pada setiap inkrementasi, perangkat lunak yang sudah ada akan diperluas untuk mencakup fitur tambahan yang diinginkan. Hal ini memungkinkan perangkat lunak dapat di-*release* lebih cepat dalam versi yang lebih sederhana namun fungsional.

Keuntungan Model Iteratif dan Inkremental

1. Fleksibilitas Tinggi: Proyek dapat menyesuaikan perubahan persyaratan dengan mudah, karena perangkat lunak dikembangkan dalam siklus yang lebih pendek.
2. Waktu Pengembangan Lebih Cepat: Setiap iterasi menghasilkan bagian perangkat lunak yang bisa digunakan, memungkinkan pengujian lebih cepat dan *release* lebih awal.
3. Umpan Balik yang Cepat: Dengan iterasi yang berkelanjutan, pengembang dapat memperoleh umpan balik dari pengguna lebih cepat, sehingga mereka dapat membuat perbaikan lebih cepat.
4. Risiko yang Lebih Rendah: Proyek dapat diidentifikasi dan ditangani lebih cepat jika ada masalah atau kegagalan, karena perangkat lunak diuji dan dievaluasi pada setiap iterasi.

Kelemahan Model Iteratif dan Inkremental

1. Pengelolaan yang Lebih Rumit: Karena prosesnya berulang, pengelolaan proyek bisa menjadi lebih kompleks, terutama jika tidak ada perencanaan yang jelas di awal.
2. Kemungkinan Lebih Mahal: Meskipun perangkat lunak dapat diselesaikan lebih cepat, biaya pengembangan dapat meningkat seiring berjalannya waktu karena siklus pengembangan yang terus menerus.
3. Dokumentasi yang Terbatas: Dengan fokus pada pengembangan yang cepat, dokumentasi proyek mungkin tidak selalu mendalam atau komprehensif, yang bisa menjadi masalah di masa depan.

Contoh Implementasi Model Iteratif dan Inkremental

Model ini sering diterapkan dalam proyek perangkat lunak yang membutuhkan pengembangan cepat dan perbaikan berkelanjutan, seperti aplikasi web atau aplikasi perangkat mobile. Misalnya, sebuah perusahaan perangkat lunak yang mengembangkan aplikasi e-commerce bisa mulai dengan fitur dasar seperti pengelolaan produk dan pembayaran, dan kemudian menambahkan fitur lain seperti ulasan pengguna atau pengelolaan stok dalam inkrementasi berikutnya.

Tabel 3.2. Proses Iteratif dan Inkremental

Iterasi	**Deskripsi**
Iterasi 1	Pengembangan dan pengujian fungsi dasar (misalnya, pendaftaran dan login pengguna).
Iterasi 2	Menambahkan fitur baru berdasarkan umpan balik (misalnya, pembayaran dan pengelolaan produk).
Iterasi 3	Menambahkan fitur lanjutan (misalnya, pengelolaan stok atau sistem rekomendasi produk).
Iterasi 4	Pengujian lebih lanjut dan perbaikan berdasarkan umpan balik.

Dalam grafik ini, Kita dapat melihat bagaimana setiap iterasi memperkenalkan fitur baru atau meningkatkan fungsionalitas perangkat lunak yang ada.

3.4 Model Spiral (*Spiral Model*)

Model Spiral, yang dikembangkan oleh Barry Boehm pada tahun 1986, merupakan salah satu metode pengembangan perangkat lunak yang menggabungkan elemen-elemen dari model Waterfall dan model Iteratif (Nilsson and Wilson, 2012). Model ini berfokus pada pengelolaan risiko melalui pendekatan siklik (spiral), di mana setiap siklus melibatkan perencanaan, desain, pengkodean, pengujian, dan evaluasi risiko (Evans, Evans and Fabbiano, 2008).
Siklus spiral terdiri dari empat tahap utama:

1. Perencanaan dan Penentuan Tujuan: Tahap pertama ini melibatkan perencanaan secara rinci, termasuk identifikasi tujuan proyek, penentuan kebutuhan, dan pengaturan anggaran.
2. Identifikasi dan Pengelolaan Risiko: Di sini, risiko proyek diidentifikasi dan dianalisis untuk memastikan bahwa pengembang dapat memitigasi masalah yang mungkin terjadi.
3. Pengembangan dan Pengujian: Dalam fase ini, perangkat lunak dikembangkan dan diuji. Hasil pengujian digunakan untuk menginformasikan siklus berikutnya.
4. Evaluasi Pelanggan dan Rencana Proyek: Pada tahap ini, perangkat lunak yang dikembangkan dievaluasi oleh pemangku kepentingan, dan perencanaan untuk siklus berikutnya dibuat.

Keuntungan Model Spiral

1. Pengelolaan Risiko yang Lebih Baik: Model ini sangat baik dalam mengelola risiko karena setiap siklus berfokus pada identifikasi dan mitigasi risiko.
2. Iterasi Berkelanjutan: Setiap siklus menghasilkan peningkatan dan pengujian perangkat lunak, memungkinkan proyek untuk berkembang secara lebih terkontrol.
3. Keterlibatan Pemangku Kepentingan: Model ini melibatkan pemangku kepentingan secara aktif dalam setiap tahap, yang meningkatkan kemungkinan perangkat lunak yang dibangun memenuhi kebutuhan pengguna.

Kelemahan Model Spiral

1. Biaya yang Lebih Tinggi: Model Spiral membutuhkan waktu dan biaya yang lebih banyak dibandingkan model lain, terutama karena berfokus pada pengujian berulang dan manajemen risiko.
2. Kompleksitas Pengelolaan: Pengelolaan proyek menggunakan model Spiral bisa menjadi sangat kompleks, terutama pada proyek besar dengan banyak pemangku kepentingan.
3. Kebutuhan akan Keahlian Khusus: Agar model ini dapat diimplementasikan dengan baik, pengembang dan manajer proyek perlu memiliki pemahaman yang mendalam tentang risiko dan teknik mitigasi.

Contoh Implementasi Model Spiral

Model Spiral sering digunakan dalam proyek perangkat lunak yang kompleks dan berisiko tinggi, seperti pengembangan perangkat lunak militer, sistem avionik, atau aplikasi yang melibatkan data sensitif. Misalnya, pengembangan sistem

pengendalian lalu lintas udara dapat menggunakan model Spiral untuk memitigasi risiko sistem yang dapat mengancam keselamatan.

Tabel 3.3. Proses Model Spiral

Siklus	**Deskripsi**
Siklus 1	Perencanaan dan identifikasi risiko. Pengembangan dan pengujian versi awal perangkat lunak.
Siklus 2	Analisis lebih lanjut dan pengujian ulang. Penambahan fitur dan pengurangan risiko.
Siklus 3	Uji coba lebih mendalam dan perencanaan untuk perbaikan.
Siklus 4	Penyelesaian dan pengiriman perangkat lunak setelah evaluasi lebih lanjut.

3.5 *Agile Software Development*

Agile adalah pendekatan yang lebih fleksibel dan adaptif terhadap pengembangan perangkat lunak. Dibandingkan dengan metode *Waterfall* atau Spiral yang lebih terstruktur, *Agile* berfokus pada pengembangan perangkat lunak melalui kolaborasi antara tim pengembang dan pelanggan, serta iterasi yang lebih cepat dan perubahan yang lebih mudah.

Ada beberapa prinsip dasar yang menjadi landasan dalam Agile:

1. Kolaborasi lebih diutamakan daripada negosiasi kontrak.
2. Menanggapi perubahan lebih penting daripada mengikuti rencana yang sudah ditentukan.
3. Perangkat lunak yang berfungsi adalah ukuran utama kemajuan.
4. Tim yang terorganisir dengan baik lebih penting daripada proses yang kaku.

Keuntungan Agile:

1. Fleksibilitas dalam Menghadapi Perubahan: Perubahan persyaratan dapat diterima kapan saja selama siklus pengembangan.
2. Penyelesaian Cepat: Setiap iterasi menghasilkan perangkat lunak yang dapat digunakan, memungkinkan peluncuran lebih cepat.
3. Kolaborasi yang Lebih Baik: Agile memprioritaskan komunikasi langsung dengan pelanggan, yang memastikan bahwa produk akhir lebih memenuhi kebutuhan pengguna.

Kelemahan Agile:

1. Tantangan dalam Pengelolaan Proyek: Tanpa perencanaan yang baik, proyek Agile bisa berisiko berlarut-larut atau kehilangan fokus.
2. Kesulitan dalam Skalabilitas: Meskipun Agile bekerja dengan baik dalam proyek kecil hingga menengah, pengelolaan proyek besar bisa menjadi tantangan.

Metode dalam Agile

Beberapa metode utama dalam Agile adalah:

1. Scrum: Fokus pada manajemen proyek dengan pembagian tugas dalam sprint.
2. Kanban: Fokus pada aliran kerja yang terkontrol dan tugas yang dikerjakan secara berkelanjutan.
3. *Extreme Programming* (XP): Menekankan pengembangan perangkat lunak melalui teknik-teknik pengkodean yang ketat dan kolaborasi yang intens.

3.6 DevOps

DevOps adalah pendekatan dalam pengembangan perangkat lunak yang mengintegrasikan pengembangan perangkat lunak (*development*) dan operasi (*operations*) untuk meningkatkan kolaborasi dan produktivitas tim. Tujuan utama dari DevOps adalah untuk mempercepat siklus hidup perangkat lunak, dari pengembangan hingga produksi, dengan meningkatkan otomatisasi, pengujian, dan pemantauan.

DevOps bertujuan untuk memecahkan hambatan tradisional antara pengembang dan tim operasi. Dengan menggunakan prinsip-prinsip seperti *Continuous Integration* (CI) dan *Continuous Delivery* (CD), DevOps memungkinkan tim untuk merilis perangkat lunak dengan lebih cepat dan lebih sering.

Prinsip Utama DevOps

1. Kolaborasi dan Komunikasi: DevOps memfokuskan pada kerja sama antara tim pengembangan dan operasi untuk meningkatkan proses pengembangan dan implementasi.
2. Otomatisasi: Banyak proses yang diotomatisasi, termasuk pengujian perangkat lunak, pengelolaan konfigurasi, dan penerapan perangkat lunak.
3. Penerapan yang Cepat dan Terus-Menerus: DevOps mengadopsi prinsip *Continuous Integration dan Continuous Delivery* (CI/CD) untuk memungkinkan pengembangan dan penerapan perangkat lunak secara berkelanjutan.
4. Pemantauan dan Pengujian yang Berkelanjutan: Pemantauan terus-menerus terhadap aplikasi yang telah diterapkan dan pengujian otomatis membantu memastikan kualitas dan stabilitas perangkat lunak.

Keuntungan DevOps

1. Peningkatan Kecepatan Pengembangan: Dengan proses otomatisasi dan integrasi yang berkelanjutan, DevOps memungkinkan perangkat lunak dirilis lebih cepat ke pasar.
2. Peningkatan Kualitas dan Stabilitas: Pengujian otomatis dan pemantauan real-time meningkatkan kualitas perangkat lunak, serta mengurangi risiko kegagalan.
3. Pengurangan Biaya dan Risiko: Dengan mengidentifikasi masalah lebih awal dalam siklus hidup perangkat lunak, DevOps membantu mengurangi biaya dan risiko yang terlibat.
4. Kolaborasi yang Lebih Baik: Tim pengembang dan operasi bekerja lebih dekat, menghilangkan kesenjangan antara keduanya, yang dapat mempercepat proses pengembangan.

Kelemahan DevOps

1. Kebutuhan Infrastruktur yang Mahal: Implementasi DevOps memerlukan alat dan infrastruktur yang tepat, yang bisa menjadi mahal, terutama untuk organisasi kecil atau menengah.
2. Perubahan Budaya yang Diperlukan: Perubahan budaya dalam organisasi sangat penting untuk DevOps sukses. Tim yang terbiasa bekerja secara terpisah mungkin merasa kesulitan dalam berkolaborasi secara erat.
3. Kompleksitas dalam Implementasi: Mengintegrasikan DevOps dengan metode atau teknologi yang sudah ada dapat menjadi tantangan bagi tim yang belum terbiasa dengan praktik ini.

Implementasi DevOps

DevOps mengadopsi beberapa alat dan praktik yang memungkinkan otomatisasi dan peningkatan kolaborasi antara pengembang dan tim operasi. Beberapa alat yang sering digunakan dalam DevOps meliputi:

1. Jenkins: Untuk otomatisasi CI/CD.

2. Docker: Untuk pengelolaan kontainer dan penyebaran aplikasi.
3. Kubernetes: Untuk orkestrasi kontainer.
4. Ansible/Chef/Puppet: Untuk manajemen konfigurasi dan otomatisasi infrastruktur.
5. Nagios: Untuk pemantauan infrastruktur dan aplikasi.

Contoh Implementasi DevOps

Contoh implementasi DevOps dapat ditemukan dalam banyak perusahaan teknologi besar. Misalnya, perusahaan e-commerce seperti Amazon menggunakan DevOps untuk memastikan bahwa setiap perubahan kode yang dilakukan oleh pengembang dapat diuji dan diterapkan dengan cepat, memungkinkan pengiriman fitur baru kepada pelanggan dalam waktu singkat.

Tabel 3.4. Proses DevOps

Tahap DevOps	**Deskripsi**
Pengembangan (*Development*)	Pengembang menulis kode dan mengembangkan fitur baru.
Integrasi Berkelanjutan (*Continuous Integration*)	Kode secara otomatis digabungkan dan diuji secara terus-menerus.
Pengujian (*Testing*)	Pengujian otomatis dilakukan untuk memeriksa fungsionalitas dan kualitas kode.
Pengiriman Berkelanjutan (*Continuous Delivery*)	Kode yang sudah teruji siap untuk diterapkan ke produksi secara terus-menerus.
Penerapan dan Operasi (*Deployment and Operations*)	Kode diterapkan ke lingkungan produksi dan dioperasikan secara berkelanjutan.
Pemantauan (*Monitoring*)	Aplikasi dipantau untuk mendeteksi masalah dan kinerja yang buruk.

3.7 *Lean Software Development*

Lean Software Development (LSD) adalah pendekatan yang terinspirasi oleh prinsip-prinsip Lean Manufacturing yang diterapkan untuk pengembangan perangkat lunak (Mehta, Anderson and Raffo, 2008; Swaminathan and Jain, 2012; Janes, 2015; Kaur, 2015). Tujuan utamanya adalah untuk mengurangi pemborosan, meningkatkan efisiensi, dan memberikan nilai lebih besar kepada pelanggan dalam waktu yang lebih singkat.

LSD mengidentifikasi dan menghilangkan pemborosan yang ada dalam setiap tahap pengembangan perangkat lunak, mulai dari perencanaan, pengkodean, pengujian, hingga penerapan (Fagarasan *et al.*, 2023).

Prinsip utama lean software development yaitu:

1. Menghilangkan Pemborosan: Setiap aspek dalam pengembangan perangkat lunak yang tidak memberikan nilai kepada pelanggan dianggap sebagai pemborosan dan harus dihilangkan.
2. Meningkatkan Kualitas: Kualitas perangkat lunak harus menjadi prioritas utama, dengan fokus pada pengujian, pemeliharaan, dan umpan balik berkelanjutan.
3. Menghargai Tim: Tim pengembang harus memiliki otonomi dan wewenang untuk membuat keputusan, serta diberdayakan untuk memperbaiki proses kerja.
4. Iterasi Cepat: Lean mendorong iterasi yang cepat untuk mengurangi waktu tunggu dan memastikan produk dikirim lebih cepat kepada pelanggan.

Keuntungan Lean Software Development:

1. Efisiensi yang Lebih Baik: Proyek dapat diselesaikan lebih cepat dengan mengurangi pemborosan dalam setiap tahap.

2. Pengiriman Nilai yang Cepat: Dengan fokus pada penyampaian nilai pelanggan yang cepat, produk dapat digunakan lebih cepat oleh pelanggan.
3. Fleksibilitas yang Lebih Tinggi: LSD memungkinkan perubahan yang lebih cepat karena menghilangkan banyak hambatan dalam proses pengembangan.

Kelemahan Lean Software Development:

1. Kesulitan dalam Skalabilitas: Pendekatan Lean dapat sulit diterapkan dalam proyek besar yang melibatkan banyak tim.
2. Kurangnya Fokus pada Dokumentasi: Meskipun efisiensi adalah fokus utama, terkadang dokumentasi yang minimal bisa menjadi kendala jika proyek memerlukan pemeliharaan yang lebih lama.

3.8 *Extreme Programming* (XP)

Extreme Programming (XP) adalah sebuah metode pengembangan perangkat lunak yang berfokus pada pengiriman perangkat lunak berkualitas tinggi dengan berfokus pada pengujian berkelanjutan, komunikasi yang terbuka, serta adaptasi cepat terhadap perubahan. XP merupakan salah satu pendekatan dalam Agile Software Development yang sangat mendalami praktik pengkodean yang efisien dan efektif.

Metode ini memanfaatkan prinsip customer-centricity, yakni pelanggan atau pengguna akhir memiliki peran penting dalam setiap fase pengembangan perangkat lunak.

Prinsip Utama Extreme Programming (XP):

1. Perencanaan Berkelanjutan: XP mendukung perencanaan yang terus berkembang seiring dengan perkembangan proyek.
2. Pasangan Pemrograman (*Pair Programming*): Dua pengembang bekerja bersama pada satu komputer untuk menulis kode, yang

meningkatkan kualitas kode dan memungkinkan transfer pengetahuan antara anggota tim.
3. Pengujian Otomatis: XP mengutamakan pengujian otomatis untuk meminimalkan bug dan kesalahan perangkat lunak.
4. Refaktorisasi Berkelanjutan: Kode yang sudah ada diperbaiki dan disempurnakan secara berkala tanpa menambah fitur baru.
5. Keterlibatan Pengguna: Pengguna terlibat aktif dalam proses pengembangan untuk memastikan bahwa produk akhir sesuai dengan harapan mereka.
6. *Simplicity* (Kesederhanaan): XP menekankan pentingnya kode yang sederhana dan mudah dipahami, serta menghindari fitur yang tidak perlu.

Keuntungan *Extreme Programming* (XP)

1. Kualitas Perangkat Lunak yang Lebih Tinggi: Dengan pengujian berkelanjutan dan refaktorisasi kode yang konsisten, kualitas perangkat lunak dapat terjaga dengan baik.
2. Respon yang Cepat terhadap Perubahan: XP mendukung perubahan cepat berdasarkan umpan balik dari pengguna dan pengujian.
3. Peningkatan Kolaborasi Tim: Dengan penerapan pasangan pemrograman, anggota tim dapat saling berbagi pengetahuan, meningkatkan keterampilan teknis, dan mengurangi kesalahan.

Kelemahan *Extreme Programming* (XP):

1. Memerlukan Komitmen Waktu yang Tinggi: Implementasi XP membutuhkan keterlibatan yang mendalam dari semua anggota tim, termasuk pelanggan, yang dapat mengakibatkan beban kerja yang tinggi.
2. Kurangnya Dokumentasi Formal: Fokus pada pengkodean dan pengujian berarti XP seringkali tidak menghasilkan dokumentasi yang lengkap.

3. Tantangan dalam Proyek Skala Besar: XP lebih cocok untuk proyek kecil atau tim dengan sedikit anggota, sehingga implementasi pada proyek besar mungkin lebih sulit.

3.9 *Feature-Driven Development* (FDD)

Feature-Driven Development (FDD) adalah metodologi pengembangan perangkat lunak yang berfokus pada pengembangan perangkat lunak melalui fitur-fitur yang dapat dipahami dan diukur(Bauer, 2004; Rychlý and Tichá, 2008). FDD berfokus pada pembuatan dan pengiriman fitur-fitur terperinci dalam jangka waktu yang singkat.

Metode ini sangat berguna dalam pengembangan perangkat lunak skala besar, karena setiap fitur dikembangkan dalam waktu yang terukur dan dilengkapi dengan dokumentasi yang jelas (Aftab *et al.*, 2019; Merzouk, Marzak and Sael, 2022).

Prinsip Utama Feature-Driven Development (FDD)

1. Pembuatan Daftar Fitur: FDD dimulai dengan pembuatan daftar fitur yang akan dikembangkan, yang dikelompokkan berdasarkan prioritas.
2. Desain untuk Setiap Fitur: Setiap fitur dirancang secara rinci sebelum pengkodean dimulai.
3. Pengembangan Fitur: Setiap fitur dikerjakan dalam waktu yang telah ditentukan (biasanya dua minggu atau lebih pendek) oleh tim pengembang.
4. Pemantauan Kinerja: Kemajuan setiap fitur dipantau secara ketat untuk memastikan bahwa pengembangan perangkat lunak berjalan sesuai rencana.

Keuntungan *Feature-Driven Development* (FDD)

1. Pengiriman Fitur Secara Cepat: FDD memungkinkan tim untuk menyampaikan fitur secara berkala dalam jangka waktu yang

relatif singkat, sehingga pengguna dapat merasakan manfaatnya lebih cepat.
2. Keterlibatan Tim yang Lebih Tinggi: Setiap fitur membutuhkan keterlibatan seluruh tim, yang membantu meningkatkan koordinasi antar anggota.
3. Skalabilitas yang Lebih Baik: FDD cocok untuk proyek perangkat lunak yang besar karena memecah proyek menjadi beberapa fitur yang lebih kecil dan mudah diatur.

Kelemahan *Feature-Driven Development* (FDD)

1. Fokus yang Terlalu Ketat pada Fitur: Terkadang, fokus yang terlalu kuat pada fitur dapat mengabaikan integrasi keseluruhan sistem.
2. Kesulitan dalam Menangani Perubahan: Jika fitur yang dikembangkan tidak sesuai dengan harapan pengguna, perubahannya bisa sangat rumit untuk diterapkan pada model FDD.

Contoh Implementasi Feature-Driven Development

FDD sering digunakan dalam perusahaan perangkat lunak besar yang memiliki tim pengembang yang tersebar, seperti pada pengembangan perangkat lunak perbankan atau aplikasi e-commerce besar yang memerlukan pengelolaan fitur yang efisien dan berkala.

3.10 Kesimpulan

Dalam bab ini, kita telah membahas berbagai metode pengembangan perangkat lunak yang digunakan oleh para praktisi untuk mengatasi tantangan dalam pengembangan perangkat lunak modern. Setiap metode memiliki kekuatan dan kelemahannya masing-masing, dan sering kali pemilihan metode yang tepat

menjadi kunci keberhasilan suatu proyek pengembangan perangkat lunak.

Metode Waterfall, misalnya, menawarkan pendekatan yang sangat terstruktur dan mudah diikuti, yang cocok untuk proyek kecil dengan persyaratan yang jelas. Namun, metode ini kurang fleksibel dalam menghadapi perubahan kebutuhan selama siklus pengembangan. Sebaliknya, Agile dan turunannya, seperti Scrum, memberikan fleksibilitas dan kemampuan adaptasi yang tinggi, yang menjadikannya pilihan utama untuk proyek dengan lingkungan yang dinamis dan perubahan yang sering terjadi. Meskipun demikian, keberhasilan metode Agile bergantung pada kolaborasi tim yang kuat dan keterlibatan aktif dari pengguna akhir.

Model seperti Spiral dan Iteratif juga memiliki keunggulan dalam mengelola proyek yang kompleks dan berisiko tinggi, terutama ketika pengelolaan risiko menjadi perhatian utama. Model-model ini menggabungkan iterasi berulang dan umpan balik berkelanjutan untuk memastikan bahwa perangkat lunak yang dikembangkan memenuhi kebutuhan pengguna sekaligus meminimalkan risiko kegagalan.

DevOps telah menjadi pendekatan yang sangat relevan dalam pengembangan perangkat lunak modern, terutama dalam konteks integrasi dan pengiriman yang berkelanjutan dengan waktu siklus yang lebih pendek. DevOps memungkinkan tim untuk bekerja lebih cepat dan lebih efisien dengan memanfaatkan otomatisasi dan pemantauan berkelanjutan, yang memastikan bahwa perangkat lunak berkualitas tinggi dapat dirilis ke pasar dengan lebih cepat.

Di sisi lain, Lean Software Development membawa filosofi efisiensi yang diadaptasi dari dunia manufaktur, berfokus pada pengurangan pemborosan dan peningkatan nilai untuk pelanggan. Metode ini cocok untuk organisasi yang ingin meningkatkan efisiensi dan produktivitas tanpa mengorbankan kualitas. *Extreme Programming* (XP) dan *Feature-Driven Development* (FDD)

melengkapi berbagai pendekatan yang ada dengan menawarkan strategi pengembangan yang lebih berfokus pada kualitas kode, fitur yang spesifik, dan keterlibatan pelanggan secara mendalam.

Kesimpulannya, tidak ada satu metode yang dapat dianggap paling unggul atau cocok untuk semua jenis proyek. Pemilihan metode yang tepat harus mempertimbangkan berbagai faktor, seperti skala proyek, kompleksitas, sumber daya yang tersedia, serta kebutuhan pengguna dan pemangku kepentingan. Dengan memahami kekuatan dan kelemahan dari setiap metode, pengembang perangkat lunak dapat mengambil keputusan yang lebih baik untuk memastikan keberhasilan proyek mereka. Metode yang dipilih harus mampu mengakomodasi kebutuhan proyek sekaligus menghadapi tantangan yang mungkin muncul di sepanjang perjalanan pengembangan.

DAFTAR PUSTAKA

Aftab, S. *et al.* (2019) 'Using FDD for small project: An empirical case study', *International Journal of Advanced Computer Science and Applications*, 10(3), pp. 151–158. Available at: https://doi.org/10.14569/IJACSA.2019.0100319.

Bauer, M. (2004) 'FDD and project management', *Cutter IT Journal*, 17(6), pp. 38–43. Available at: https://www.scopus.com/inward/record.uri?eid=2-s2.0-3042661973&partnerID=40&md5=6067f14ad54006595a33605f34ddb4c1.

Budake, R., Bhoite, S. and Kharade, K. (2023) 'A Study of AI-based Techniques for Requirement Analysis in Software Engineering', in *AIP Conference Proceedings*. Available at: https://doi.org/10.1063/5.0178114.

Evans, J.D., Evans, I.N. and Fabbiano, G. (2008) 'The software development process at the chandra X-ray center', in *Proceedings of SPIE - The International Society for Optical Engineering*. Available at: https://doi.org/10.1117/12.790524.

Fagarasan, C. *et al.* (2023) 'Kanban in Software Development—The Role of Leadership and Metrics', in *Lecture Notes in Production Engineering*, pp. 369–384. Available at: https://doi.org/10.1007/978-3-031-15602-1_28.

Hanafiah, M. and Kasirun, Z.M. (2007) 'Using rule-based technique in developing the tool for finding suitable software methodology', *Malaysian Journal of Computer Science*, 20(2), pp. 209–224. Available at: https://doi.org/10.22452/mjcs.vol20no2.8.

Janes, A. (2015) 'A guide to lean software development in action', in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2015 -*

Proceedings. Available at: https://doi.org/10.1109/ICSTW.2015.7107412.

Kaur, D. (2015) 'Lean manufacturing to lean IT: An emerging quality assurance methodology in IT', in *Achieving Enterprise Agility through Innovative Software Development*, pp. 67–86. Available at: https://doi.org/10.4018/978-1-4666-8510-9.ch004.

Kepkep, T.B., Durmusoglu, A. and Dereli, T. (2024) 'Using Axiomatic Design and Fuzzy Axiomatic Design for Risk Management in Software Development', *IEEE Access* [Preprint]. Available at: https://doi.org/10.1109/ACCESS.2024.3501672.

Klünder, J. *et al.* (2020) 'Determining Context Factors for Hybrid Development Methods with Trained Models', in *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, pp. 65–66. Available at: https://doi.org/10.18420/SE2021_21.

Mehta, M., Anderson, D. and Raffo, D. (2008) 'Providing value to customers in software development through lean principles', in *Software Process Improvement and Practice*, pp. 101–109. Available at: https://doi.org/10.1002/spip.367.

Merzouk, S., Marzak, A. and Sael, N. (2022) 'The Proposal of a Process Flow Model and Metamodel for the Feature Driven Development Method', in *Lecture Notes in Networks and Systems*, pp. 485–494. Available at: https://doi.org/10.1007/978-3-031-02447-4_50.

Nilsson, A. and Wilson, T.L. (2012) 'Reflections on Barry W. Boehm's "A spiral model of software development and enhancement"', *International Journal of Managing Projects in Business*, 5(4), pp. 737–756. Available at: https://doi.org/10.1108/17538371211269031.

Rychlý, M. and Tichá, P. (2008) 'A tool for supporting feature-driven development', in *Lecture Notes in Computer Science*

(including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 196–207. Available at: https://doi.org/10.1007/978-3-540-85279-7_16.

Swaminathan, B. and Jain, K. (2012) 'Implementing the lean concepts of continuous improvement and flow on an agile software development project: An industrial case study', in *Proceedings - Agile India 2012, AgileIndia 2012*, pp. 10–19. Available at: https://doi.org/10.1109/AgileIndia.2012.12.

BAB 4
TIM PENGEMBANG PERANGKAT LUNAK

4.1 Pendahuluan

Pada bab ini menjelaskan terkait dengan pengembang perangkat lunak, Pengembangan perangkat lunak merupakan sekelompok aktivis ilmu komputer yang bekerja untuk proses pembuatan, pemeliharaan, perancangan pada perangkat lunak. Perangkat lunak sendiri merupakan program yang memberikan pemberitahuan ke komputer untuk terprogram,namun hal ini tidak bergantung pada perangkat keras.

Di dalam membentuk sebuah pengembang perangkat lunak dibutuhkan kerja sama tim pengelolan untuk menghasilkan karya yang lebih baik. Pengembangan perangkat lunak dibutuhkan tim untuk membangun proyek berupa produk teknis, setiap anggota dalam tim pengembangan perangkat lunak mempunyai tugas dan tanggung jawab atas pekerjaan mereka. Manajemen dalam suatu pekerjaan tidaklah mudah bagi banyak orang, maka yang perlu kita terapkan adalah mengawasi tim pengembangan perangkat lunak, maka yang diperlukan kita adalah sebagai berikut :

1. Penentuan Tujuan dan Harapan

   Dalam proyek sebuah perangkat lunak, kita harus mengetahui terlebih dahulu apa tujuan dan harapan kita sebagai pengembang perangkat lunak. Dalam sebuah pembuatan proyek pengembangan perangkat lunak, salah satunya adalah menjabarkan ekspektasi tujuan yang spesifik sehingga tim pengembang dapat memenuhi capaian, selain itu penentuan cara untuk mencapai apabila

tim akan lebih fokus untuk hasil yang berkualitas dan lebih baik, sehingga dapat bekerja secara cepat dan efisien.

2. Menentukan Pertemuan Secara Teratur dengan Anggota Tim
   Setiap anggota tim memiliki tugas dan tanggung jawab masing-masing, Untuk itu diperlukan pertemuan satu per satu secara teratur dengan anggota, untuk meninjau Kembali progres kemajuan tugas dan tanggung jawabnya, Selain untuk mengetahui informasi tentang kemajuan proyek, hal ini juga memberi anggota tim untuk berbagi pikiran atau kekhawatiran apa pun yang mereka miliki. Hal tersebut juga merupakan cara untuk lebih mengenal secara individu (kepribadian) dalam tim dan dapat merancang bagaimana dapat bekerja sama lebih baik.

3. Pembagian Tugas
   Pemberian tugas yang efektif merupakan sebuah kunci keberhasilan dalam proyek pengembangan perangkat lunak. Dengan pembagian tugas yang tepat, setiap anggota tim dapat berkontribusi secara optimal dan proyek dapat diselesaikan sesuai dengan target yang ditetapkan. Untuk melanjutkan Tugas maka yang pelu diperhatikan adalah :
   a. Sesuai dengan Keahlian, Setiap anggota tim memiliki keahlian yang berbeda. Maka yang harus diperhatikan adalah memastikan sesuai bidang keahlian anggota
   b. Jelas dan Terukur, Tugas harus dijelaskan secara jelas dan terukur dengan tujuan yang spesifik. Hindari memberikan tugas yang terlalu umum atau ambigu.
   c. Realitistis, Mempertimbangkan beban kerja setiap anggota tim dan memastikan tugas yang diberikan

dapat diselesaikan dalam waktu yang sudah ditentukan, artinya juga harus bijak dalam memanfaatkan waktu.

4. Pertemuan Antar Anggota

Pertemuan satu-lawan-satu (*one-on-one meeting*) secara teratur antara manajer atau pemimpin tim dengan setiap anggota tim merupakan praktik yang sangat berharga dalam dunia pengembangan perangkat lunak. Pertemuan ini bukan hanya sekadar formalitas, melainkan sarana yang efektif untuk membangun hubungan yang kuat, meningkatkan produktivitas, dan memastikan setiap anggota tim merasa dihargai dan didukung,

Hal ini dapat memberikan manfaat bagi anggota diantaranya adalah meningkatkan Komunikasi bagi anggota tim untuk berbagi ide, kekhawatiran, atau tantangan yang mereka hadapi secara terbuka, kemudian dapat memahami kebutuhan dan hambatan individu, sehingga manajer dapat memberikan dukungan yang tepat untuk meningkatkan produktivitas (Zakhralativa Ruskanda,2013)

Mengembangkan Karir dalam hal ini juga dapat menjadi ajang untuk membahas tujuan karir, memberikan *feedback*, dan merencanakan pengembangan skil. Kemudian membangun hubungan yang kuat antara manajer dan anggota tim akan menciptakan lingkungan kerja yang lebih positif dan kolaboratif.

5. Memanfaatkan Alat Kolaborasi dan Manajemen Proyek
   Alat kolaborasi dan manajemen proyek menjadi semakin penting dalam dunia pengembangan perangkat lunak yang semakin kompleks dan kolaboratif. Alat kolaborasi dalam tim menajemen proyek dapat Memudahkan berbagi informasi, ide, dan umpan balik secara *real-time*, mengotomatiskan tugas rutin dan melacak kemajuan proyek dengan lebih baik, meningkatkan transparansi: Semua anggota tim dapat melihat status proyek dan tugas yang sedang berjala, membantu mengidentifikasi dan mengatasi masalah lebih cepat (Migunani, Kom & Kom 2022).

   Dalam hal ini tim dapat memanfaatkan beberapa alat komunikasi perangkat lunak seperti *Trello, Slack, Microsoft Team*.

4.2 Tantangan Pengelolaan Tim Pengembang

Dalam menjalankan sebuah proyek tak lepas dari sebuah tantangan, tantangan yang sering dihadapi pasti akan terjadi, Maka yang harus dilakukan adalah memahami Langkah-langkah untuk menemukan solusi yang tepat (Refiana Said 2023)

1. Perkembangan Teknologi
   Pada saat ini teknologi yang berkembang pesat, akan mendorong kita semakin hari semakin harus belajar untuk tetap relevan, setelah itu keputusan dalam memilih

teknologi yang tepat adalah menjadi pertimbangan untuk proyek agar menjadi sebuah keputusan yang tepat supaya kedepannya tidak salah dalam memilih.

2. Perubahan Persyaratan
   Perubahan kebutuhan pelanggan atau *client* sering terjadi di tengah-tengah pengerjaan , hal ini dapat mengganggu dari jadwal dan rencana kerja yang sudah diatur, kemudian perubahan dalam strategi bisnis dapat mempengaruhi arah proyek. Dengan adanya tantangan tersebut maka dibutuhkan sebuah komunikasi, dan hubungan kerja yang baik agar pekerjaan dalam proyek berjalan dengan lancar

3. Komunikasi
   Tim berisi anggota dengan orang-orang yang berlatar belakang dan gaya kerja yang berbeda, kemudian Kesalahpahaman dalam komunikasi dapat menyebabkan kesalahan dan keterlambatan dalam bekerja.

4. Motivasi
   Pengembang seringkali bekerja dengan tenggat waktu yang ketat dan tekanan yang tinggi dan kurangnya pengakuan: Anggota tim perlu merasa dihargai dan diakui atas kontribusi mereka.

5. Manajemen Perubahan
   Anggota tim yang bekerja enggan mengubah cara kerja mereka bekerja, hal ini perlu penyesuaian dalam perubahan cara kerja agar lebih efisien

4.3 Peran Dan Tanggung Jawab Tim Pengelolaan Pengembang Perangkat Lunak

Sukses bekerja dalam suatu tim salah satunya adalah pengelolaan pengembangan proyek yang bisa saling mendukung dan komunikasi yang baik. Bagaimana proyek bisa berjalan dengan sukses yaitu dengan cara membagikan minat pelanggan kepada mereka, daripada mengkritik orang secara langsung, mereka mengkritik ide-ide nya, Membagikan pengalaman masa lalu dan masa kini untuk menawarkan wawasan tim baru dan saling percaya.

Tim yang solid dengan pembagian tugas yang jelas akan meningkatkan efisiensi dan kualitas produk akhir. Bab ini akan membahas secara rinci peran dan tanggung jawab masing-masing anggota dalam tim pengelolaan pengembangan perangkat lunak. peran dan tanggung jawab paling penting yang berpusat pada bisnis dan teknologi untuk setiap tim pengembangan perangkat lunak pada umumnya

1. Manajer Unit Bisnis

   Manajer unit bisnis adalah semacam manajer tugas. Tugas mereka adalah memastikan tujuan tercapai dan berkontribusi pada kesuksesan perusahaan dalam jangka panjang. Beberapa tanggung jawab mereka meliputi :

   a. Memulai proyek
   b. Memberikan visi proyek atau produk dan rencana strategis yang jelas
   c. Menandatangani tonggak penting

2. Manajer Unit Bisnis

   Meskipun tim bisnis yang lebih kecil mungkin memiliki manajer bisnis yang bertindak sebagai manajer produk, penting untuk membedakan peran dan tanggung jawab mereka.

3. Manajer produk

   Manajer produk bertanggung jawab untuk mengawasi keberhasilan produk tertentu, bukan bisnis secara keseluruhan. Tanggung jawab utama seorang manajer produk meliputi:

   a. Menerjemahkan visi proyek manajer unit bisnis ke dalam peta jalan
   b. Mengembangkan dan mendefinisikan kriteria dan fitur produk

4. Analis Bisnis
   Analis bisnis bekerja sebagai penghubung antara sisi teknis dan bisnis pembangunan. Mereka menggunakan data untuk meningkatkan proses dan menghasilkan laporan. Manajer produk dan analis bisnis bekerja sama untuk lebih menyempurnakan dan mendefinisikan fitur produk bersama dengan pimpinan teknis untuk memastikan mereka siap untuk pengembangan. Analisis bisnis meliputi tugas :
   a. Memperjelas fitur produk
   b. Menyelesaikan pertanyaan antara pimpinan teknis dan manajer produk
   c. Memastikan pengembang tidak terganggu oleh masalah bisnis

5. Manajer Proyek
   Manajer proyek mencerminkan manajer bisnis tetapi pada sisi teknis. Mereka memastikan bahwa tim pengembangan perangkat lunak menyelesaikan proyek mereka tepat waktu dan sesuai anggaran (Migunani et al. 2022). Untuk mencapai hal ini, mereka bertanggung jawab untuk:
   a. Penjadwalan, hosting, dan mendokumentasikan setiap pertemuan yang relevan.
   b. Mengamankan tim pengembangan perangkat lunak dengan sumber daya yang diperlukan untuk menyelesaikan proyek mereka
   c. Memantau kinerja tim pengembangan perangkat lunak dan memberikan saran yang berguna

6. Pimpinan Teknis
   Seorang pemimpin teknis bertindak bersama dengan tim pengembangan perangkat lunak, menyediakan titik kontak

utama bagi manajer proyek dan analis bisnis yang bersangkutan. Mereka berfungsi sebagai moderator antara manajemen bisnis dan pengembangan perangkat lunak serta mengawasi arahan teknis perusahaan (Migunani et al. 2022). Peran mereka meliputi:

a. Mengambil tanggung jawab untuk seluruh proyek teknis
b. Menerapkan standar dan prosedur pengkodean yang paling sesuai
c. Merinci struktur proyek dan memenuhi persyaratan

7. Pengembang

   Pengembang merupakan bagian penting dari tim pengembangan perangkat lunak. Mengingat kinerja pengembang mempunyai dampak penting terhadap pengembangan dan penerapan produk, waktu mereka digunakan dengan bijak untuk melakukan hal-hal berikut (Migunani et al. 2022):

   a. Mengembangkan dan menerapkan fitur
   b. Memperbarui pimpinan teknis dan manajer proyek dengan laporan proyek

8. Desainer UI/UX

   Para desainer ini akan terlibat secara menyeluruh dari awal hingga akhir. Untuk lebih jelasnya, desainer pengalaman pengguna) mengerjakan perilaku produk perangkat lunak sementara desainer antarmuka pengguna menangani desain grafis atau tata letak produk. Mereka bertanggung jawab untuk:

   a. Bekerja sama dengan manajer produk untuk menciptakan pengalaman pengguna yang layak dan memenuhi persyaratan

b. Mendukung tim pengembangan perangkat lunak selama proses pembangunan

9. Jaminan Mutu / Penguji Produk
   Penguji jaminan kualitas dapat membuat atau menghancurkan suatu produk hanya berdasarkan pemahaman mereka tentang persyaratan fitur dan masukan yang dihasilkan. Mereka yang berada di posisi ini memiliki keputusan akhir apakah fitur proyek/produk sudah sesuai atau tidak. Tanggung jawab utama meliputi:
   a. Menyatakan bahwa pengembang memenuhi kriteria dan ketentuan yang ditentukan dalam persyaratan.
   b. Secara aktif terlibat dengan tim pengembangan dan membimbing mereka melalui proses penjaminan kualitas.

4.4 Metode Pengembangan Perangkat Lunak

Perangkat lunak (*software*) menjadi komponen penting dalam dunia teknologi saat ini. Proses pengembangannya melibatkan metode dan pendekatan tertentu yang memungkinkan pengembang untuk merancang, membangun, dan menguji solusi perangkat lunak yang handal. Dalam artikel ini, kita akan menjelajahi beberapa metode pengembangan perangkat lunak yang umum digunakan .

1. *Waterfall*
   Model pengembangan *waterfall* atau air jatuh adalah model yang bersifat liner (linear sequential) yang mana setiap fase (perencanaan, analisis, desain, implementasi, pengujian, dan pemeliharaan) dilakukan secara berurutan. Kelebihan pada model ini adalah struktur yang jelas, mudah dipahami. Kekurangan nya adalah flesibelitas kurang untuk perubahan,

risiko tinggi jika terjadi kesalahan di awal (Titania Pricillia 2021).

2. Agile
   Model *agile* adala metode dengan model yang dilakukan secara berulang dan bertahap. Iteratif dan Inkremental yaitu Pengembangan dilakukan dalam siklus pendek (sprint), dengan menghasilkan produk yang dapat digunakan secara bertahap. Kelebihannya Fleksibel, responsif terhadap perubahan, melibatkan pelanggan secara aktif., serta kekurangan dari metode ini adalah membutuhkan disiplin tinggi dari tim, kurang cocok untuk proyek dengan persyaratan yang sangat ketat (Mahendra, Tresno & Yanto 2018)

3. Scrum
   Metode tersebut merupakan varian dari Agile, mnggunakan *sprint, backlog* produk, dan peran-peran spesifik (*Product Owner, Scrum Master, Development Team*). Kelebihannya adalah fokus pada kolaborasi, adaptasi, dan pengiriman nilai secara cepat, kekurangannya adalah membutuhkan perubahan budaya dalam organisasi (Raditia Vindua 2024)

4. DevOps
   Metode ini merupakan penggabungan pengembangan (*Development*) dan operasi (*Operations*) untuk mempercepat pengiriman perangkat lunak. Kelebihan dari metode ini adalah peningkatan kecepatan pengiriman, peningkatan kualitas, dan keandalan, kekurangannya membutuhkan budaya organisasi yang mendukung otomatisasi dan kolaborasi (Raditia Vindua 2024).

5. Lean

   Metode pengembangan ini memiliki ciri khas fokus pada nilai dimana mengidentifikasi dan menghilangkan pemborosan dalam proses pengembangan. Kelebihannya adalah meningkatkan efisiensi, mengurangi waktu pengembangan. Sedangkan kekurangannya membutuhkan perubahan mendasar dalam budaya organisasi.

DAFTAR PUSTAKA

Mahendra, I., Tresno, D. & Yanto, E., 2018, 'AGILE DEVELOPMENT METHODS DALAM PENGEMBANGAN SISTEM INFORMASI PENGAJUAN KREDIT BERBASIS WEB (STUDI KASUS : BANK BRI UNIT KOLONEL SUGIONO)', *JURNAL TEKNOLOGI DAN OPEN SOURCE*, 1(2).

Migunani, S., Kom, M. & Kom, 2022, *Rekayasa Perangkat Lunak*.

Raditia Vindua, S.Si. ,M. K., 2024, *Metode pengembangan Sistem Secara Tradisional dan Terbaru*.

Refiana Said, L., 2023, *TANTANGAN DAN PELUANG PENGELOLAAN ORGANISASI DI ERA DIGITAL*.

Titania Pricillia, Z., 2021, 'Perbandingan Metode Pengembangan Perangkat Lunak Waterfall Prototype RAD', *Bangkit Indonesia,* X(1), 1–12.

Zakhralativa Ruskanda, F., 2013, 'Pengembangan Perangkat Lunak Asesmen Kerja Tim', *Seminar Teknik Informatika dan Sistem Informasi*.

BAB 5
TANTANGAN DAN ISU-ISU TERKINI DALAM PENGEMBANGAN PERANGKAT LUNAK

5.1 Pendahuluan

Dalam era digital yang terus berkembang pesat, industri pengembangan perangkat lunak menghadapi serangkaian tantangan dan isu-isu yang semakin kompleks. Seiring dengan kemajuan teknologi yang eksponensial, para pengembang perangkat lunak dituntut untuk tidak hanya mengikuti perkembangan terbaru, tetapi juga mengantisipasi perubahan di masa depan. Lanskap teknologi yang dinamis ini menciptakan peluang besar sekaligus tantangan signifikan bagi profesional di bidang pengembangan perangkat lunak.

Menurut laporan "*State of Software Development* 2023" yang diterbitkan oleh *Stack Overflow*, lebih dari 70% pengembang perangkat lunak mengidentifikasi kecepatan perubahan teknologi sebagai salah satu tantangan utama yang mereka hadapi. Hal ini menunjukkan betapa pentingnya pemahaman dan adaptasi terhadap isu-isu terkini dalam industri ini.

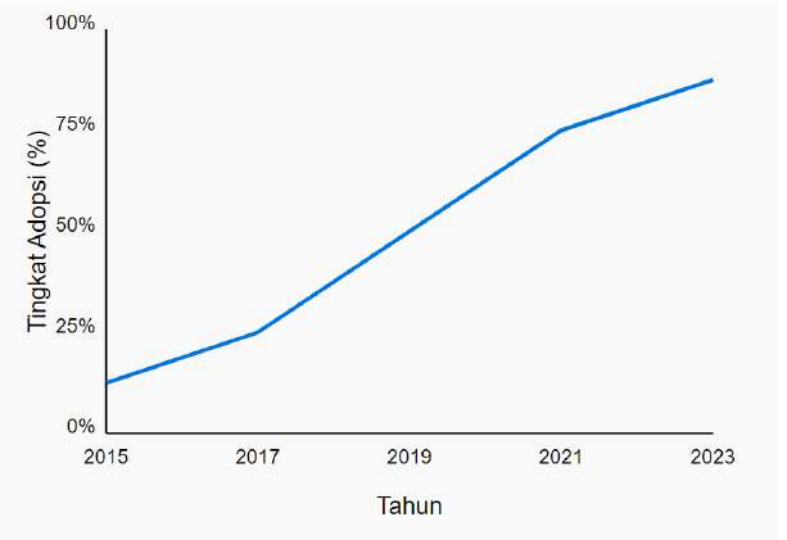
Bab ini bertujuan untuk mengeksplorasi dan menganalisis tantangan serta isu-isu terkini yang dihadapi oleh para pengembang perangkat lunak di era modern. Dengan memahami kompleksitas dan dinamika dari tantangan-tantangan ini, diharapkan para profesional di bidang pengembangan perangkat lunak dapat lebih siap menghadapi dan mengatasi berbagai rintangan dalam pekerjaan mereka.

Ruang lingkup pembahasan akan mencakup berbagai aspek pengembangan perangkat lunak, mulai dari tantangan teknis seperti keamanan dan skalabilitas, hingga isu-isu non-teknis seperti manajemen tim dan keberlanjutan. Setiap topik akan dibahas dengan mengacu pada penelitian terbaru, praktik industri terkini, dan studi kasus yang relevan. Dalam konteks transformasi digital, pengembangan perangkat lunak harus mempertimbangkan inovasi terbuka yang mengintegrasikan kompetensi digital. Ini terutama penting untuk mendukung keberlanjutan dan inovasi di berbagai sektor (Espina, et al. 2024).

5.2 Evolusi Teknologi dan Dampaknya

5.2.1 Perkembangan Pesat Teknologi

Industri teknologi informasi telah mengalami percepatan perkembangan yang belum pernah terjadi sebelumnya. Menurut laporan dari Gartner, Inc., kecepatan adopsi teknologi baru telah meningkat sebesar 25% dalam lima tahun terakhir. Grafik berikut menunjukkan tren adopsi teknologi dari tahun 2015 hingga 2023:



Gambar 5.1. GrafikTren Adopsi Teknologi dari Tahun 2015-2023

Grafik di atas menunjukkan peningkatan yang signifikan dalam tingkat adopsi teknologi baru dari tahun ke tahun. Hal ini mencerminkan bagaimana industri pengembangan perangkat lunak harus terus beradaptasi dengan cepat terhadap perubahan teknologi.

5.2.2 Adaptasi terhadap Perubahan Teknologi

Menghadapi perubahan teknologi yang cepat, pengembang perangkat lunak dituntut untuk terus memperbarui keterampilan mereka. Sebuah survei yang dilakukan oleh *IEEE Computer Society* menemukan bahwa 85% profesional IT merasa perlu meningkatkan keterampilan mereka setiap 12-18 bulan untuk tetap relevan dalam industri.

Penelitian dari Institute of Data menyoroti bahwa menerapkan program pembelajaran formal, seperti pelatihan dan sertifikasi, dapat membantu mengatasi tantangan keahlian yang berubah cepat dalam pengembangan perangkat lunak. Selain itu, menciptakan lingkungan yang mendukung, di mana karyawan didorong untuk mengambil risiko dan belajar dari kegagalan, memperkuat budaya belajar yang adaptif dan inovatif. Manfaat lainnya termasuk peningkatan kinerja individu dan tim, serta peningkatan kepuasan karyawan (Fonseca, et al. 2019).

5.2.3 Integrasi Teknologi Baru

Integrasi teknologi baru ke dalam proses pengembangan perangkat lunak yang ada merupakan tantangan tersendiri. Menurut laporan dari McKinsey & Company, hanya 30% perusahaan yang berhasil mengintegrasikan teknologi baru secara efektif ke dalam sistem mereka yang sudah ada. Hal ini disebabkan oleh berbagai masalah utama seperti kompatibilitas dengan sistem yang sudah ada, kurva pembelajaran yang curam bagi para

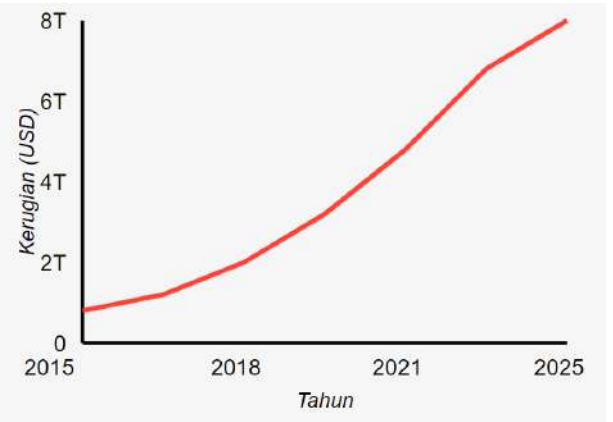
pengembang, serta resistensi terhadap perubahan dalam tim pengembang.

Untuk mengatasi tantangan ini, banyak organisasi menerapkan pendekatan bertahap, seperti memulai dengan proyek percontohan sebelum implementasi penuh. Selain itu, mengatasi masalah budaya organisasi dan menyelaraskan struktur tim dengan alur kerja teknologi baru sangat penting. Sebuah tinjauan literatur sistematis juga menyoroti bahwa pembelajaran berkelanjutan, pelatihan, dan peningkatan komunikasi dalam tim dapat meningkatkan kemampuan beradaptasi selama proses transisi teknologi (Hoy, Xu. 2023).

5.3 Keamanan dan Privasi Data

5.3.1 Ancaman Keamanan Siber

Keamanan siber telah menjadi salah satu isu paling kritis dalam pengembangan perangkat lunak modern. Menurut laporan *Cybersecurity* Ventures, *cyber crime* diproyeksikan akan menyebabkan kerugian global sebesar $10,5 triliun per tahun pada 2025. Grafik berikut menunjukkan tren peningkatan serangan siber dari tahun 2015 hingga proyeksi 2025:



Gambar 5.2. Grafik Tren Peningkatan Serangan Siber dari Tahun 2015 Hingga Proyeksi 2025

Grafik di atas menunjukkan peningkatan yang drastis dalam kerugian akibat serangan siber, menekankan pentingnya keamanan dalam pengembangan perangkat lunak.

Menurut sebuah studi, banyak organisasi menghadapi kesulitan dalam mendeteksi kerentanan ini dan dalam berbagi informasi yang cukup mengenai ancaman baru secara cepat, yang menyebabkan rentannya sistem terhadap serangan siber. Untuk mengatasi masalah ini, pengembang perangkat lunak perlu mengadopsi pendekatan proaktif seperti pemodelan ancaman dan integrasi langkah-langkah keamanan siber ke dalam siklus pengembangan perangkat lunak sejak dini. Dengan cara ini, potensi serangan dapat diidentifikasi dan dicegah lebih awal, mengurangi dampak kerusakan yang disebabkan oleh serangan siber yang sukses (Abbaspour, et al. 2024) (Bennouk, et al. 2024).

5.3.2 Implementasi Keamanan dalam Pengembangan

Mengintegrasikan keamanan ke dalam proses pengembangan perangkat lunak telah menjadi keharusan. Pendekatan "*Security by Design*" dan "*Privacy by Design*" semakin diadopsi oleh industri. Menurut survei yang dilakukan oleh Ponemon Institute, 71% organisasi yang mengadopsi pendekatan ini melaporkan peningkatan signifikan dalam keamanan aplikasi mereka.

Beberapa praktik terbaik dalam implementasi keamanan meliputi:

1. Pengujian keamanan terintegrasi
2. Enkripsi data *end-to-end*
3. Manajemen akses dan identitas yang kuat
4. Pemantauan dan logging keamanan
5. Pelatihan keamanan rutin untuk tim pengembang

5.4 *Artificial Intelligence* dan *Machine Learning*

5.4.1 Integrasi AI dalam Pengembangan Perangkat Lunak

Artificial Intelligence (AI) dan *Machine Learning* (ML) semakin terintegrasi dalam proses pengembangan perangkat lunak. Menurut laporan Gartner, pada tahun 2023, 40% dari tim pengembangan aplikasi perusahaan akan menggunakan *AI-augmented testing tools*.

Beberapa area di mana AI/ML memiliki dampak signifikan dalam pengembangan perangkat lunak meliputi:

1. *Automated code generation* dan *completion*
2. Prediksi bug dan analisis kualitas kode
3. Optimisasi performa aplikasi
4. Personalisasi pengalaman pengguna
5. *Automated testing* dan *quality assurance*

5.4.2 Tantangan Etis dalam Penggunaan AI

Dengan meningkatnya penggunaan AI dalam pengembangan perangkat lunak, muncul berbagai tantangan etis yang perlu dipertimbangkan. Menurut survei IEEE, 54% profesional teknologi percaya bahwa isu etika terkait AI adalah perhatian utama mereka.

Tantangan etis dalam penggunaan AI semakin mendominasi perdebatan di kalangan profesional teknologi, terutama seiring dengan peningkatan adopsi AI dalam berbagai sektor. Sebuah studi oleh Jobin et al. (2019) menegaskan bahwa isu bias dalam data pelatihan dapat berkontribusi pada hasil yang diskriminatif, yang berpotensi memperburuk ketidakadilan sosial. Penelitian ini juga menunjukkan pentingnya transparansi dan *explainability* dalam keputusan AI untuk membangun kepercayaan publik dan memastikan akuntabilitas. Dengan meningkatnya kekhawatiran tentang privasi data, penting bagi pengembang untuk mempertimbangkan keamanan dan ketahanan sistem AI terhadap serangan adversarial. Selain itu, otomatisasi yang dipicu oleh AI dapat memiliki dampak signifikan terhadap pekerjaan,

mengharuskan masyarakat untuk memikirkan kembali bagaimana mereka akan beradaptasi dengan perubahan tersebut.

Tabel berikut merangkum beberapa prinsip etis dalam pengembangan AI:

Tabel 5.1. Prinsip Etis dalam Pengembangan AI

Prinsip	**Deskripsi**
Fairness	AI harus adil dan tidak diskriminatif
Transparency	Proses pengambilan keputusan AI harus dapat dijelaskan
Privacy	AI harus menghormati privasi individu
Accountability	Ada tanggung jawab jelas atas keputusan AI
Safety	AI harus aman dan tidak membahayakan manusia

5.4.3 *Machine Learning Ops* (MLOps)

MLOps, atau *Machine Learning Operations*, muncul sebagai praktik untuk mengelola siklus hidup sistem ML dalam produksi. Menurut Cognilytica, pasar MLOps diproyeksikan tumbuh dari $350 juta pada 2019 menjadi $4 miliar pada 2025.

Tantangan dalam implementasi MLOps meliputi:

1. Mengelola kompleksitas *pipeline* ML
2. Memastikan reprodusibilitas eksperimen
3. Menangani *drift* dalam data dan model performance
4. Integrasi dengan infrastruktur IT yang ada
5. Memenuhi persyaratan regulasi dan *compliance*

5.5 *User Experience* (UX) dan Aksesibilitas

5.5.1 Desain Berpusat pada Pengguna

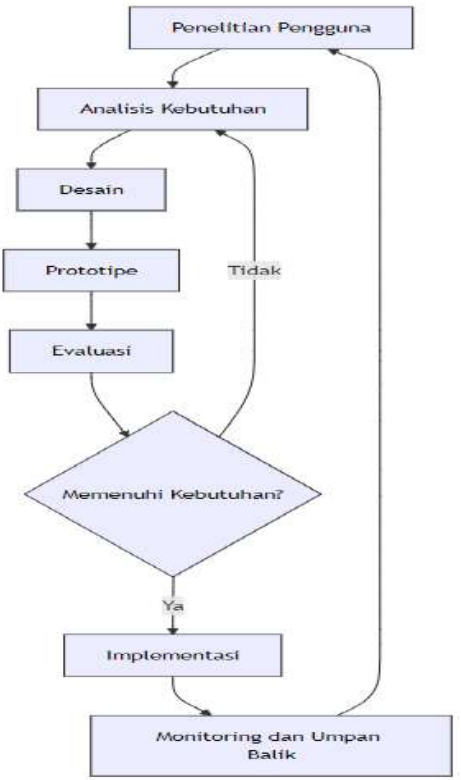
Desain berpusat pada pengguna (*User-Centered Design*, UCD) telah menjadi aspek kritis dalam pengembangan perangkat lunak modern. Menurut sebuah studi oleh Forrester, setiap dollar yang

diinvestasikan dalam UX menghasilkan *return on investment* (ROI) sebesar $100. Ini menunjukkan betapa pentingnya fokus pada pengalaman pengguna dalam proses pengembangan.

Prinsip-prinsip kunci dalam desain berpusat pada pengguna meliputi:

1. Empati terhadap kebutuhan pengguna
2. Iterasi berdasarkan umpan balik pengguna
3. Konsistensi dalam desain antarmuka
4. Minimalisasi beban kognitif pengguna
5. Pengujian kegunaan secara reguler

Berikut adalah diagram yang mengilustrasikan siklus desain berpusat pada pengguna:



Gambar 5.3. Siklus Desain Berpusat pada Pengguna

5.5.2 Aksesibilitas dan Inklusivitas

Aksesibilitas dalam pengembangan perangkat lunak bukan hanya tentang memenuhi regulasi, tetapi juga tentang menciptakan produk yang dapat digunakan oleh semua orang, termasuk mereka dengan disabilitas. Menurut World Health Organization, sekitar 15% populasi dunia hidup dengan beberapa bentuk disabilitas.

Beberapa praktik terbaik dalam mengimplementasikan aksesibilitas meliputi:

1. Mengikuti standar WCAG (*Web Content Accessibility Guidelines*)
2. Menggunakan semantic HTML untuk struktur konten
3. Menyediakan alternatif teks untuk konten non-teks
4. Memastikan navigasi keyboard yang baik
5. Menggunakan kontras warna yang cukup

Tabel berikut menunjukkan tingkat kepatuhan WCAG dan implikasinya:

Tabel 5.2. Tingkat Kepatuhan WCAG dan Implikasinya

Level	Deskripsi	Contoh Kriteria
A	Minimum	Semua konten non-teks memiliki alternatif teks
AA	Menengah	Rasio kontras warna minimal 4.5:1
AAA	Tinggi	Tersedia versi bahasa isyarat untuk semua video

5.5.3 Personalisasi Pengalaman Pengguna

Personalisasi telah menjadi aspek penting dalam meningkatkan engagement pengguna. Menurut Accenture, 91% konsumen lebih cenderung berbelanja dari merek yang menyediakan rekomendasi dan penawaran yang relevan.

Personalisasi dalam pengembangan perangkat lunak, terutama melalui penggunaan sistem rekomendasi berbasis *machine learning*, sangat penting dalam meningkatkan keterlibatan

pengguna. Strategi ini memanfaatkan algoritma canggih untuk memahami preferensi pengguna dan memberikan konten yang relevan. Penelitian terbaru menunjukkan bahwa integrasi metode kolaboratif dan personalisasi berbasis tindakan pengguna menghasilkan rekomendasi yang lebih tepat dan memperkaya pengalaman pengguna. Contoh implementasi ini dapat ditemukan pada studi tentang *hybrid recommender systems* yang dibuat oleh Troussas et, al (2024).

5.6 Keberlanjutan dan *Green Computing*

5.6.1 Efisiensi Energi dalam Pengembangan Perangkat Lunak

Dengan meningkatnya kesadaran akan dampak lingkungan dari teknologi, efisiensi energi dalam pengembangan perangkat lunak menjadi semakin penting. Menurut laporan Greenpeace, sektor IT bertanggung jawab atas sekitar 2-3% dari emisi gas rumah kaca global.

Beberapa strategi untuk meningkatkan efisiensi energi dalam pengembangan perangkat lunak meliputi:

1. Optimisasi algoritma untuk mengurangi penggunaan CPU
2. Efisiensi dalam penggunaan memori dan penyimpanan
3. Implementasi mode hemat daya dalam aplikasi
4. Optimisasi jaringan untuk mengurangi transfer data
5. Penggunaan *cloud computing* yang efisien energy

5.6.2 Praktik Pengembangan Berkelanjutan

Praktik pengembangan berkelanjutan bertujuan untuk mengurangi dampak lingkungan dari proses pengembangan perangkat lunak. Ini melibatkan berbagai aspek, dari penggunaan sumber daya hingga manajemen limbah elektronik.

Beberapa praktik pengembangan berkelanjutan meliputi:

1. Penggunaan perangkat keras yang efisien energi
2. Implementasi kebijakan paperless dalam tim pengembangan

3. Penggunaan tools kolaborasi online untuk mengurangi perjalanan
4. Daur ulang dan pembuangan yang bertanggung jawab untuk perangkat elektronik
5. Penggunaan energi terbarukan untuk *operasi data center*

5.7 Kesimpulan dan Pandangan ke Depan

Seiring dengan perkembangan teknologi yang terus berlanjut, industri pengembangan perangkat lunak akan terus menghadapi tantangan baru dan peluang yang menarik. Beberapa tren yang diperkirakan akan membentuk masa depan pengembangan perangkat lunak meliputi:

1. **Peningkatan Adopsi AI dan Machine Learning**: AI akan semakin terintegrasi dalam setiap aspek pengembangan perangkat lunak, dari generasi kode otomatis hingga pengujian prediktif.
2. **Edge Computing**: Dengan pertumbuhan *Internet of Things* (IoT), pengembangan untuk edge computing akan menjadi semakin penting.
3. **Quantum Computing**: Meskipun masih dalam tahap awal, quantum computing berpotensi merevolusi beberapa aspek pengembangan perangkat lunak, terutama dalam kriptografi dan optimisasi.
4. **Peningkatan Fokus pada Keamanan dan Privasi**: Dengan meningkatnya ancaman siber dan regulasi privasi yang lebih ketat, keamanan dan privasi akan menjadi aspek integral dalam setiap tahap pengembangan perangkat lunak.
5. **Evolusi Metodologi Pengembangan**: Metodologi pengembangan akan terus berevolusi untuk mengakomodasi kompleksitas yang meningkat dan kebutuhan untuk delivery yang lebih cepat.

6. **Keberlanjutan dan Green IT**: Fokus pada pengembangan perangkat lunak yang ramah lingkungan akan semakin meningkat seiring dengan kesadaran akan dampak lingkungan dari teknologi.

Menghadapi tantangan-tantangan ini akan membutuhkan adaptasi terus-menerus dari para profesional pengembangan perangkat lunak. Pendidikan berkelanjutan, kolaborasi lintas disiplin, dan inovasi akan menjadi kunci dalam menghadapi lanskap pengembangan perangkat lunak yang terus berubah.

Dengan memahami dan mengantisipasi tantangan-tantangan ini, industri pengembangan perangkat lunak dapat terus berkembang dan memberikan solusi yang inovatif dan berkelanjutan untuk masalah-masalah kompleks di dunia modern.

DAFTAR PUSTAKA

Accenture. (2023). Personalization Pulse Check.

Bennouk, K.; Ait Aali, N.; El Bouzekri El Idrissi, Y.; Sebai, B.; Faroukhi, A.Z.; Mahouachi, D. A Comprehensive Review and Assessment of Cybersecurity Vulnerability Detection Methodologies. *J. Cybersecur. Priv.* 2024, *4*, 853-908. https://doi.org/10.3390/jcp4040040

Bennouk, K.; Ait Aali, N.; El Bouzekri El Idrissi, Y.; Sebai, B.; Faroukhi, A.Z.; Mahouachi, D. A Comprehensive Review and Assessment of Cybersecurity Vulnerability Detection Methodologies. *J. Cybersecur. Priv.* 2024, *4*, 853-908. https://doi.org/10.3390/jcp4040040

Cognilytica. (2023). MLOps Market Forecast.

Cybersecurity Ventures. (2023). Cybercrime Report 2023.

Espina-Romero, L.; Ríos Parra, D.; Noroño-Sánchez, J.G.; Rojas-Cangahuala, G.; Cervera Cajo, L.E.; Velásquez-Tapullima, P.A. Navigating Digital Transformation: Current Trends in Digital Competencies for Open Innovation in Organizations. *Sustainability* 2024, *16*,2119. https://doi.org/10.3390/su16052119

Fonseca, L. R. D., Silva, M. R., Silva, S. W., & Pereira, G. M. (2019). Continuous-learning work environment: A study with developers in software development organizations. Knowledge Management & ELearning, 11(3), 281–303. https://doi.org/10.34105/j.kmel.2019.11.015

Forrester. (2023). The ROI of User Experience.

Gartner, Inc. (2023). Emerging Technology Adoption Trends.

Gartner. (2023). AI in Software Development.

Greenpeace. (2023). Clicking Clean: Who is Winning the Race to Build a Green Internet?

Hoy, Z.; Xu, M. Agile Software Requirements Engineering Challenges-

Solutions—A Conceptual Framework from Systematic Literature Review. *Information* **2023**, *14*, 322. https://doi.org/10.3390/info14060322

IEEE Computer Society. (2023). Technology Skills Survey.

International Energy Agency. (2023). Data Centres and Data Transmission Networks.

Jobin, A., Ienca, M. & Vayena, E. The global landscape of AI ethics guidelines. *Nat Mach Intell* **1**, 389–399 (2019). https://doi.org/10.1038/s42256-019-0088-2

McKinsey & Company. (2023). The State of AI in 2023.

Ponemon Institute. (2023). The State of Application Security.

Stack Overflow. (2023). State of Software Development 2023.

Troussas, C.; Krouska, A.; Koliarakis, A.; Sgouropoulou, C. Harnessing the Power of User-Centric Artificial Intelligence: Customized Recommendations and Personalization in Hybrid Recommender Systems. *Computers* **2023**, *12*, 109. https://doi.org/10.3390/computers12050109

World Health Organization. (2023). World Report on Disability.

BIODATA PENULIS

Purwita Sari, S.SI., M.Kom.
Dosen Program Studi Manajemen Informatika
Fakultas Ilmu Komputer Universitas Sriwijaya

Penulis lahir di Palembang tanggal 09 Juni 1992. Penulis adalah dosen tetap pada Program Studi Manajemen Informatika Fakultas Ilmu Komputer, Universitas Sriwijaya. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi di Universitas Sriwijaya dan melanjutkan S2 pada Jurusan Teknik Informatika di Universitas Bina Darma. Penulis menekuni bidang riset tentang Sistem Informasi.

BIODATA PENULIS

Dr. Lucky Indra Kesuma, S.SI., M.Kom
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer Universitas Sjakhyakirti Palembang

Penulis Lahir di Palembang, 25 September 1990. Menyelesaikan pendidikan sebagai Ahli Madya pada program studi komputerisasi Akuntansi pada tahun 2011 di Universitas Sriwijaya Melanjutkan Pendidikan sebagai sarjana S1 program studi Sistem Informasi lulus pada tahun 2014 di Universitas Sriwijaya, menempuh pendidikan kembali S2 Magister Komputer Program Studi Teknik Informatika menyelesaikan studi pada tahun 2017 di Universitas Binadarma, Pada tahun 2023 telah selesai menempuh pendidikan S3 Program Doktor program studi Ilmu Teknik (Informatika). Penulis adalah Dosen tetap pada program studi Teknologi Informasi Fakultas Teknik Universitas Muhammadiyah Palembang.

BIODATA PENULIS

Ir. Ali Impron, S.Kom., M.Kom.
Dosen Program Studi Informatika
Fakultas Teknik dan Pertanian Universitas Muhammadiyah Sampit

Penulis lahir di Grobogan, Jawa Tengah, pada tahun 1983. Penulis adalah seorang dosen dan profesional IT dengan pengalaman lebih dari 17 tahun dalam mengelola layanan IT, internet service provider, dan industri pertambangan. Penulis memiliki keahlian dalam infrastruktur IT, IoT, Data Center, dan Bussiness Intelligence. Saat ini, penulis menjabat sebagai Head of Information Technology di PT. Darma Henwa Tbk, di mana penulis bertanggung jawab untuk mengelola tim IT dan memastikan layanan IT berkualitas tinggi.

Penulis menyelesaikan gelar Sarjana Teknik Informatika dari STMIK AKAKOM Yogyakarta dan gelar Magister Teknologi Informasi dari Universitas Teknologi Digital Indonesia, Yogyakarta. Saat ini, penulis sedang menempuh Program Doktor Ilmu Teknik di Universitas Negeri Yogyakarta. Selain perannya di dunia industri, penulis juga tercatat sebagai dosen tetap di Universitas Muhammadiyah Sampit.

Selama karirnya, penulis telah memperoleh berbagai sertifikasi internasional yang mengakui keahliannya di bidang teknologi informasi, termasuk Microsoft Certified IT Professional (MCITP), Cisco Certified Network Associate (CCNA), dan Juniper Networks Certified Associate (JNCIA). Penulis memiliki minat besar dalam penelitian dan integrasi produk IT terbaru dengan proses bisnis, serta memiliki pemahaman mendalam tentang administrasi sistem, keamanan IT, dan jaringan.

BIODATA PENULIS

Safira Fegi Nisrina
Desen Universitas Widya Husada Semarang

Safira Fegi Nisrina, lahir di Semarang, 28 September 1996. Melanjutkan Pendidikan ke jenjang sarjana jurusan Teknik elektro di Universitas Islam Sultan Agung Semarang pada tahun 2014. Dipercayakan menjadi asisten pelaksana praktikum di laboraturium Dasar Telekomunikasi pada tahun 2016. Kemudian dengan bekal ilmu sarjana melanjutkan studi pasca sarjana S2 di Universitas Islam Sultan Agung Semarang jurusan Teknik elektro pada tahun 2020.

Sejak tahu 2020 menjadi dosen/ tenaga pengajar di salah satu perguruan tinggi suasta yakni Universitas Widya Husada Semarang untuk mengampu beberapa mata kuliah pada program studi Teknologi Elektro Medis.

BIODATA PENULIS

Reska Mayefis, S.Pd., M.Pd. T
Dosen Program Studi D3 Tata Boga
Fakultas Pariwisata dan Perhotelan Universitas Negeri Padang

Penulis lahir di Padang tanggal 07 Maret 1994. Penulis adalah dosen tetap pada Program Studi D3 Tata Boga Fakultas Pariwisata dan Perhotelan, Universitas Negeri Padang. Menyelesaikan pendidikan S1 pada Jurusan Pendidikan Teknik Informatika dan Komputer di Universitas Bung Hatta dan melanjutkan S2 pada Jurusan Pendidikan Teknologi dan Kejuruan di Universitas Negeri Padang.