sada
kurnia pustaka

ARSITEKTUR PERANGKAT LUNAK

Panduan Komprehensif Perancangan Sistem

Tim Penulis:
Dahlan | Norbertus Tri Suswanto Saptadi | Ahmad Budi Trisnawa
Budi Chehabudin | Hendri Julian Pramana | Kotim Subandi | Irawan Afrianto
Bayu Waseso | Gergorius Kopong Pati | Lilis Supratman

ARSITEKTUR PERANGKAT LUNAK

Panduan Komprehensif Perancangan Sistem

Dahlan
Norbertus Tri Suswanto Saptadi
Ahmad Budi Trisnawa
Budi Chehabudin
Hendri Julian Pramana
Kotim Subandi
Irawan Afrianto
Bayu Waseso
Gergorius Kopong Pati
Lilis Supratman

Editor: Ajay Supriadi, M.Kom.

sada
kurnia pustaka

ARSITEKTUR PERANGKAT LUNAK

Panduan Komprehensif Perancangan Sistem

Tim Penulis:
Dahlan
Norbertus Tri Suswanto Saptadi
Ahmad Budi Trisnawa
Budi Chehabudin
Hendri Julian Pramana
Kotim Subandi
Irawan Afrianto
Bayu Waseso
Gergorius Kopong Pati
Lilis Supratman

Editor : Ajay Supriadi, M.Kom.
Tata Letak : Lilis Khalisatul Karimah, S.H.
Desain Cover : Asep Nugraha, S.Hum.
Ukuran : UNESCO 15,5 x 23 cm
Halaman : vii, 165
ISBN : 978-634-7522-33-7
Terbit Pada : Maret 2026
Anggota IKAPI : No. 073/BANTEN/2023

PENERBIT PT SADA KURNIA PUSTAKA
Jl. Kramat, Panenjoan Kec. Carenang, Kab. Serang – Banten, 42195
Email : sadapenerbit@gmail.com
Website : sadapenerbit.com & repository.sadapenerbit.com
Telpon/WA : +62 838 1281 8431

KATA PENGANTAR

Puji syukur ke hadirat Tuhan Yang Maha Esa, karena berkat rahmat dan karunia-Nya buku "*Arsitektur Perangkat Lunak: Panduan Komprehensif" Perancangan Sistem* dapat terselesaikan dengan baik. Buku ini hadir sebagai upaya untuk memberikan pemahaman yang mendalam mengenai konsep, prinsip, dan praktik arsitektur perangkat lunak yang menjadi fondasi penting dalam pengembangan sistem modern.

Dalam era digital yang semakin kompleks, kebutuhan akan sistem yang terstruktur, efisien, dan berkelanjutan menjadi semakin mendesak. Arsitektur perangkat lunak bukan hanya sekadar rancangan teknis, melainkan juga strategi untuk memastikan kualitas, skalabilitas, dan keberlangsungan sistem. Oleh karena itu, buku ini disusun secara sistematis agar dapat menjadi panduan komprehensif bagi mahasiswa, peneliti, praktisi, maupun pengembang perangkat lunak.

Isi buku ini mencakup berbagai aspek mulai dari konsep dasar arsitektur, pola desain, metodologi perancangan, hingga studi kasus penerapan dalam berbagai bidang. Harapannya, pembaca tidak hanya memperoleh pengetahuan teoritis, tetapi juga keterampilan praktis yang dapat diaplikasikan dalam dunia kerja maupun penelitian.

Semoga buku ini dapat memberikan manfaat nyata, memperkaya wawasan, serta menjadi referensi utama dalam memahami dan mengimplementasikan arsitektur perangkat lunak.

Akhir kata, semoga buku ini dapat menjadi kontribusi kecil dalam mendukung perkembangan ilmu pengetahuan dan praktik teknologi informasi di Indonesia.

Penulis

DAFTAR ISI

BAB 1
INTRODUKSI ARSITEKTUR PERANGKAT LUNAK

Ir. Dahlan, S.T., M.Kom.
Universitas Muhammadiyah Bima

Latar Belakang

Perkembangan teknologi informasi dalam tiga dekade terakhir telah mengubah lanskap sistem komputasi dari sistem monolitik berskala kecil menjadi sistem terdistribusi berskala besar yang kompleks, adaptif, dan saling terintegrasi. Transformasi *digital* di berbagai sektor pendidikan, kesehatan, pemerintahan, industri, hingga keuangan menempatkan perangkat lunak sebagai tulang punggung infrastruktur modern.

Kompleksitas ini tidak lagi hanya berkaitan dengan baris kode, tetapi melibatkan integrasi lintas platform, interoperabilitas, keamanan, skalabilitas, serta keberlanjutan sistem dalam jangka panjang. Arsitektur perangkat lunak menjadi fondasi konseptual yang menentukan kualitas, berkelanjutan, dan keberhasilan sistem. Arsitektur bukan sekadar struktur teknis, melainkan representasi keputusan desain tingkat tinggi yang memengaruhi seluruh siklus hidup perangkat lunak.

Ia menjembatani kebutuhan bisnis dan implementasi teknis, sekaligus menjadi sarana komunikasi antara pemangku kepentingan (Kleppmann, 2017). Tanpa arsitektur yang dirancang secara sistematis, sistem cenderung berkembang secara *ad hoc*, sulit dipelihara, dan rentan terhadap kegagalan. Pengalaman empiris di industri menunjukkan bahwa banyak proyek gagal bukan karena kesalahan pengkodean, tetapi akibat keputusan arsitektural yang tidak tepat pada tahap awal pengembangan. Oleh karena itu, pemahaman konseptual dan metodologis tentang arsitektur perangkat lunak menjadi kompetensi esensial bagi akademisi maupun praktisi (Brown, 2018). Bab ini menguraikan landasan konseptual arsitektur perangkat lunak, evolusinya, ruang lingkup kajian, serta relevansinya dalam konteks sistem modern berbasis *cloud, microservices,* dan *Internet of Things* (IoT).

Definisi dan Konsep Dasar Arsitektur Perangkat Lunak

Arsitektur perangkat lunak merupakan fondasi konseptual yang menentukan bagaimana suatu sistem dirancang, dibangun, dan dikembangkan sepanjang siklus hidupnya. Pada tingkat paling mendasar, arsitektur mendefinisikan struktur sistem, elemen-elemen penyusunnya, hubungan antar elemen tersebut, serta prinsip dan batasan yang mengatur desain dan evolusinya.

Arsitektur bukan sekadar diagram teknis, melainkan representasi strategis yang menjembatani kebutuhan bisnis dengan solusi teknologis (Kruchten, 2004). Arsitektur perangkat lunak mulai diformalkan sebagai disiplin tersendiri pada akhir 1980-an. Kontribusi penting datang dari Mary Shaw dan David Garlan yang mendefinisikan arsitektur sebagai struktur sistem yang terdiri dari komponen, konektor, serta pola interaksi di antara keduanya.

Definisi ini menekankan pentingnya struktur dan organisasi sistem dibanding sekadar detail implementasi. Menurut (Richards & Ford, 2020) arsitektur perangkat lunak adalah struktur atau struktur-struktur dari sistem yang mencakup elemen perangkat lunak, properti yang terlihat secara eksternal dari elemen tersebut, dan hubungan di antaranya.

Properti eksternal mencakup antarmuka (*interfaces*), protokol komunikasi, serta perilaku yang dapat diamati oleh komponen lain. Definisi ini menunjukkan bahwa arsitektur berfokus pada aspek yang berdampak pada integrasi dan interoperabilitas sistem.

1. **Perbedaan Arsitektur dan Desain**

   Sering terjadi kerancuan antara istilah arsitektur dan desain perangkat lunak. Arsitektur berfokus pada keputusan tingkat tinggi (*high-level decisions*) yang memiliki dampak luas dan jangka panjang, seperti pemilihan pola arsitektur, strategi komunikasi antar layanan, atau pemilihan infrastruktur *deployment*. Sementara itu, desain detail berkaitan dengan implementasi teknis pada level modul atau kelas, seperti struktur data, algoritma, dan penulisan kode (Evans, 2004).

2. **Elemen Utama Arsitektur**

   a. Komponen

      Komponen adalah unit modular dalam sistem yang memiliki tanggung jawab spesifik. Komponen dapat berupa modul, layanan (*services*), subsistem, atau bahkan aplikasi eksternal yang terintegrasi. Setiap komponen memiliki batas tanggung jawab yang jelas.

   b. Konektor

      Konektor adalah mekanisme komunikasi antar komponen. Contohnya meliputi pemanggilan prosedur (*procedure call*),

REST API, *message queue, event bus*, atau protokol jaringan. Konektor menentukan bagaimana data dan kontrol mengalir dalam sistem.

c. Konfigurasi

Konfigurasi adalah susunan atau pola hubungan antara komponen dan konektor. Konfigurasi inilah yang membentuk struktur keseluruhan sistem.

d. *Constraint* dan Prinsip Arsitektur

Setiap arsitektur memiliki batasan dan prinsip desain yang harus dipatuhi, seperti pemisahan tanggung jawab (*separation of concerns*) atau aturan dependensi antar lapisan.

3. Prinsip Fundamental Dalam Arsitektur

Arsitektur perangkat lunak dibangun di atas sejumlah prinsip dasar yang bertujuan meningkatkan kualitas sistem:

a. *Separation of Concerns*

Sistem harus dibagi berdasarkan tanggung jawab yang berbeda agar kompleksitas dapat dikendalikan.

b. *Modularity*

Sistem dibangun dari modul-modul independen sehingga perubahan pada satu modul tidak berdampak besar pada modul lain.

c. *Abstraction*

Detail implementasi disembunyikan untuk mengurangi kompleksitas dan meningkatkan fleksibilitas.

d. *Information Hiding*

Prinsip ini diperkenalkan oleh David Parnas yang menyatakan bahwa setiap modul harus menyembunyikan detail internalnya agar perubahan tidak menyebar ke seluruh sistem.

e. *Loose Coupling* dan *High Cohesion*

Komponen harus memiliki ketergantungan minimal (*loose coupling*) dan tanggung jawab internal yang terfokus (*high cohesion*).

4. Arsitektur Sebagai Keputusan Strategis

Arsitektur tidak hanya berorientasi teknis, tetapi juga strategis. Keputusan seperti menggunakan arsitektur monolitik atau *microservices*, memilih teknologi *cloud* tertentu, atau menentukan pendekatan keamanan akan berdampak pada biaya, waktu

pengembangan, serta risiko jangka panjang. Dalam organisasi besar, arsitektur sering dikaitkan dengan tata kelola TI (IT *governance*) dan strategi bisnis. Oleh karena itu, arsitek perangkat lunak harus memahami konteks bisnis selain aspek teknis.

5. **Arsitektur dan Atribut Kualitas**
   Salah satu tujuan utama arsitektur adalah menjamin atribut kualitas sistem, seperti skalabilitas, keamanan, kinerja, dan ketersediaan. Keputusan arsitektural menentukan bagaimana sistem akan merespons peningkatan beban, serangan keamanan, atau kegagalan komponen. Pemilihan arsitektur berbasis layanan (*service-oriented*) dapat meningkatkan skalabilitas, namun menambah kompleksitas komunikasi antar komponen. Dengan demikian, setiap keputusan arsitektural selalu melibatkan *trade-off*.
6. **Representasi Arsitektur**
   Arsitektur biasanya direpresentasikan dalam bentuk model atau diagram. Pendekatan populer seperti model 4+1 *View* oleh Philippe Kruchten membantu menggambarkan sistem dari berbagai perspektif: logis, pengembangan, proses, fisik, dan skenario penggunaan. Selain itu, notasi seperti UML dan C4 Model juga banyak digunakan untuk mendokumentasikan struktur sistem secara visual dan sistematis.
7. **Evolusi Konsep Arsitektur**
   Konsep arsitektur perangkat lunak terus berkembang mengikuti kemajuan teknologi. Pada era awal, fokus utama adalah pada struktur modular dan *client-server*.

   Saat ini, arsitektur telah mencakup sistem terdistribusi, *cloud-native*, *containerization*, hingga integrasi kecerdasan buatan. Perubahan ini menunjukkan bahwa arsitektur bukan konsep statis, melainkan disiplin dinamis yang terus beradaptasi terhadap kebutuhan industri dan inovasi teknologi.

   Secara keseluruhan definisi dan konsep dasar arsitektur perangkat lunak menekankan bahwa arsitektur adalah struktur fundamental yang menentukan arah dan kualitas sistem. Ia mengintegrasikan prinsip rekayasa, kebutuhan bisnis, serta pertimbangan teknis menjadi satu kesatuan yang koheren.

Daftar Pustaka

Bass, L., Clements, P., & Kazman, R. (2022). *Software Architecture In Practice (4th ed.).* Addison-Wesley.

Brown, S. (2018). *Software Architecture for Developers*. Leanpub.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., & Stafford, J. (2010). *Documenting Software Architectures: Views And Beyond (2nd ed.).* Addison-Wesley.

Evans, E. (2004). *Domain-Driven Design: Tackling Complexity In The Heart of Software.* Addison-Wesley.

Ford, N., Parsons, R., & Kua, P. (2017). *Building Evolutionary Architectures: Support Constant Change.* O'Reilly Media.

Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code (2nd ed.).* Addison-Wesley.

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

Humble, J., & Farley, D. (2011). *Continuous Delivery: Reliable Software Releases Through Build, Test, And Deployment Automation*. Addison-Wesley.

ISO/IEC/IEEE. (2011). ISO/IEC/IEEE 42010:2011 Systems and Software Engineering-Architecture Description. International Organization for Standardization.

Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media.

Kruchten, P. (2004). *The Rational Unified Process: An Introduction (3rd ed.).* Addison-Wesley.

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

Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach.* O'Reilly Media.

PROFIL PENULIS

Ir. Dahlan, S.T., M.Kom.

Ketertarikan penulis terhadap ilmu komputer dimulai pada tahun 2011 silam. Hal tersebut membuat penulis memilih untuk masuk ke Sekolah Menengah Kejuruan di SMK Negeri 2 Kota Bima dengan memilih Jurusan Teknik Komputer dan Jaringan dan berhasil lulus pada tahun 2014. Penulis kemudian melanjutkan pendidikan ke Perguruan Tinggi dan berhasil menyelesaikan Studi S1 di prodi Teknik Informatika Universitas Islam Makassar pada tahun 2019.

Dua tahun kemudian, penulis melanjutkan studi S2 dan menyelesaikan studi di prodi Sistem Komputer Program Pasca Sarjana Universitas Handayani Makassar pada tahun 2024. Pada tahun 2024 penulis kemudian melanjutkan studi di Universitas Negeri Makassar pada program studi Profesi Insinyur BK Teknik Informatika dan menyelesaikan studi pada tahun 2025. Penulis memiliki kepakaran di bidang *Computer System Engineering*. Serta untuk mewujudkan karir sebagai dosen profesional, penulis pun aktif sebagai peneliti bidang kepakaran tersebut. Selain peneliti, penulis juga menulis buku dengan harapan dapat memberikan kontribusi positif bagi bangsa dan negara RI.

Email Penulis: dahlanlanggudu@gmail.com.

BAB 2
KARAKTERISTIK DAN ATRIBUT KUALITAS SISTEM

Dr. Ir. Norbertus Tri Suswanto Saptadi, S.Kom., M.T., M.M., IPM.
Universitas Atma Jaya Makassar

Pendahuluan

Transisi menuju Masyarakat Pengetahuan 4.0 menggarisbawahi pentingnya penciptaan suatu nilai berbasis data, dengan 7 (tujuh) tugas utama manajemen pengetahuan yang meliputi: pengumpulan data, pengolahan data, pembangkitan pengetahuan, distribusi, pemanfaatan, pengembangan organisasi pembelajaran, dan adaptasi terhadap kemajuan teknologi (Radlbauer, Moser and Wagner, 2025).

Arsitektur perangkat lunak (*software*) merupakan fondasi konseptual yang menentukan bagaimana sistem dirancang, dibangun, dan dikembangkan dalam jangka panjang. Dalam konteks sistem yang semakin kompleks dan terdistribusi, arsitektur tidak hanya berfungsi sebagai cetak biru (*blue print*) teknis, tetapi juga sebagai sarana untuk menjembatani kebutuhan bisnis dengan solusi teknologi yang tepat. Perancangan arsitektur yang baik menjadi faktor kunci keberhasilan dalam pengembangan perangkat lunak (Study *et al.*, 2024).

Keberhasilan sebuah sistem perangkat lunak tidak semata-mata diukur dari kemampuannya menyediakan fungsi yang sesuai dengan kebutuhan pengguna semata. Aspek non-fungsional, seperti kinerja, keandalan, keamanan, dan kemudahan pemeliharaan akan memiliki peran penting dalam menentukan kualitas sistem secara keseluruhan. Aspek-aspek ini menjadi pembeda utama antara sistem yang hanya berfungsi dan sistem yang benar-benar bernilai bagi organisasi. Dalam praktiknya, banyak kegagalan sistem bukan dikarenakan kesalahan fungsional, melainkan lemahnya perhatian kualitas sistem sejak tahap awal perancangan. Keputusan arsitektural yang kurang tepat dapat mengakibatkan sistem sulit dikembangkan, tidak mampu menangani pertumbuhan beban, atau kurang begitu adaptif terhadap perubahan kebutuhan. Hal ini menunjukkan pentingnya memahami kualitas sistem sebagai bagian integral arsitektur perangkat lunak.

Karakteristik dan atribut kualitas sistem memberikan kerangka berpikir dasar bagi arsitek untuk mengevaluasi dan mengarahkan keputusan desain. Karakteristik sistem menggambarkan sifat umum dan struktur dasar sistem, sementara atribut kualitas menjelaskan tingkat keunggulan sistem dalam berbagai aspek non-fungsional. Keduanya saling melengkapi dan menjadi dasar dalam menilai apakah sebuah arsitektur akan mampu memenuhi tujuan teknis dan bisnis.

Peran karakteristik dan atribut kualitas sistem dalam konteks arsitektur perangkat lunak difokuskan pada pemahaman konsep

dasar, jenis karakteristik sistem, serta pentingnya atribut kualitas sebagai suatu pertimbangan utama dalam pengambilan keputusan arsitektural (Bi, Liang and Tang, 2017). Dengan pemahaman ini, diharapkan mampu merancang sistem yang tidak hanya berfungsi dengan baik, tetapi juga berkualitas tinggi dan berkelanjutan.

Konsep Karakteristik Sistem

Karakteristik sistem merujuk pada sifat dasar yang menggambarkan bagaimana sebuah sistem perangkat lunak disusun, berperilaku, dan berinteraksi dengan suatu lingkungan internal maupun eksternal. Karakteristik muncul sebagai hasil langsung dari suatu keputusan arsitektural, seperti pemilihan gaya arsitektur, pola desain, serta mekanisme komunikasi antar komponen.

Melalui cara memahami karakteristik sistem, arsitek memperoleh gambaran menyeluruh mengenai kekuatan dan keterbatasan sistem yang dirancang. Karakteristik sistem bersifat menyeluruh dan lintas komponen, sehingga tidak dapat dikaitkan hanya dengan satu modul atau fungsi tertentu. Perubahan bagian sistem seringkali berdampak pada karakteristik sistem secara keseluruhan. Karakteristik sistem menjadi alat konseptual untuk mengevaluasi konsekuensi jangka panjang dari keputusan desain, terutama sistem berskala besar dan kompleks.

Karakteristik sistem perangkat lunak memiliki kompleksitas di mana mencerminkan tingkat kerumitan struktur internal sistem, jumlah komponen, serta pola interaksi di antaranya. Sistem dengan kompleksitas tinggi cenderung lebih sulit untuk dipahami, diuji, dan dipelihara. Dalam mengendalikan kompleksitas, arsitektur yang baik menerapkan prinsip modularisasi, pemisahan tanggung jawab, dan penggunaan abstraksi yang jelas (Sidqi, Yuswanto and Yaqin, 2024).

Karakteristik skalabilitas adalah kemampuan sistem menangani peningkatan beban kerja, jumlah pengguna, atau volume data tanpa mengalami penurunan kinerja signifikan. Skalabilitas dipengaruhi oleh struktur arsitektur dan strategi distribusi komponen. Sistem yang dirancang tanpa mempertimbangkan skalabilitas akan berisiko mengalami keterbatasan serius ketika kebutuhan bisnis berkembang.

Keterpaduan (*cohesion*) dan keterikatan (*coupling*) merupakan karakteristik struktural yang mencerminkan kualitas organisasi modular sistem. *Cohesion* yang tinggi menunjukkan bahwa elemen dalam satu modul memiliki tanggung jawab yang saling berkaitan erat, sedangkan *coupling* yang rendah akan menandakan minimnya dari ketergantungan antar modul. Kombinasi *cohesion* tinggi dan *coupling* rendah menjadi tujuan utama dalam perancangan arsitektur yang mudah dipelihara dan dikembangkan.

Karakteristik sistem mencakup fleksibilitas dan konsistensi. Fleksibilitas menggambarkan kemampuan sistem untuk beradaptasi terhadap perubahan kebutuhan atau lingkungan sekitar, sementara konsistensi berkaitan dengan keseragaman aturan, mekanisme, dan pola desain yang digunakan di seluruh sistem. Konsistensi yang baik membantu mengurangi kesalahan dan mempermudah pemahaman terhadap sistem oleh tim pengembang perangkat lunak.

Fungsionalitas menggambarkan sejauh mana perangkat lunak mampu menyediakan fungsi-fungsi yang sesuai dengan kebutuhan pengguna dan tujuan sistem. Karakteristik ini menekankan pada kelengkapan fitur, kebenaran proses, serta kesesuaian hasil yang diperoleh dengan spesifikasi yang telah ditetapkan. Perangkat lunak ini akan mampu menjalankan seluruh skenario penggunaan tanpa kesalahan logika dan mendukung aktivitas pengguna secara efektif.

Reliabilitas menunjukkan kemampuan perangkat lunak untuk beroperasi secara konsisten dan stabil dalam kondisi tertentu selama periode waktu tertentu. Karakter ini mencakup tingkat keandalan sistem dalam menangani kesalahan, mencegah kegagalan, serta dalam memulihkan diri ketika terjadi gangguan. Perangkat lunak yang reliabel akan meminimalkan *downtime* dan memastikan layanan tetap tersedia sesuai dengan yang diharapkan pengguna.

Usabilitas berkaitan dengan kemudahan pengguna dalam upaya mempelajari, memahami, dan menggunakan suatu perangkat lunak. Karakteristik ini menekankan pada desain antarmuka yang intuitif, konsistensi navigasi, serta kejelasan suatu informasi yang disajikan. Perangkat lunak dengan usabilitas tinggi memungkinkan pengguna menyelesaikan tugas dengan cepat, tepat, akurat, dan dengan tingkat kepuasan yang tinggi tanpa memerlukan pelatihan yang kompleks.

Efisiensi menggambarkan kemampuan perangkat lunak dalam menggunakan sumber daya sistem secara optimal, seperti waktu

pemrosesan, memori, dan kapasitas penyimpanan. Perangkat lunak yang efisien mampu memberikan kinerja yang baik dengan konsumsi sumber daya yang minimal, bahkan ketika menangani beban kerja yang tinggi. Karakteristik ini penting untuk memastikan sistem tetap responsif dan ekonomis dalam pengoperasiannya.

Pemeliharaan mengacu pada kemudahan perangkat lunak untuk dapat diperbaiki, dimodifikasi, dan dikembangkan lebih lanjut. Karakteristik dipengaruhi oleh struktur kode yang rapi, dokumentasi yang jelas, serta penggunaan suatu standar pengembangan yang baik. Perangkat lunak yang mudah dipelihara memungkinkan pengembang melakukan perbaikan *bug*, penyesuaian kebutuhan, dan peningkatan fitur dengan risiko kesalahan yang rendah.

Portabilitas menunjukkan kemampuan perangkat lunak untuk dijalankan pada berbagai lingkungan atau *platform* dengan sedikit atau tanpa perubahan. Karakteristik ini mencakup kompatibilitas dengan keberadaan sistem operasi, perangkat keras, dan konfigurasi berbeda. Perangkat lunak yang portabel memberikan fleksibilitas tinggi bagi pengguna dan organisasi karena dapat diimplementasikan di berbagai sistem tanpa biaya adaptasi yang besar.

Gambar 2.1: Karakteristik Perangkat Lunak
Sumber: https://www.kaashivinfotech.com/blog/characteristics-of-software-in-software-engineering/.

Daftar Pustaka

Bi, T., Liang, P. and Tang, A. (2017). *Architecture Patterns, Quality Attributes, and Design Contexts: How Developers Design with Them,* (61472286).

Cahyani, E., Nururrohmah, T. and Deka, C.F. (2024). The Role of Service Quality in Building Customer Satisfaction, *Journal of Management and Creative Business*, 2(4), pp. 192–205.

Kevin, A. and Kurniawan, H.C. (2020). Analisis Pengaruh Design Pattern Terhadap Pemeliharaan Perangkat Lunak Learning Management System, *Jurnal Telematika*, 18(1), pp. 21–31.

Komputer, S. *et al.* (2023). Perancangan Sistem Informasi Manajemen Persediaan Berbasis Web di CV. Makmur Sejahtera Palopo, *Processor: Jurnal Ilmiah Sistem Informasi, Teknologi Informasi dan Sistem Komputer*, 18(2), pp. 227–240.

Mahdi Sahlabadi, Ravie Chandren Muniyandi, Z.S. And F.Q. (2022). Lightweight Software Architecture Evaluation for Industry: A Comprehensive Review, *Sensors [Preprint].* Available at: https://doi.org/https://doi.org/10.3390/s22031252.

Nabot, A. And Al-Qerem, A. (2025). Impact of Software Quality on Organizational Performance, *Array*, 27(February), p. 100476. Available at: https://doi.org/10.1016/j.array.2025.100476.

Radlbauer, E., Moser, T. And Wagner, M. (2025). Designing a System Architecture For Dynamic Data Collection As A Foundation For Knowledge Modeling in Industry, *Applied Sciences*, pp. 1–28.

Santos, A. *et al.* (2025). Context-Aware Systems Architecture in Industry 4.0: A Systematic Literature Review, *Applied Sciences*, pp. 1–23. Available at: https://doi.org/https://doi.org/10.3390/app15115863.

Sidqi, A., Yuswanto, A. And Yaqin, M.A. (2024). Analisis Pengukuran Kompleksitas Website SOC Menggunakan Function Point, *JACIS : Journal Automation Computer Information System*, 4(2), pp. 67–75.

Study, C. *et al.* (2024). Studi Komparasi Pemilihan Software dan Tipologi Dalam Proses Desain Studio Arsitektur, *Border: Jurnal Arsitektur*, 6(1), pp. 15–28. Available at: https://doi.org/https://doi.org/10.33005/border.v6i1.756.

PROFIL PENULIS

Dr. Ir. Norbertus Tri Suswanto Saptadi, S.Kom., M.T., M.M., IPM.

Lahir di Cirebon, Jawa Barat, tanggal 7 Juni 1975. Memiliki Jabatan Fungsional Lektor Kepala, Pembina Tingkat I (IV/b). Berpendidikan Sarjana Komputer (S.Kom.) di Universitas Teknologi *Digital* Indonesia (UTDI) tahun 1998, Magister Manajemen (M.M.) di Universitas Hasanuddin (UNHAS) tahun 2004, Magister Teknologi Informasi (M.T.) di Universitas Gadjah Mada (UGM) tahun 2007, Insinyur (Ir.) di Pendidikan Profesi Insinyur UNHAS tahun 2020, Insinyur Profesional Madya (IPM.) di Persatuan Insinyur Indonesia (PII) tahun 2026, Doktor (Dr.) di Fakultas Teknik UNHAS tahun 2023, Kursus Kader Pimpinan (Suskapin) XXVI Menwa RI tahun 1997, dan Program Pendidikan Reguler Angkatan (PPRA) LX Lemhannas RI tahun 2020.

Menjadi Tenaga Pengajar (Dosen) pada Program Studi Teknik Informatika Fakultas Teknologi Informasi Universitas Atma Jaya Makassar (UAJM). Peraih Poster terbaik DPRM Dikti tahun 2016. Dosen berprestasi IKDKI tahun 2020, 2021, 2024 dan 2025. Pernah menjabat Kepala UPT Komputer, Kepala BAPSI, Wakil Dekan FT, Dekan FT dan FTI, Wakil Rektor III, Ketua Penjaminan Mutu. Tim PAK Dosen dan Asesor BKD UAJM. *Reviewer International Conference* dan Jurnal SINTA. Pemenang Hibah Kemdikbud Penelitian Dosen Pemula, Bersaing, Fundamental, dan Strategi Nasional.

Penulis artikel media massa Tribun Timur, Koinonia, Bisnis Sulawesi, Sesawi.net, OMKNet, Mirifica.net, HidupKatolikCom, KatolikanaTV, Jalan Hidup Katolik, dll. Penulis Buku di Kanisius, Sada Kurnia Pustaka, Aksara Sastra Media, *Future Science*, HEI *Publishing*, Mifandi Mandiri *Digital*, Rey Media Grafika, Widina Salemba, Andi, dan Cendikia Mulia Mandiri. Aktifis organisasi IKA Lemhannas RI LX, IARMI, DPP ISKA, BAPOMI Sulsel, LP3KD Sulsel, IKDKI SulSelTraBar, Komkep KAMS, Komsos KAMS, PUKAT KAMS, TPP KAMS, FMKI KAMS, UPS KAMS,

Pengurus Kebun Sawit Laimbo, FDI, PII Makassar, INAPR, Dewan Keuangan Paroki dan Program Ayo Sekolah Mariso, Animator Laudato Si', dll.

Email Penulis: ntsaptadi@gmail.com

BAB 3
ANALISIS *TRADE-OFF* DAN PENGAMBILAN KEPUTUSAN ARSITEKTUR

Ahmad Budi Trisnawa, S.T., M.Kom.
Universitas Mahakarya Asia, Jakarta

Pengantar Analisis *Trade-Off* Arsitektur

Perancangan arsitektur sistem merupakan proses strategis yang menentukan struktur fundamental, komponen utama, serta interaksi antarbagian dalam sebuah sistem (Vikasari *et al.,* 2025). Dalam praktiknya, arsitektur sistem tidak hanya ditentukan oleh kebutuhan fungsional, tetapi juga oleh berbagai kebutuhan non-fungsional, seperti kinerja, skalabilitas, keamanan, keandalan, dan biaya (Miswadi *et al.,* 2025). Kebutuhan-kebutuhan tersebut sering kali bersifat saling bertentangan, sehingga mustahil untuk memaksimalkan seluruh aspek secara bersamaan.

Gambar 3.1: Analisis *Trade-Off* Arsitektur
Sumber: Diolah Penulis.

Dalam konteks inilah analisis *trade-off* arsitektur menjadi sangat penting. Analisis *trade-off* adalah proses evaluasi sistematis terhadap berbagai alternatif arsitektur dengan mempertimbangkan keuntungan dan konsekuensi dari setiap pilihan desain (Trisnawan *et al.,* 2026). Tujuan utama dari analisis ini bukan untuk mencari solusi yang sempurna, melainkan untuk menemukan keseimbangan terbaik antara berbagai atribut kualitas sesuai dengan prioritas sistem dan tujuan organisasi.

Setiap keputusan arsitektur membawa dampak jangka panjang terhadap siklus hidup sistem, mulai dari tahap pengembangan, implementasi, operasional, hingga pemeliharaan dan pengembangan

lanjutan (Jayusman *et al.,* 2025). Keputusan yang diambil tanpa analisis *trade-off* yang matang berpotensi menimbulkan masalah teknis, peningkatan biaya, serta keterbatasan adaptasi sistem di masa depan.

Dalam sistem modern, khususnya yang berbasis arsitektur terdistribusi, komputasi awan, dan *machine learning*, kompleksitas *trade-off* semakin meningkat (Rahajeng *et al.,* 2025). Sebagai contoh, peningkatan akurasi model prediktif sering kali membutuhkan sumber daya komputasi yang lebih besar, yang berdampak langsung pada biaya dan latensi sistem (Tuti Handayani *et al.,* 2025). Demikian pula, penerapan arsitektur mikro servis dapat meningkatkan skalabilitas dan fleksibilitas, namun menambah kompleksitas integrasi dan pengelolaan sistem (Trisnawan, 2025).

Oleh karena itu, analisis *trade-off* arsitektur berperan sebagai alat bantu pengambilan keputusan yang rasional dan terukur. Dengan pendekatan ini, arsitek sistem dapat mengidentifikasi konsekuensi yang dapat diterima (*acceptable trade-offs*) serta menentukan batasan yang tidak boleh dilanggar (*non-negotiable constraints*). Analisis ini juga memungkinkan penyelarasan antara kepentingan teknis dan kebutuhan bisnis, sehingga arsitektur yang dipilih tidak hanya unggul secara teknologi, tetapi juga memberikan nilai tambah yang berkelanjutan bagi organisasi (Suwanda *et al.,* 2025).

Sebagai fondasi awal dalam pengambilan keputusan arsitektur, pengantar analisis *trade-off* ini menjadi pijakan penting untuk memahami bagaimana dan mengapa keputusan desain tertentu diambil, serta bagaimana keputusan tersebut mempengaruhi performa dan keberlangsungan sistem secara keseluruhan (Delsi Samsumar *et al.,* 2025).

Dimensi *Trade-Off* Dalam Arsitektur Sistem

Dalam perancangan arsitektur sistem, keputusan desain selalu melibatkan pertukaran (*trade-off*) antar berbagai atribut kualitas (Everhard & Wisjhnuadji, 2024). Tidak ada arsitektur yang mampu mengoptimalkan seluruh atribut secara bersamaan. Oleh karena itu, pemahaman terhadap dimensi *trade-off* menjadi kunci dalam menentukan arsitektur yang paling sesuai dengan kebutuhan sistem dan tujuan organisasi.

Gambar 3.2: Dimensi *Trade-Off* Dalam Arsitektur Sistem
Sumber: Diolah Penulis.

1. **Kinerja (*Performance*) Vs Skalabilitas (*Scalability*)**
   Kinerja mengacu pada kemampuan sistem dalam merespons permintaan secara cepat dan efisien, sedangkan skalabilitas berkaitan dengan kemampuan sistem untuk menangani peningkatan beban kerja (Joseph Teguh Santoso, 2024).
   a. Arsitektur yang sangat dioptimalkan untuk kinerja, seperti sistem monolitik dengan komunikasi internal minimal, sering kali sulit di skalakan.
   b. Sebaliknya, arsitektur terdistribusi atau berbasis *microservices* lebih mudah diskalakan, namun dapat mengalami latensi akibat komunikasi antar layanan.

   Trade-off ini menuntut perancang sistem untuk menentukan apakah sistem lebih membutuhkan respons cepat atau kemampuan pertumbuhan jangka panjang (Migunani, 2022).

2. **Akurasi (*Accuracy*) Vs Kompleksitas (*Complexity*)**
   Dalam sistem berbasis data dan *machine learning*, peningkatan akurasi model biasanya sejalan dengan meningkatnya kompleksitas algoritma dan kebutuhan komputasi (Siti Maesaroh *et al.,* 2024).

Daftar Pustaka

Afrizal Zein, Dahlan Susilo, Mustakim, Ryan Effendi, Winny Purbaratri, Achmad Ridwan, Subhan Nooriansyah, Faridatun Nadziroh, Anyan, & Ali Ibrahim. (2023). *Konsep Dasar Rekayasa Perangkat Lunak*. Yayasan Cendikia Mulia Mandiri.

Delsi Samsumar, L., Firdaus, M., Septiana Windyasari, V., Rachendu, S., Anwar, C., Asy Syifa Nurul Haq, F., Bakti, I., Arina Romli, N., Nurnaningsih, D., Budi Trisnawan, A., Yulianti, B., Halim Mursyidin, I., Wayahdi, M. R., Setiyani, H., Kuswoyo, D., & Kusmaningrum, A. (2025). *Sistem Informasi Manajemen: Strategi, Desain, dan Penerapan (1st ed.).* Hadla Media Informasi. www.media.hadlacorp.com.

Edy. (2024). *Rekayasa Perangkat Lunak*. Universitas Buddhi Darma.

Everhard, J., & Wisjhnuadji, T. (2024). *Arsitektur Komputer*. Penerbit Deepublish Digital.

Fitria Nur Hasanah, & Rahmania Sri Untari. (2020). *Rekayasa Perangkat Lunak (1st ed.).* Umsida Press.

Jayusman, Y., Hadikusumo, R. A., Anggoro, T., Trisnawan, A. B., Mulyana, D. I., Budiman, D. A., Kusjani, A., Alpiyasin, F., Widaretna, T., Oktavia, O., & Emalia, L. (2025). *Pengantar Teknologi Informasi (1st ed.).* Langit Kata Publisher.

Joseph Teguh Santoso. (2024). *Teori & Problem RPL (Rekayasa Perangkat Lunak)*. Yayasan Prima Agus Teknik.

Marwondo, & Rini Melati. (2023). *Dasar-Dasar Pengembangan Perangkat Lunak dan Gim (1st ed.).* Kementrian Pendidikan, Kebudayaan, Riset, dan teknologi. https://buku.kemdikbud.go.id.

Migunani. (2022). *Rekayasa Perangkat Lunak*. Yayasan Prima Agus Teknik.

Miswadi, Ahmad Budi Trisnawan, Muhaimin Hasanudin, Tuti Handayani, Chairul Anwar, Imam Zaenuddin, M. Rhifky Wayahdi, Dedy Alamsyah, Rhmat Hartono, Eka Prasetya Adhy Sugara, Fahmi Ruziq, Prayogo, Arif Riyandi, A. Taqwa Martadinata, & Suwandono. (2025). *Buku Ajar Rekayasa Perangkat Lunak: Prinsip, Praktik, dan Teknologi Modern (1st ed.).* Hadla Media Informasi.

Muhammad Abdillah Syam, M. Abyan Zhafran Wijaya, Lulu Nuha Khalisah, Mhd. Akbar Bathnul Wadi Nst, & Yahfizham. (2024).

Macam Dan Fungsi Perangkat Lunak Yang Perlu Dipahami Anak Muda Masa Kini. *2*(1), 85–98.

Mursalim Tonggiroh, Victor Benny Alexsius Pardosi, Basiroh, & Fifto Nugroho. (2024). *Rekayasa Perangkat Lunak*. Mafy Media Literasi Indonesia.

Prasetyo, K. W. (2023). Konsep Ontologi Pada Manajemen Rekayasa Perangkat Lunak: Kajian Tentang Perkembangan dan Tantangannya. *ELANG: Journal of Interdisciplinary Research*, 1–9.

Rahajeng, E., Wahyudin, M. I., Tamriesfatno, S., Trisnawan, A. B., Mulyana, Y., Prayudani, S., & Martani, A. (2025). *Informatika Untuk Pemula: Panduan Menuju Era Teknologi (1st ed.).* PT. Sister Books Press. https://sisterpress.id/.

Safa Nadia Bakri, & Muhammad Irwan Padli Nasution. (2024). Penerapan Metodologi Rekayasa Perangkat Lunak untuk Efisiensi Pengembangan Sistem. *JSITIK: Jurnal Sistem Informasi Dan Teknologi Informasi Komputer*, *3*(1), 53–66. https://doi.org/10.53624/jsitik.v3i1.542.

Satriawaty Mallu, Jeprianto, Satya A Hendrawan, Rifka Widyastuti, Diky Wardhani, Sitti Arni, Suyono, M Rhifky Wayahdi, Kurnia Yahya, Dita Nurmadewi, Mardiyanto, & Samsul Arifin. (2023). *Rekayasa Perangkat Lunak (1st ed.).* PT. Mifandi Mandiri Tunggal.

Siti Maesaroh, Dedy Iskandar, Meri Mayang Sari, Erna Astriyani, Martono, Norbertus Tri Suswanto Saptadi, Fifit Alfiah, Mujibbur Rohman, Arif Muhammad Nurdin, Nur Azizah, Ade Setiadi, Rio Wirawan, Oleh Soleh, Euis Nur Fitriani Dewi, & Saryani. (2024). *Rekayasa Perangkat Lunak*. Sada Kurnia Pustaka.

Soleman, P., Dwi Retnoningsih, M., Arnes Yuli Vandika, M., & Fuadi, A. (2024). *Inovasi Terbaru Dalam Rekayasa Perangkat Lunak Ilmu Komputer*. Mutiara Intelektual Indonesia. www.MII-Press.com.

Sumirat, L. P., Cahyono, D., Kristyawan, Y., & Kacung, S. (2023). *Dasar-Dasar Rekayasa Perangkat Lunak (1st ed.).* Madza Media. www.madzamedia.co.id.

Susilawati, T., & Trisnawan, A. B. (2025). Pemanfaatan Machine Learning untuk Peningkatan Akurasi Sistem Pendukung Keputusan Prediktif. *Jurnal Unitek*, *18*(2), 303–312.

Suwanda, R., Nugroho, A. Y., Candra, D. G. A., Rustiyana, R., Fitriyanto, I., Prapcoyo, H., Pasinggi, E. S., Jinan, A., Putra, K. O., Ritonga, R. D., Khusna, T. N., Muis, A., Bahri, S., Trisnawan, A. B., & Habibullah, R. (2025). *Ilmu Komputer dan Teknologi Informasi: Pengenalan Untuk Pemula (1st ed.).* CV. MMFast Publishing. http://mmfast.id/.

Syahminan. (2020). Pengembangan Pembelajaran Teknik Digital Dengan Media Perangkat Lunak Proteus dan Emulator Jurusan Teknik Informatika Universitas Kanjuruhan. *Jurnal SPIRIT*, *12*(2), 41–45.

Trisnawan, A. B. (2025). Analisis Efektivitas Algoritma Komputasi Pada Sistem Pendukung Keputusan. *Telcomatics*, *10*(1), 82–85. https://doi.org/10.37253/telcomatics.v10i1.11022.

Trisnawan, A. B., Sholikhan, M., & Koerniawan, I. (2026). Analyzing the Role of Enterprise Information Systems in Driving Organizational Innovation: A Multi Method Study. *Information System Analysis, Design And Development*, *1*(1), 41–50. https://journal.apjikom.or.id/index.php/ISADD.

Tuti Handayani, Intan Murniasih, Lukman Medriavin Silalahi, Soma Setiawan Ponco Nugroho, Agung Yuliyanto Nugroho, Chairul Anwar, Imam Halim Mursyidin, Asep Sumantri, Devit Setiono, Budi Berlinton Sitorus, Ahmad Budi Trisnawan, Doni Prastyo, El Vionna Laellyn Nurul Fatich, Imam Zaenuddin, M. Rhifky Wayahdi, Rismen Sinambela, Prima Yustitia Nurul Islami, & Bekti Yulianti. (2025). *Pengantar Sistem Informasi: Konsep, Teknologi, dan Implementasi (1st ed.).* Hadla Media Informasi. www.media.hadlacorp.com.

Usman Ependi. (2024). *Rekayasa Perangkat Lunak: Wawasan Dari Teori Ke Praktik*. Asosiasi Doktor Sistem Informasi Indonesia.

Vikasari, C., Suryani, S., Zein, M. T. A. A., Faizal, F., Trisnawan, A. B., Budiman, D. A., Apriyanti, L., Rahayu, M. I., Ibrahim, R. N., Utami, F. H., & Nugraha, M. F. (2025). *Rekayasa Perangkat Lunak (1st ed.).* CV. Langit Kata Publisher.

Widiyawati, Nazaruddin Ahmad, Eka Hartati, Erly Krisnanik, Yuiansyah, Yunita Ardilla, Iin Ernawati, Guntoro, Kraugusteeliana, I Waya Widi Pradnyana, Titus Kristanto, D Tri Octafian, & Irwanto. (2022). *Rekayasa Perangkat Lunak (1st ed.).* Penerbit Widina Bhakti Persada. www.penerbitwidina.com.

Zatin Noqotaini, Indah Purnamasari, Cholid Fauzi, Yoga Sahria, Dartono, Dian Nursantika, Ida Afriliana, Cahyo Prihantoro, Petrus Christo, Andi Wijaya, Anang Anggono Lutfi, Mohammad Robihul Mufid, Arif Rizki Marsa, & Yuni Widastiwi. (2023). *Rekayasa Perangkat Lunak (1st ed.).* Penamuda Media.

PROFIL PENULIS

Ahmad Budi Trisnawa, S.T., M.Kom.

Ketertarikan penulis terhadap ilmu komputer dimulai pada tahun 2010 silam. Hal tersebut membuat penulis memilih untuk masuk ke Sekolah Menengah Kejuruan di SMK Negeri 2 Kota Tangerang dan berhasil lulus pada tahun 2010. Penulis kemudian melanjutkan pendidikan ke Perguruan Tinggi dan berhasil menyelesaikan studi S1 pada prodi Teknik Informatika Universitas Satya Negara Indonesia pada tahun 2014.

Lima tahun kemudian, penulis menyelesaikan studi S2 pada Prodi Ilmu Komputer Program Pasca Sarjana Universitas Budi Luhur. Penulis memiliki dua (2) buah hati yakni Ardanu Fatih Trisnawan dan Bahira Freya Trisnawan dari pasangan Budi Lestiarini, S.E. Penulis memiliki kepakaran dibidang Sistem Informasi, *Data Mining*, dan *Big Data*. Guna mewujudkan karir sebagai dosen profesional, penulis pun aktif sebagai peneliti di bidang kepakarannya tersebut. Selain peneliti, penulis juga aktif menulis buku dengan harapan dapat memberikan kontribusi positif bagi bangsa dan negara yang sangat tercinta ini.

Email Penulis: abudit75@gmail.com.

BAB 4
ARSITEKTUR BERLAPIS (*LAYER ARCHITECTURE*)

Budi Chehabudin, M.Kom.
Universitas Garut

Definisi dan Konsep Dasar Arsitektur Berlapis

Perkembangan sistem komputer termasuk aplikasi dari generasi ke generasi menghadapi tantangan yang semakin kompleks, mulai dari peningkatan volume data hingga kebutuhan akan integrasi dengan sistem lain. Awalnya, sistem dibangun dengan struktur yang terpisah dengan antarmuka pengguna, logika bisnis, serta akses data semuanya berada dalam satu modul.

Kondisi tersebut membuat proses pengembangan dan perbaikan menjadi lebih sulit yang diakibatkan dari perubahan pada satu bagian yang berdampak ke seluruh sistem. Oleh karena itu, konsep arsitektur berlapis muncul sebagai jawaban atas masalah tersebut dengan mengadopsi prinsip pemisahan fungsi untuk mengurangi ketergantungan antar komponen.

Dalam pengembangan perangkat lunak, arsitektur berlapis (AL) menurut Mrabet dkk (2020) atau dikenal sebagai gaya arsitektur n-*tier* menjadi pilihan umum dalam pengembangan perangkat lunak (Ramdhani, 2024). Sejak tahun 1980-an, arsitektur berlapis telah berevolusi secara substansial yang berawal dari sistem aplikasi dua tingkat berkembang menjadi model yang lebih canggih dan mampu memisahkan antara logika bisnis, interaksi pengguna, serta manajemen data (*System Design School,* n.d.). Pola arsitektur berlapis telah menjadi model desain dasar dalam pengembangan perangkat lunak. Dimana pola ini mengorganisir ke dalam lapisan-lapisan yang berbeda dengan tanggung jawab masing-masing atas serangkaian tugas tertentu.

Pemisahan ini dapat menyederhanakan pengembangan, pemeliharaan, dan skalabilitas sekaligus mendorong modularitas (Cherif, 2024). Menurut Fowler (2018), arsitektur berlapis adalah pola desain perangkat lunak yang mengorganisir sistem menjadi serangkaian lapisan horizontal, di mana setiap lapisan melakukan fungsi spesifik dan hanya berkomunikasi dengan lapisan di atas atau di bawahnya.

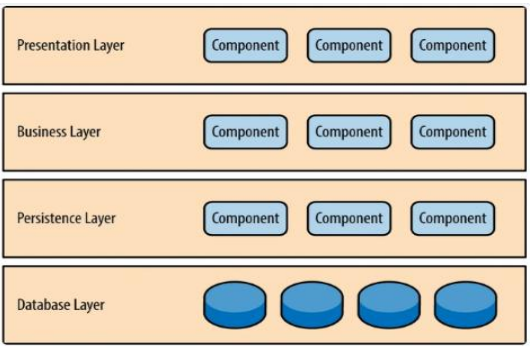
Sedangkan Pressman dan Maxim (2019) mendefinisikannya sebagai struktur yang memisahkan logika sistem menjadi beberapa tingkat abstraksi, di mana setiap tingkat memberikan layanan pada

tingkat berikutnya. Menurut Shaw dan Garlan (1996), arsitektur berlapis juga dapat diartikan sebagai cara untuk mengelola kompleksitas dengan membatasi interaksi antar komponen melalui batasan yang jelas antara lapisan.

Lapisan Dalam Pengembangan Perangkat Lunak

Pada umumnya, arsitektur berlapis terdiri dari beberapa lapisan dengan serangkaian tanggung jawab tertentu. Dimana setiap lapisan pada dasarnya adalah abstraksi yang menjalankan serangkaian fungsi spesifik dan hanya dapat berinteraksi dengan lapisan yang berdekatan. Tidak hanya itu, enkapsulasi juga meningkatkan modularitas dan kemampuan penggunaan kembali dan menciptakan sistem perangkat lunak yang skalabel (*System Design School*, n.d.). Berikut ini adalah lapisan umum dalam arsitektur berlapis, yaitu:

1. **Lapisan Presentasi**
   Lapisan presentasi adalah lapisan yang menyediakan antarmuka pengguna (UI), menampilkan data pengguna akhir, dan mengumpulkan masukan pengguna (*AppMaster*, n.d.). Contoh: Halaman web, antarmuka aplikasi seluler, dan GUI *desktop*.
2. **Lapisan Bisnis**
   Lapisan ini bisa disebut sebagai otak dari setiap perangkat lunak yang berada di antara titik interaksi pengguna dan manajemen data. Tidak hanya itu, lapisan ini berfungsi untuk mengatur bagaimana data diproses dan dijalankan sesuai dengan kebutuhan aplikasi (Lestary, 2025). Contoh: logika validasi, perhitungan, dan algoritma inti.
3. **Lapisan Persistensi**
   Lapisan ini bertanggung jawab untuk memastikan data yang disimpan dan dipertahankan dalam jangka waktu yang lama sesuai kebutuhan (Lestary, 2025). Contoh: *query* basis data, *caching*, dan API.
4. **Lapisan Basis Data**
   Lapisan ini berisi sistem penyimpanan data aktual sebagai tempat informasi disimpan (Cherif, 2024). Contoh: basis data relasional (SQL *Server*).



Gambar 4.1: Lapisan Utama Arsitektur Berlapis
Sumber: Diolah Penulis.

Berdasarkan gambar tersebut, pola ini diadopsi karena selaras dengan bagaimana tim di organisasi IT yang biasanya terstruktur seperti *front-end, back-end,* dan administrator basis data (Cherif, 2024).

Fitur Utama Arsitektur Berlapis

Berikut ini adalah fitur-fitur utama untuk pengembangan aplikasi, yaitu (Cherif, 2024):

1. **Pemisahan Kepentingan**
   Adalah prinsip dasar dari arsitektur berlapis dengan pemisahan yang ketat untuk menyederhanakan pemeliharaan. Oleh karena itu, perubahan pada satu lapisan tidak akan berdampak pada lapisan lainnya. Lapisan pada fitur ini meliputi lapisan presentasi, bisnis, persistensi, dan basis data.
2. **Lapisan Isolasi**
   Pada lapisan ini memastikan bahwa ketergantungan diminimalkan untuk perubahan pada satu lapisan yang hanya memengaruhi lapisan berdekatan. Selain itu, lapisan ini juga memastikan setiap lapisan lain melalui antarmuka yang telah ditentukan.
3. **Lapisan Tertutup dan Terbuka**
   Lapisan tertutup untuk menerapkan interaksi melalui lapisan yang berada di bawahnya. Sedangkan, lapisan terbuka untuk mengizinkan akses langsung melewati beberapa lapisan jika diperlukan.

Daftar Pustaka

AppMaster. (n.d.). *Arsitektur Berlapis*. AppMaster. https://appmaster.io/id/glossary/arsitektur-berlapis.

Binus. (2025). *Software Architecture-Layered Architecture*. Binus Univesity Bekasi. https://binus.ac.id/bekasi/2025/07/software-architecture-layered-architecture/#:~:text=Kelebihan Layered Architecture Modularitas: Memudahkan Pemeliharaan Dan, Diskalakan Karena Sistem Dibagi Menjadi Komponen-Komponen Terpisah.

Cherif, Y. (2024). *Understanding The Layered Architecture Pattern: A Comprehensive Guide*. DEV Community. https://dev.to/yasmine_ddec94f4d4/understanding-the-layered-architecture-pattern-a-comprehensive-guide-1e2j.

Fowler, M. (2018). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

Lestary, E. S. I. (2025). *Kenali Arsitektur Layered dan Manfaatnya Untuk Developer! Tim Tanggap Insiden Siber-Universitas Teknokrat Indonesia.* https://csirt.teknokrat.ac.id/kenali-arsitektur-layered-dan-manfaatnya-untuk-developer/.

Pressman, R. S., & Maxim, B. R. (2019). *Software Engineering: A Practitioner's Approach (9th ed.).* McGraw-Hill Education.

Ramdhani, T. I. (2024). Kajian Arsitektur Berlapis Untuk Pengembangan Aplikasi Dalam Mendukung Pembangunan Aplikasi Di Pemerintahan Provinsi Jawa Barat. *Creative Research for West Java Development (CR Journal)*, *10*(1), 17–26.

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on An Emerging Discipline.* Prentice Hall.

System Design School. (n.d.). *Understanding Layered Software Architecture*. System Design School. https://systemdesignschool.io/blog/layered-software-architecture.

Yonkeu, S. (2024). *Dissecting Layered Architecture*. DEV Community. https://dev.to/yokwejuste/dissecting-layered-architecture-2ppb.

PROFIL PENULIS

Budi Chehabudin, M.Kom.

Ketertarikan penulis terhadap ilmu komputer dimulai pada tahun 2004 silam. Hal tersebut membuat penulis memilih untuk masuk ke STMIK LPKIA Bandung dengan memilih jurusan Manajemen Informatika dan berhasil lulus diploma pada tahun 2008. Penulis kemudian melanjutkan pendidikan ke program sarjana di Perguruan Tinggi yang sama dan berhasil menyelesaikan studi S1 di jurusan Sistem Informasi STMIK LPKIA Bandung pada tahun 2009.

Enam tahun kemudian, penulis menyelesaikan studi S2 di jurusan Sistem Informasi Program Pasca Sarjana STMIK LIKMI. Penulis memiliki kepakaran dibidang *Web Technology* dan *Database*. Pernah mengajar di beberapa perguruan tinggi. Guna mewujudkan karir sebagai dosen profesional, penulis pun aktif sebagai peneliti di bidang kepakarannya tersebut. Beberapa penelitian yang telah dilakukan di perguruan tinggi. Selain peneliti, penulis juga menulis buku dengan harapan dapat memberikan kontribusi positif bagi bangsa dan negara yang sangat tercinta ini. Atas dedikasi dan kerja keras dalam menulis buku mudah-mudahan bisa diterima oleh praktisi maupun bagi pemula.

Email Penulis: chehacool@gmail.com.

BAB 5
ARSITEKTUR
MICROSERVICES

Hendri Julian Pramana, S.Kom., M.Kom.
Universitas Garut

Pendahuluan

Pernahkah Anda membayangkan bagaimana raksasa teknologi seperti *Netflix*, *Amazon*, atau aplikasi *super-app* Gojek melayani jutaan pengguna secara bersamaan tanpa mengalami kegagalan sistem yang fatal setiap harinya?. Pencapaian tersebut tentu tidak diraih dengan sekadar mengandalkan satu komputer super canggih yang menjalankan sebuah program tunggal berukuran raksasa.

Sebagai sebuah analogi dalam tata kota, perusahaan-perusahaan tersebut tidak membangun sebuah *"benteng batu raksasa"* yang kaku dan sulit dipugar. Sebaliknya, pendekatan arsitektur yang mereka gunakan lebih menyerupai pembangunan sebuah *"kota metropolitan"* yang dinamis, terdiri dari ribuan bangunan kecil yang mandiri. Setiap bangunan memiliki fungsi yang spesifik, namun saling terhubung dan berkolaborasi secara harmonis. Konsep tata ruang inilah yang menjadi esensi dasar dari apa yang kita sebut sebagai Arsitektur *Microservices*.

Dalam bab ini, kita akan mempelajari arsitektur *microservices* tidak hanya sebagai tren teknologi, tetapi sebagai solusi atas tantangan kompleksitas dan skalabilitas yang dihadapi oleh sistem modern. Mengapa ini penting? Karena dalam dunia pengembangan perangkat lunak saat ini, satu-satunya hal yang pasti adalah *perubahan*. Bisnis menuntut fitur baru dirilis lebih cepat, pengguna menuntut aplikasi yang tidak pernah *down*, dan data terus bertumbuh secara eksponensial. Arsitektur tradisional seringkali kewalahan menghadapi dinamika ini.

Evolusi Arsitektur Perangkat Lunak

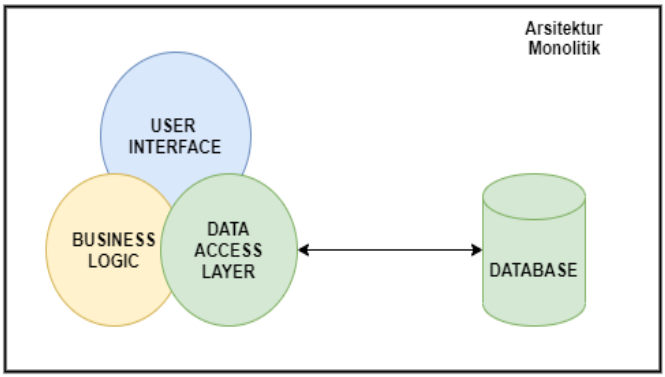
Mari kita melakukan sedikit perjalanan melintasi waktu. Layaknya organisme hidup yang berevolusi untuk beradaptasi dengan lingkungan yang semakin ekstrim, arsitektur perangkat lunak juga mengalami fase evolusi yang menakjubkan.

Evolusi ini bukanlah tentang arsitektur yang satu secara mutlak lebih buruk dari yang lain, melainkan tentang pencarian bentuk yang paling tepat untuk menjawab tuntutan skala bisnis yang terus membesar. Mari kita telaah tiga fase utama dalam evolusi ini: Monolitik, *Service-Oriented Architecture* (SOA), dan *Microservices*.

1. Arsitektur Monolitik

Sebelum ada *microservices,* penguasa tunggal dunia pengembangan aplikasi adalah Arsitektur Monolitik. (Newman, 2020) dalam

bukunya *Monolith to Microservices* mendefinisikan monolitik sebagai gaya rancangan di mana seluruh komponen perangkat lunak (antarmuka pengguna, logika bisnis, dan akses data) digabungkan menjadi satu unit yang tak terpisahkan dan dieksekusi sebagai satu proses tunggal (*single deployable unit*).



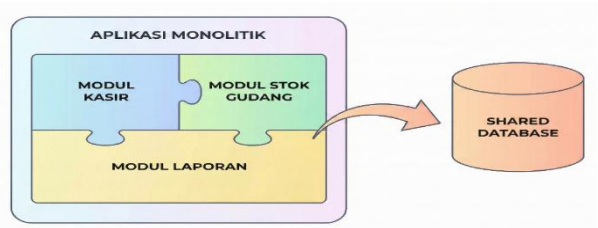
Gambar 5.1: Arsitektur Monolitik
Sumber: https://hellodit.medium.com.

Karakteristik utama dari arsitektur ini meliputi:

a. Satu Basis Kode (*Single Codebase*): seluruh baris kode program berada dalam satu repositori yang sama.
b. Satu *Deployment Unit*: aplikasi di-kompilasi dan di-*deploy* (diterbitkan) sebagai satu paket utuh (misalnya satu *file .war* pada Java atau satu *executable file*).
c. Satu *Database* Terpusat *(Centralized Database)*: semua fitur atau modul di dalam aplikasi menggunakan satu basis data (*database*) yang sama untuk menyimpan informasi.
d. Keterikatan Kuat (*Tight Coupling*): modul-modul di dalamnya saling memanggil secara langsung melalui level memori atau fungsi bawaan bahasa pemrograman, sehingga sangat bergantung satu sama lain.

Ilustrasi Pada Sistem POS Warung: bayangkan aplikasi Warung Berkah versi 1.0. Di dalamnya terdapat `Modul Kasir`, `Stok Gudang`, dan `Laporan`. Ketiganya dibungkus dalam satu file `.jar` (jika menggunakan Java) atau satu folder proyek Express.js. Jika Pak Budi ingin mengubah fitur diskon di `modul Kasir`,

pengembang harus merakit ulang (*rebuild*) dan menyebarkan ulang (*redeploy*) seluruh aplikasi, termasuk `modul Gudang` dan `Laporan` yang sebenarnya tidak berubah.



Gambar 5.2: Arsitektur Monolitik Sistem Warung
Sumber: *Generate by Gemini* AI.

Kelebihan pendekatan ini adalah kesederhanaannya. Untuk tim kecil atau *startup* tahap awal, monolitik adalah pilihan cerdas karena sangat mudah untuk dikembangkan (*develop*), diuji (*test*), dan disebarkan (*deploy*) (Newman, 2021). Namun ketika sistem mulai bertumbuh, fitur semakin banyak dan tim pengembang semakin membesar, monolitik perlahan berubah menjadi jebakan yang menyulitkan. (Richardson, 2019) merangkum masalah ini sebagai "Neraka Monolitik" (*Monolithic Hell*). Tantangan utama yang muncul antara lain:

a. Skalabilitas Terbatas: jika hanya modul "Laporan" yang membutuhkan spesifikasi *server* tinggi, anda tetap harus memperbesar (*scale-up*) seluruh aplikasi. Anda tidak bisa membesarkan satu modul saja. Ini adalah bentuk pemborosan sumber daya komputasi (*cloud cost*).
b. Risiko Kegagalan Menyeluruh (*Single Point of Failure*): kesalahan kecil (*bug*) pada satu modul, misalnya kebocoran memori pada fitur cetak struk, dapat meruntuhkan seluruh aplikasi kasir secara total.
c. Sulit Melakukan Perubahan Tanpa *Downtime*: setiap kali ada pembaruan fitur sekecil apapun, seluruh aplikasi harus dimatikan sesaat (*restart*).
d. Kompleksitas Kode Meningkat: basis kode yang raksasa membuat pengembang baru kebingungan dan memperlambat proses adaptasi (*onboarding*).

Daftar Pustaka

Choudhary, M. (2025, October 7). *What Are Microservices? How Microservices Architecture Works*. https://middleware.io/blog/microservices-architecture/.

Fowler, S. J. (2017). *Production-Ready Microservices*. O'Reilly Media, Inc.

GeeksforGeeks. (2026, January 21). *Microservices-GeeksforGeeks*. https://www.geeksforgeeks.org/system-design/microservices/.

Larsson, M. (2023). *Microservices With Spring Boot 3 and Spring Cloud*.

Newman, S. (2020). *Monolith to Microservices*. O'Reilly Media, Inc. https://learning.oreilly.com/videos/monolith-to-microservices/1492047848/1492047848-1492047848_002_MP3/.

Newman, S. (2021). Building Microservices 2nd Designing Fine-Grained Systems. In *O'Reilly*. O'Reilly Media, Incorporated. https://www.google.hr/books?hl=en&lr=&id=jjl4BgAAQBAJ&pgis=1%5Cnhttp://oreilly.com/catalog/errata.csp?isbn=9781491950357.

Oyeniran, O. C., Adewusi, A. O., Adeleke, A. G., Akwawa, L. A., & Azubuko, C. F. (2024). Microservices Architecture In Cloud-Native Applications: Design Patterns And Scalability. *Computer Science & IT Research Journal*, *5*(9), 2107–2124. https://doi.org/10.51594/csitrj.v5i9.1554.

Papazoglou, M. P., & Van Den Heuvel, W.-J. (2007). Service Oriented Architectures: Approaches, Technologies And Research Issues. *The VLDB Journal*, *16*(3), 389–415. https://doi.org/10.1007/s00778-007-0044-3.

Prasetya, A. (2020, December 27). *Belajar Microservices: Pengenalan*. https://hellodit.medium.com/belajar-microservices-pengenalan-96c3bb66cf2c.

Prasetya, A. (2022, February 17). *Komunikasi Antar Service Pada Microservice | by Asdita Prasetya | Medium*. https://hellodit.medium.com/komunikasi-antar-service-pada-microservice-43d57be18996.

Richardson, C. (2019). *Microservices Patterns; With Example In JAVA*. Manning Publications Co.

Suryotrisongko, H. (2017). Arsitektur Microservice Untuk Resiliensi Sistem Informasi. *SISFO*, *06*(02), 235–250.

Wardana, B., & Hermanto, H. (2024). Implementasi Microservices di Situs Web Frontend. *Jurnal Pengembangan Teknologi Informasi Dan Komunikasi (JUPTIK)*, *2*(1), 24–27. https://doi.org/10.52060/juptik.v2i1.2212.

PROFIL PENULIS

Hendri Julian Pramana, S.Kom., M.Kom.
Adalah seorang akademisi di bidang pengembangan Rekayasa Perangkat Lunak (RPL). Ketertarikan penulis terhadap dunia komputer dan teknologi sudah tumbuh sejak masa sekolah menengah pertama. Belajar secara otodidak dan lembaga-informal yang terbatas tidak menghalangi keinginan untuk melanjutkan pendidikan tinggi di bidang Teknik Informatika setelah lulus Sekolah Menengah Atas di Kota Tasikmalaya.

Penulis berhasil menyelesaikan studi Sarjana Komputer dan kemudian melanjutkan pendidikan Magister Komputer pada kampus UDINUS Semarang dan lulus pada tahun 2020. Saat ini, penulis menjadi dosen tetap untuk Prodi Rekayasa Perangkat Lunak, Fakultas Komunikasi dan Informasi (FKOMINFO), Universitas Garut (UNIGA). Penulis mengampu beberapa mata kuliah di antaranya: Arsitektur Perangkat Lunak, Sistem Terdistribusi, Pemrograman Bergerak, Sistem Pendukung Keputusan, Peretasan Beretika (*Ethical Hacking*), dan *Design Thinking*. Fokus dan ketertarikan penulis dalam bidang keilmuan meliputi *design thinking*, *decision support system*, *machine learning*, pengenalan objek menggunakan *computer vision*, serta *Internet of Things* (IoT). Selain mengajar, penulis juga berkomitmen untuk terus aktif melakukan penelitian dan meningkatkan pengetahuan melalui kegiatan menulis buku guna memperkuat kontribusi pada dalam pengembangan pengetahuan dan teknologi di Indonesia.

Email Penulis: hendri.jp@uniga.ac.id.

BAB 6
KEAMANAN DALAM ARSITEKTUR (*SECURITY BY DESIGN*)

Kotim Subandi, S.Kom, M.T.
Universitas Pakuan

Mengapa Keamanan Adalah Masalah Arsitektur?

Dalam pengembangan perangkat lunak tradisional, keamanan sering kali diperlakukan sebagai fitur tambahan yang disisipkan di akhir fase pengembangan, atau sekadar menjadi tanggung jawab tim infrastruktur firewall dan jaringan. Namun, dalam lanskap ancaman siber modern yang kompleks, pendekatan ini tidak lagi memadai.

Keamanan bukanlah lapisan cat yang bisa dioleskan setelah rumah selesai dibangun, namun keamanan adalah struktur beton, tata letak pintu, dan fondasi dari bangunan itu sendiri. Dalam konteks arsitektur perangkat lunak, keamanan harus bersifat *intrinsic* yang melekat dan diterapkan melalui pendekatan *Security by Design* (Al-Qerem, 2024).

Seorang arsitek perangkat lunak bertanggung jawab untuk merancang sistem yang tidak hanya fungsional dan cepat, tetapi juga tangguh (*resilient*) terhadap serangan, melindungi data pengguna, dan mematuhi regulasi privasi. Kegagalan dalam merancang keamanan di tingkat arsitektur dapat menyebabkan kerentanan sistemik yang sulit dan mahal untuk diperbaiki di kemudian hari (Alshaikh, 2025).

Fondasi Keamanan Informasi

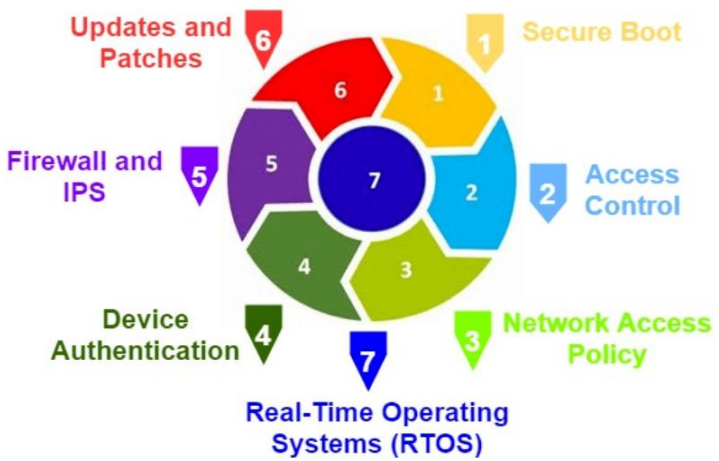
Sebelum menyelami pola arsitektur, kita harus memahami tiga pilar utama keamanan informasi yang dikenal sebagai CIA *Triad*. Setiap keputusan arsitektur harus menyeimbangkan ketiga aspek ini (Bertino, 2024).

Gambar 6.1: Aspek Dalam Keamanan Informasi
Sumber: Diolah Penulis.

1. ***Confidentiality*** **(Kerahasiaan)**
   Menjamin bahwa informasi hanya dapat diakses oleh pihak yang memiliki wewenang. Implementasi Arsitektur Enkripsi data (baik saat diam maupun saat transit), manajemen akses (ACL), dan penyembunyian topologi jaringan internal (CIS, 2024).
2. ***Integrity*** **(Integritas)**
   Menjamin bahwa data akurat, konsisten, dan tidak diubah oleh pihak yang tidak sah.*Implementasi Arsitektur: Hashing*, tanda tangan *digital* (*digital signature*), dan validasi input yang ketat pada setiap lapisan layanan (CNSA, 2023).
3. ***Availability*** **(Ketersediaan)**
   Menjamin bahwa sistem dan data tersedia saat dibutuhkan oleh pengguna yang sah. *Implementasi Arsitektur* Redundansi *server*, *Load Balancing*, perlindungan terhadap serangan DDoS, dan strategi pemulihan bencana (*Disaster Recovery*).

Prinsip Utama *Security By Design*

Konsep keamanan berdasarkan desain menekankan bahwa keamanan bukanlah fitur tambahan *add-on* yang dipasang setelah aplikasi jadi, melainkan aspek yang harus diintegrasikan sejak tahap perencanaan awal hingga implementasi. Untuk mencapai arsitektur yang aman, terdapat beberapa prinsip desain universal yang wajib diterapkan oleh seorang arsitek (Fisher, 2023).



Gambar 6.2: Security By Design
Sumber: Diolah Penulis.

1. **Prinsip Hak Akses Minimum (*Least Privilege*)**
   Setiap komponen sistem *user, service,* atau *database* hanya boleh memiliki hak akses seminimal mungkin yang diperlukan untuk menjalankan fungsinya. Sebuah layanan *microservice payment* hanya boleh mengakses tabel *database* transaksi , dan tidak boleh memiliki akses ke tabel "*Profil User*". Jika layanan ini diretas, kerusakan dapat dilokalisir (Garbis, 2024).

2. **Pertahanan Berlapis (*Defense In Depth*)**
   Jangan pernah bergantung pada satu mekanisme keamanan tunggal. Arsitektur harus memiliki lapisan keamanan yang bertingkat, ibarat lapisan bawang (Gupta, 2024).
   a. Lapisan 1 *Web Application Firewall* (WAF) di tepi jaringan.
   b. Lapisan 2 Otentikasi *Gateway* (API *Gateway*).
   c. Lapisan 3 Validasi *input* pada level aplikasi.
   d. Lapisan 4 Enkripsi *database*. Jika satu lapisan ditembus, lapisan berikutnya masih melindungi sistem.

3. ***Zero Trust Architecture***
   Paradigma lama mengasumsikan bahwa jaringan internal adalah aman dan jaringan eksternal adalah berbahaya. Paradigma *Zero Trust* menolak asumsi ini. Prinsipnya adalah *Never Trust, Always Verify*. Setiap permintaan akses, baik yang berasal dari luar maupun dari dalam jaringan internal, harus di otentikasi, diotorisasi, dan dienkripsi.

4. **Gagal Dengan Aman (*Fail Secure*)**
   Jika sistem mengalami kegagalan *crash* atau *error,* sistem harus berada dalam kondisi tertutup/aman, bukan terbuka. Jika sistem validasi login error karena database mati, sistem harus menolak semua akses masuk, bukan malah mengizinkan semua orang masuk *bypass*. (Kumar, 2025).

Pola Arsitektur Untuk Keamanan Informasi

Kumpulan solusi terstruktur dan berulang yang digunakan untuk menyelesaikan masalah keamanan yang umum terjadi dalam desain sistem. Pola ini membantu arsitek memastikan bahwa sistem tidak

Daftar Pustaka

Al-Qerem, A. e. (2024). A Comprehensive Review of Zero Trust Architecture: Challenges, Opportunities, And Future Directions For IoT Systems. *IEEE Access.*

Alshaikh, M. &. (2025). Human-Centric Cybersecurity: Evaluating the Role of Security Awareness In The Era of Deepfakes. *Computers & Security*, pp. 103-115.

Bertino, E. e. (2024). Zero Trust Architecture: Foundations, Principles, And Future Perspectives. *IEEE Transactions on Dependable and Secure Computing,* 1102-1118.

CIS. (2024). *CIS Critical Security Controls (CIS Controls).* Center for Internet Security.

CNSA, C. &. (2023). *Shifting The Balance of Cybersecurity Risk: Principles And Approaches for Security-by-Design And -Default.* Cybersecurity And Infrastructure Security Agency.

Fisher, D. (2023). *Application Security Program Handbook: A Guide for Software Engineers And Team Leaders.* Manning Publications.

Foundation, O. (2024). *OWASP API Security Top 10 2023.* The Open Web Application Security Project.

Garbis, J. &. (2024). *Zero Trust Security: An Enterprise Guide (2nd Edition).* Apress.

Gupta, M. (2024). *Cybersecurity In The Age of Generative AI: Risks, Challenges, And Defense Strategies.* Apress.

Kumar, P. &. (2025). AI-Driven Threat Detection In Cloud Native Architectures: A Real-time Approach. *International Journal of Information Security (Early Access/Pre-print).*

Newman, S. (2024). *Building Microservices: Designing Fine-Grained Systems. (2nd Edition-Updated Sections on Security).* O'Reilly Media.

Nugroho, A. S. (2024). Evaluasi Keamanan API Gateway menggunakan OWASP API Security Top 10 2023: Studi Kasus Fintech. *Indonesian Journal of Computing and Cybernetics Systems (IJCCS)*, 1-12.

Porambage, P. O. (2024). Survey on Multi-Access Edge Computing For 5G And Beyond. *IEEE Communications Surveys & Tutorials*, 2661-2702.

Poston, H. (2024). *Practical API Architecture And Development With Azure And AWS.* Packt Publishing.

Pratama, I. P. (2024). Analisis Keamanan Arsitektur Microservices Menggunakan Metode STRIDE dan DREAD Pada Aplikasi E-Commerce. *Jurnal Teknik Informatika dan Sistem Informasi (JuTISI)*, 112-125.

Rahman, M. &. (2023). Security Smells In Infrastructure As Code Scripts: A Systematic Literature Review. *ACM Transactions on Software Engineering And Methodology*, 1-35.

Sarker, I. H. (2024). AI-Driven Cybersecurity: An Overview, Security Intelligence Modeling And Research Directions. *SN Computer Science*, 1-18..

Setiawan, B. e. (2024). Perancangan DevSecOps Pipeline untuk Meningkatkan Keamanan Aplikasi Web Berbasis Docker Container. *Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI)*, 201-210.

Stallings, W. (2025). *Cryptography And Network Security: Principles And Practice (9th Edition).* Pearson.

Suehring, S. (2023). *Learning DevSecOps: A Practical Guide to Security In DevOps.* O'Reilly Media.

Vacca, J. R. (2024). *Managing Information Security (3rd Edition).* Syngress.

Verizon. (2024). *Data Breach Investigations Report (DBIR).* Verizon Enterprise Solutions.

Whitman, M. E. (2024). *Management of Information Security (7th Edition).* Cengage Learning.

Wijaya, K. &. (2024). Implementasi Arsitektur Zero Trust Network Access (ZTNA) pada Sistem Informasi Akademik Universitas XYZ. *Jurnal Sistem Informasi dan Telematika (JST), 15(1)*, 45-56.

Zhang, Y. e. (2023). DevSecOps: A Systematic Mapping Study on Security Integration In DevOps. *Journal of Systems And Software*, 198, 111603.

PROFIL PENULIS

Kotim Subandi, S.Kom., M.T.

Salah satu tugas dosen selain melakukan penelitian dan Pengabdian Kepada Masyarakat, yaitu menulis buku ajar. Saya dosen peneliti di bidang Teknologi Informasi di Universitas Pakuan Bogor. Fokus penelitiannya meliputi *big data,* keamanan siber, dan transformasi digital di sektor bisnis *digital*.

Buku ini merupakan bagian dari kontribusinya dalam meningkatkan literasi *digital* di kalangan akademisi dan praktisi di Indonesia. Karena penting buku ajar sebagai pegangan saat mengajar membuat penulis Penulis menempuh pendidikan ke Perguruan Tinggi dan berhasil menyelesaikan studi S1 di prodi Teknik Informatika, kemudian penulis menyelesaikan studi S2 di prodi Rekayasa Manufaktur Program Pasca Sarjana Universitas Pancasila Jakarta. Dengan pengalaman lebih dari 10 tahun sebagai konsultan ISO 27002 untuk berbagai instansi pemerintah dan perusahaan swasta, Saya dikenal sebagai pengajar yang mampu menyederhanakan konsep-konsep teknis menjadi materi yang mudah dipahami. Selain aktif mengajar di perguruan tinggi, ia juga rutin menjadi narasumber dalam seminar dan pelatihan *Cyber Security* di tingkat nasional. Penulis memiliki kepakaran di bidang *Network Security* dan *Data Analyst.* Demi mencapai karir sebagai dosen profesional, penulis pun aktif sebagai peneliti di bidang *Networking*. Beberapa penelitian yang telah dilakukan didanai oleh internal perguruan tinggi dan aktif di dunia industri terkait IT dan ISO 27002.

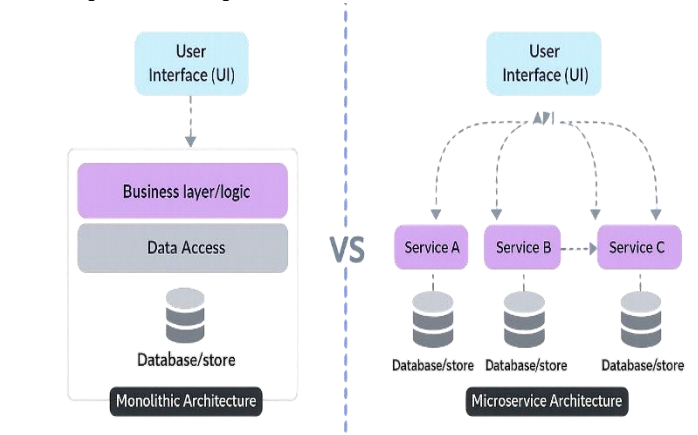
BAB 7
ARSITEKTUR *CLOUD NATIVE* DAN *SERVERLESS*

Dr. Irawan Afrianto, S.T., M.T.
Universitas Komputer Indonesia

Evolusi dan Paradigma *Cloud-Native*

Dunia pengembangan perangkat lunak telah melewati transformasi yang luar biasa dalam kurun waktu satu dekade terakhir. Pada awalnya, standar pembangunan sistem didominasi oleh arsitektur monolitik, sebuah model di mana seluruh komponen fungsional mulai dari antarmuka pengguna, logika pemrosesan, hingga manajemen data dibangun sebagai satu unit kode yang masif dan saling terikat erat.

Namun, seiring dengan meningkatnya kebutuhan akan kecepatan rilis (*time-to-market*) dan skalabilitas yang tinggi, arsitektur monolitik mulai menemui titik jenuh. Keterbatasan utama model ini terletak pada kekakuannya; setiap perubahan kecil pada satu modul mewajibkan pengembang untuk membangun ulang dan menggelar ulang seluruh sistem, yang sering kali menyebabkan risiko kegagalan operasional yang besar. Perbandingan antara arsitektur monolitik dan Mikroservis dapat dilihat pada Gambar 7.1.

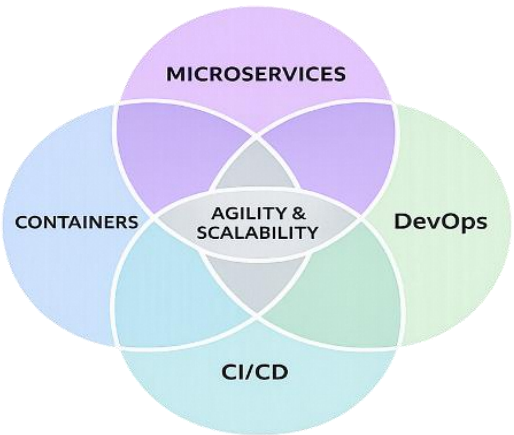


Gambar 7.1: Arsitektur Monolitik vs Mikroservis
Sumber: IcePanel, 2023.

Sebagai jawaban atas tantangan tersebut, muncul paradigma *Cloud Computing* yang menggeser konsep kepemilikan infrastruktur fisik menjadi layanan berbasis kebutuhan (*on-demand*). Menurut Gill *et al*. (2019), evolusi komputasi awan bukan sekadar perpindahan lokasi penyimpanan data, melainkan perubahan mendasar dalam cara sumber daya komputasi dikonsumsi secara elastis.

Di Indonesia, transformasi ini telah menjadi pendorong utama modernisasi infrastruktur digital di berbagai sektor, memungkinkan industri untuk lebih adaptif terhadap dinamika pasar (Nurdin *et al,* 2025). Dari sinilah lahir metodologi *Cloud-Native*, sebuah pendekatan yang dirancang khusus untuk membangun aplikasi yang mampu mengeksploitasi seluruh potensi kelincahan dan ketahanan yang ditawarkan oleh ekosistem awan.

Inti dari keberhasilan ekosistem *cloud-native* terletak pada sinkronisasi empat pilar teknologi utama (Gambar 7.2). Pertama, penggunaan *microservices* yang memecah aplikasi menjadi layanan-layanan kecil independen. Kedua, penerapan kontainerisasi melalui teknologi kontainer yang menjamin konsistensi aplikasi di berbagai lingkungan, mulai dari tahap pengembangan hingga produksi (Rajan, 2020). Ketiga, pengadopsian budaya *DevOps* dan jalur *Continuous Delivery* (CI/CD) yang mengotomatisasi pengiriman kode untuk meminimalkan kesalahan manusia. Terakhir, aspek keamanan kini bertransformasi menuju strategi *Zero Trust*, dimana identitas setiap entitas diverifikasi secara ketat dalam lingkungan yang terdistribusi (Dommari & Khan, 2025). Integrasi keempat elemen ini menciptakan fondasi yang memungkinkan organisasi mencapai standar inovasi global dalam lingkungan yang sangat dinamis (Anwar *et al,* 2025).



Gambar 7.2: Empat Pilar *Cloud-Native*
Sumber: Diolah Penulis.

Daftar Pustaka

Ahmadi, S. (2024). Challenges And Solutions In Network Security For Serverless Computing. *International Journal of Current Science Research and Review*, *7*(01), 218-229.

Akhtar, N., Raza, A., Ishakian, V., & Matta, I. (2020, July). COSE: Configuring Serverless Functions Using Statistical Learning. In IEEE INFOCOM 2020-IEEE conference on computer communications (pp. 129-138). IEEE.

Anwar, C., Ramadhani, G., Aditiya, M. Z., & Sari, P. A. (2025). Pemanfaatan Cloud Computing Untuk Solusi Disaster Recovery Dan Kontinuitas Bisnis Sistem Informasi Utama (Studi Kasus: Universitas Pamulang). *Journal of Information Systems And Business Technology*, *1*(1), 161-166.

Aslanpour, M. S., Toosi, A. N., Cheema, M. A., & Chhetri, M. B. (2024). FaasHouse: Sustainable Serverless Edge Computing Through Energy-Aware Resource Scheduling. *IEEE Transactions on Services Computing*, *17*(4), 1533-1547.

Choi, J., & Lee, K. (2020). Evaluation of Network File System As A Shared Data Storage In Serverless Computing. *WOSC 2020-Proceedings of The 2020 6th International Workshop on Serverless Computing, Part of Middleware 2020*, 25–30. https://doi.org/10.1145/3429880.3430097.

Dash, S., Sodhi, R., & Sodhi, B. (2020). A Serverless Cloud Computing Framework For Real-Time Appliance-Usage Recommendation. *2020 21st National Power Systems Conference (NPSC)*. https://doi.org/10.1109/NPSC49263.2020.9331847.

Dommari, S., & Khan, S. (2023). Implementing Zero Trust Architecture In Cloud-Native Environments: Challenges And Best Practices. *Available at SSRN 5259339*.

Eismann, S., Scheuner, J., Van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., ... & Iosup, A. (2021). The State Of Serverless Applications: Collection, Characterization, And Community Consensus. IEEE Transactions on Software Engineering, 48(10), 4152-4166.

Gill, S. S., & Buyya, R. (2019). Resource Provisioning Based Scheduling Framework For Execution of Heterogeneous And Clustered

Workloads In Clouds: From Fundamental To Autonomic Offering. *Journal of Grid Computing*, *17*(3), 385-417.

Hassan, H. B., Barakat, S. A., & Sarhan, Q. I. (2021). Survey On Serverless Computing. *Journal of Cloud Computing*, *10*(1), 39.

IcePanel. (2023, March 7). Monolithic Vs Microservices Architectures. *IcePanel Blog*. https://icepanel.io/blog/2023-03-07-monolithic-vs-microservices-architectures.

Kelly, D., Glavin, F. G., & Barrett, E. (2021). Denial of Wallet Defining A Looming Threat To Serverless Computing. *Journal of Information Security and Applications,* 60, 102843.

Kritikos, K., & Skrzypek, P. (2018, December). A Review of Serverless Frameworks. *In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)* (pp. 161-168). IEEE.

Li, Z., Tan, Y., Li, B., Zhang, J., & Wang, X. (2021, March). A Survey Of Cost Optimization In Serverless Cloud Computing. *In Journal of Physics: Conference Series* (Vol. 1802, No. 3, p. 032070). IOP Publishing.

Maleh, Y., Shojafar, M., Darwish, A., & Haqiq, A. (Eds.). (2019). *Cybersecurity And Privacy In Cyber Physical Systems*. CRC Press.

Nurdin, A. M., Saptadi, N. T. S., Kristiawan, H., Nurhaeni, T., Yunianto, I., Arisandi, D., ... & Soleh, O. (2024). *Pengantar Teknologi Informasi*. Sada Kurnia Pustaka.

Nurdin, A. M., Faisal, M., Aliyah, A., Prihantoro, C., Dewi, E.N.F., Candra, N.A., ... & Subandi, K. (2025). *Cloud Computing Konsep, Arsitektur, dan Implementasi*. Sada Kurnia Pustaka.

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2017, October). Energy Efficiency Across Programming Languages: How Do Energy, Time, And Memory Relate?. *In Proceedings of the 10th ACM SIGPLAN International Conference On Software Language Engineering* (pp. 256-267).

Putra, C. B. A., Asmoro, D. P. T., & Yulianto, A. B. (2025). Serverless Computing: A Comparative Analysis of Cloud Run And Cloud Function Prices On Google Kubernetes Engine Cluster Node Management. *Jurnal Teknologi Informatika dan Komputer*, *11*(2).

Rajan, A. P. (2020). A Review On Serverless Architectures-Function As A Service (FaaS) In Cloud Computing. *TELKOMNIKA*

(Telecommunication Computing Electronics and Control), *18*(1), 530-537.

Samueljabiodun, (2021, February 14). *The 3 Pillars of System Observability: Logs, Metrics, And Tracing.* Hubofcode. Retrieved February 19, 2026, from https://hubofco.de/softwareengineering/2021/02/14/the-3-pillars-of-system-observability-logs-metrics-and-tracing.

Shafiei, H., Khonsari, A., & Mousavi, P. (2020). Serverless Computing: A Survey of Opportunities, Challenges, And Applications. *ACM Computing Surveys (CSUR)*, *53*(3), 1–35. https://doi.org/10.1145/3382733.

Singh, R. K., Raichura, H., Malhotra, A., Kalra, H., Raghu, N., & Garima. (2025). Optimizing Data Service With Innovative Model For Multi-Cloud Driven Storage Platform. *International Journal of System Assurance Engineering and Management*, 1-10.

The Software House. (2021, February 16). *Why And How Serverless Computing Helps Teams Scale Up* [Graph Illustrating Cost Benefits Of Serverless]. Retrieved February 19, 2026, from https://tsh.io/blog/serverless-computing-scalability.

Xie, R., Tang, Q., Qiao, S., Zhu, H., Yu, F. R., & Huang, T. (2021). When Serverless Computing Meets Edge Computing: Architecture, Challenges, And Open Issues. *IEEE Wireless Communications*, *28*(5), 126-133.

Yussupov, V., Breitenbücher, U., Brogi, A., Harzenetter, L., Leymann, F., & Soldani, J. (2022, July). Serverless or Serverful? A Pattern-Based Approach For Exploring Hosting Alternatives. *In Symposium and Summer School on Service-Oriented Computing* (pp. 45-67). Cham: Springer International Publishing.

PROFIL PENULIS

Dr. Irawan Afrianto, S.T., M.T.
Salah satu dosen di Universitas Komputer Indonesia (UNIKOM) pada program studi Teknik Informatika mulai dari tahun 2003 hingga saat ini. Bidang keilmuan yang didalami adalah keamanan informasi, multimedia, dan teknologi blockchain. Menyelesaikan Pendidikan sarjana Teknik Informatika di Universitas Komputer Indonesia (UNIKOM) Bandung pada tahun 2002, program Magister Teknik Elektro (KK-Teknologi Informasi) di Institut Teknologi Bandung tahun 2010, serta program Doktoral Ilmu Komputer di IPB University pada tahun 2025.

Penulis juga aktif dalam penelitian dan telah menghasil penelitian-penelitian pada lingkup Informatika. Beberapa penelitian yang dilakukan merupakan penelitian yang dibiayai dari program hibah penelitian pemerintah Indonesia melalui DIKTI. Diseminasi hasil penelitian dilakukan dengan mengikuti seminar nasional, internasional, serta publikasi artikel ilmiah pada jurnal bereputasi (nasional dan internasional). Penulis juga aktif dalam kegiatan pengabdian bertemakan transformasi dan penerapan teknologi informasi yang dapat digunakan oleh masyarakat. Salah satu hal yang membanggakan adalah penelitian yang dilakukan pernah mendapatkan penghargaan sebagai penelitian terbaik tahun 2016 dari provinsi Jawa Barat.

Email Penulis: irawan.afrianto@email.unikom.ac.id.

BAB 8
DOKUMENTASI DAN VISUALISASI ARSITEKTUR PERANGKAT LUNAK

Bayu Waseso, M.Kom.
Universitas Mercu Buana

Peran Strategis Dokumentasi Arsitektur

Dokumentasi arsitektur seringkali dipersepsikan sebagai artefak administratif, padahal dalam praktik profesional, ia adalah instrumen strategis yang menentukan keberhasilan komunikasi teknis, kesinambungan sistem, dan kualitas pengambilan keputusan (Kumar, Lokku, *et al.,* 2021; Palma *et al.,* 2022a; Valle *et al.,* 2021).

Dalam konteks organisasi modern, terutama yang mengadopsi *DevOps*, *cloud-native,* dan tim terdistribusi, arsitektur yang tidak terdokumentasi dengan baik akan menghasilkan ketergantungan pada individu (*knowledge silos*), inkonsistensi implementasi, dan kesulitan dalam evolusi sistem. Standar internasional menegaskan bahwa arsitektur bukan hanya struktur sistem, melainkan deskripsi formal yang menghubungkan elemen, relasi, dan rasional keputusan desain.

Oleh karena itu, dokumentasi bukan sekadar gambar diagram, tetapi representasi sistematis yang menjelaskan mengapa dan bagaimana sistem dirancang (Martin, 2021).

1. Tujuan Dokumentasi Arsitektur

Dokumentasi arsitektur memiliki fungsi strategis yang melampaui sekadar pencatatan desain. Ia berperan sebagai mekanisme kontrol, komunikasi, dan legitimasi keputusan teknis dalam organisasi (Kumar, Anand, *et al*., 2021; Palma *et al.,* 2022b). Berikut adalah lima fungsi utama dokumentasi arsitektur beserta penjelasan ringkasnya.

a. Mendukung Komunikasi Antar-*Stakeholder*

Arsitektur adalah representasi abstrak sistem yang harus dapat dipahami oleh berbagai peran dengan latar belakang berbeda. Dokumentasi berfungsi sebagai media komunikasi formal agar interpretasi desain tidak bergantung pada persepsi individu.

1) Mengurangi Ambiguitas Interpretasi Desain

Diagram dan deskripsi eksplisit mencegah asumsi yang berbeda antar tim terhadap struktur dan batas sistem.

2) Menjadi Referensi Diskusi Teknis

Dokumentasi menyediakan artefak konkret yang dapat digunakan dalam *review* desain, *sprint planning*, maupun *architecture board.*

3) Mempercepat *Alignment* Antar Tim
   Dengan pemahaman yang sama terhadap konteks dan struktur sistem, konflik teknis dapat diminimalkan sejak awal.

b. Menjadi Dasar Pengambilan Keputusan Teknis
   Keputusan arsitektur bersifat jangka panjang dan berdampak luas. Dokumentasi berfungsi sebagai rekam jejak rasionalitas teknis agar keputusan tidak bersifat arbitrer.
   1) Menyimpan Rasional Pemilihan Teknologi
      Penjelasan mengapa suatu *framework* atau pola dipilih mencegah pengulangan diskusi yang sama di masa depan.
   2) Mendokumentasikan Pertimbangan *Trade-Off*
      Setiap keputusan memiliki konsekuensi terhadap performa, skalabilitas, atau kompleksitas; dokumentasi menjelaskan kompromi tersebut.
   3) Menjelaskan Batasan (*Constraints*) Sistem.
      Constraint seperti regulasi, biaya, atau batas teknologi menjadi konteks penting dalam memahami desain arsitektur.

c. Mendukung *Traceability* dan Audit
   Dalam lingkungan *enterprise*, sistem sering diaudit baik secara teknis maupun regulatif. Dokumentasi arsitektur berfungsi sebagai bukti bahwa sistem dirancang secara terkontrol.
   1) Menunjukkan kepatuhan terhadap standar
      Dokumentasi memetakan bagaimana arsitektur memenuhi standar keamanan, interoperabilitas, atau tata kelola.
   2) Menjelaskan kontrol keamanan dan dependensi
      Relasi antar komponen dan kontrol akses dapat ditelusuri melalui dokumentasi arsitektur.
   3) Mendukung audit teknis dan risiko
      Auditor dapat memahami struktur sistem tanpa harus membaca seluruh *source code.*
   4) Memfasilitasi *Onboarding* dan Transfer *Knowledge.*

   Organisasi modern menghadapi rotasi tim yang tinggi. Dokumentasi arsitektur berfungsi sebagai repositori pengetahuan sistem.

Daftar Pustaka

Cavalcante, E., & Batista, T. (2023). *Using Software Architecture Descriptions to Detect Architectural Smells at Design Time*. 122–129. https://doi.org/10.5753/cibse.2023.24697.

Kumar, A., Anand, E., Natarajan, S., & Dandekar, A. (2021). Architecture Analysis Methods. *INCOSE International Symposium*, *31*. https://doi.org/10.1002/j.2334-5837.2021.00907.x.

Kumar, A., Lokku, D. S., & Natarajan, S. (2021). Architecture Literacy. *INCOSE International Symposium*, *31*. https://doi.org/10.1002/j.2334-5837.2021.00903.x.

Martin, J. (2021). Overview of The Revised Standard on Architecture Description – ISO/IEC 42010. *INCOSE International Symposium*, *31*. https://doi.org/10.1002/j.2334-5837.2021.00906.x.

Martínez-López, J. A., García, F., Ruiz, F., & Vizcaíno, A. (2023). Contributions of Enterprise Architecture To Software Engineering: A Systematic Literature Review. *Journal of Software: Evolution and Process*, *36*. https://doi.org/10.1002/smr.2572.

Palma, E. S., Nakagawa, E., Paiva, D., & Cagnin, M. (2022a). Evolving Reference Architecture Description: Guidelines Based on ISO/IEC/IEEE 42010. *ArXiv*, *abs/2209.14714*. https://doi.org/10.48550/arxiv.2209.14714.

Pfeifer, S., Akgül, D., Röbenack, S., Tihlarik, A., Albert, B., Anacker, H., & Dumitrescu, R. (2022). Design Decisions in the Architecture Development of Advanced Systems: Towards Traceable And Sustainable Documentation And Communication. *How Product and Manufacturing Design Enable Sustainable Companies and Societies*. https://doi.org/10.35199/norddesign2022.18.

Valle, P., Garcés, L., Volpato, T., Martínez-Fernández, S., & Nakagawa, E. (2021). Towards Suitable Description of Reference Architectures. *PeerJ Computer Science*, *7*. https://doi.org/10.7717/peerj-cs.392.

PROFIL PENULIS

Bayu Waseso, M.Kom.

Penulis merupakan lulusan sarjana Teknik Informatika dari Universitas Bina Nusantara, Jakarta. Magister Komputer dari STMIK Eresha, Jakarta, jurusan Teknik Informatika dan sedang melanjutkan pendidikan doktoral pada Asia e *University* Kuala Lumpur Malaysia. Selain sebagai dosen di Universitas Mercu Buana Jakarta pada Fakultas Ilmu Komputer, jurusan Sistem Informasi, penulis juga aktif terlibat bekerja sebagai Konsultan pada berbagai proyek TI sejak tahun 2010.

Pengalaman terlibat dalam berbagai proyek TI tersebut memberikan banyak manfaat untuk bisa berbagi pengetahuan dan pengalaman saat di kelas maupun sebagai bahan penelitian ataupun saat melakukan pengabdian masyarakat, sebagai bagian dari Tri Dharma Pendidikan. Bidang spesialisasi yang penulis tekuni antara lain *Project Management,* IT *Governance,* dan *Software Engineering*. Penulis juga aktif berorganisasi pada *Project Management Institute* (PMI) Indonesia *Chapter*.

Selain itu penulis seringkali memberikan pelatihan profesional untuk pelatihan sertifikasi ITIL 4 *Foundation*, COBIT 2019 *Foundation*, serta PRINCE2 *Foundation*. Penulis menyadari teknologi selalu berkembang setiap saat oleh sebab itu penulis senantiasa untuk selalu belajar dan membagi pengetahuan serta pengalamannya baik saat di kelas ataupun melalui media buku ini, semoga pengetahuan yang sedikit bisa memberikan manfaat bagi pembaca. Semua ini hanya karena Allah SWT semata yang telah memberikan karunia-Nya.

Email Penulis: bayu.waseso@mercubuana.ac.id.

BAB 9
EVOLUSI ARSITEKTUR DAN MANAJEMEN HUTANG TEKNIS

Gergorius Kopong Pati
Universitas Stella Maris Sumba

Pendahuluan

Arsitektur perangkat lunak merupakan salah satu aspek yang paling penting dalam perancangan sistem perangkat lunak yang sukses dan berkelanjutan. Dalam pengembangan perangkat lunak, arsitektur sering kali dianggap sebagai fondasi utama yang menentukan bagaimana komponen-komponen perangkat lunak berinteraksi satu sama lain, bagaimana aplikasi akan diskalakan, dan bagaimana aplikasi akan dipelihara serta dikembangkan dalam jangka panjang.

Oleh karena itu, pemahaman yang mendalam mengenai evolusi arsitektur perangkat lunak sangat penting untuk menciptakan solusi perangkat lunak yang tidak hanya memenuhi kebutuhan pengguna, tetapi juga dapat berkembang dengan efisien di masa depan. Arsitektur perangkat lunak, sebagaimana diuraikan oleh Bass, Clements, & Kazman (2012), adalah struktur yang mendasari desain perangkat lunak dan menyatukan elemen-elemen sistem dalam cara yang terorganisir.

Ini mencakup berbagai keputusan desain yang menyangkut komponen perangkat lunak, komunikasi antar komponen, dan bagaimana aplikasi dapat ditingkatkan atau dimodifikasi sesuai kebutuhan. Evolusi arsitektur perangkat lunak mencerminkan bagaimana kebutuhan pengguna, serta kemajuan dalam teknologi, mengarahkan perubahan dalam cara perangkat lunak dirancang dan dibangun. Seiring waktu, perubahan besar dalam paradigma perangkat lunak telah terjadi, dari sistem monolitik yang awalnya sangat terpusat menuju arsitektur berbasis layanan, serta dari pendekatan tradisional menuju arsitektur yang lebih modern dan terdistribusi.

Evolusi Arsitektur Perangkat Lunak

Evolusi arsitektur perangkat lunak telah mengalami berbagai tahap, masing-masing dengan karakteristik yang menentukan cara perangkat lunak dibangun dan dikelola.

Di awal perkembangan perangkat lunak, kebanyakan aplikasi dirancang dengan pendekatan monolitik di mana semua komponen aplikasi, seperti antarmuka pengguna, logika bisnis, dan basis data, berada dalam satu sistem terintegrasi. Meskipun pendekatan ini sederhana dan mudah diimplementasikan pada awalnya, sistem monolitik cenderung sulit dipelihara dan diskalakan seiring

bertambahnya kompleksitas aplikasi. Selain itu, perubahan dalam satu komponen bisa mempengaruhi keseluruhan sistem, yang pada akhirnya meningkatkan biaya pemeliharaan.

Seiring dengan kemajuan teknologi dan kebutuhan untuk menciptakan aplikasi yang lebih fleksibel dan skalabel, arsitektur perangkat lunak pun berkembang. Salah satu perkembangan besar adalah munculnya konsep arsitektur berlapis (*layered architecture*), yang memisahkan aplikasi menjadi lapisan-lapisan yang lebih modular. Pendekatan ini memungkinkan pengembang untuk lebih mudah memelihara dan mengembangkan sistem karena perubahan yang dilakukan pada satu lapisan tidak langsung mempengaruhi lapisan lainnya.

Namun, meskipun pendekatan berlapis menawarkan beberapa keunggulan, ia tetap memiliki keterbatasan dalam menghadapi tuntutan aplikasi yang semakin kompleks. Pada tahun 2000-an, dengan berkembangnya teknologi *web* dan internet, arsitektur berbasis layanan (*Service-Oriented Architecture*/SOA) mulai diperkenalkan. SOA memungkinkan aplikasi untuk dibangun dengan menghubungkan layanan-layanan terpisah yang dapat berkomunikasi satu sama lain melalui protokol standar. Pendekatan ini membuka jalan bagi arsitektur *microservices*, yang menjadi semakin populer dalam beberapa tahun terakhir.

Microservices membagi aplikasi menjadi layanan-layanan yang lebih kecil, masing-masing bertanggung jawab atas fungsi tertentu dalam aplikasi. Keuntungan dari arsitektur ini adalah kemampuannya untuk mempermudah skalabilitas, pengelolaan, dan pengembangan aplikasi dengan lebih efisien, serta mengurangi ketergantungan antar komponen.

Dengan kemajuan *cloud computing* dan *containerization*, seperti Docker dan Kubernetes, arsitektur perangkat lunak semakin berkembang menjadi lebih terdistribusi dan fleksibel. Aplikasi dapat dibangun, dijalankan, dan diperbarui secara lebih mudah di lingkungan *cloud* tanpa terikat pada infrastruktur fisik yang spesifik. Konsep *serverless* juga muncul sebagai bagian dari perkembangan ini, memungkinkan pengembang untuk menjalankan kode tanpa perlu mengelola *server* secara eksplisit.

Semua perubahan ini menunjukkan bagaimana evolusi arsitektur perangkat lunak tidak hanya dipengaruhi oleh perkembangan teknologi, tetapi juga oleh kebutuhan yang semakin beragam dari pengguna dan bisnis.

Peran Arsitektur Dalam Manajemen Hutang Teknis

Namun, meskipun kemajuan dalam arsitektur perangkat lunak memberikan banyak keuntungan, ada tantangan yang tidak kalah penting, yaitu manajemen hutang teknis (*technical debt*). Hutang teknis adalah istilah yang pertama kali diperkenalkan oleh Ward Cunningham pada tahun 1992 untuk menggambarkan pengorbanan kualitas dalam pengembangan perangkat lunak untuk mencapai tujuan jangka pendek. Dalam konteks ini, pengembang perangkat lunak terkadang harus membuat keputusan desain yang tidak ideal atau terburu-buru dalam upaya memenuhi tenggat waktu atau persyaratan proyek yang mendesak. Keputusan ini mungkin tampak menguntungkan dalam jangka pendek, tetapi dapat menyebabkan masalah serius di masa depan, seperti kesulitan dalam pemeliharaan, pengembangan fitur baru, dan penurunan kualitas aplikasi secara keseluruhan. Manajemen hutang teknis ini menjadi sangat penting dalam konteks evolusi arsitektur perangkat lunak.

Misalnya, saat beralih dari arsitektur monolitik ke *microservices*, tim pengembang seringkali menghadapi masalah hutang teknis yang muncul sebagai akibat dari perubahan yang cepat dan besar dalam struktur aplikasi. Perubahan besar ini seringkali melibatkan komponen yang sebelumnya tidak terkelola dengan baik, dan keputusan desain yang diambil saat transisi dapat menyebabkan pengumpulan hutang teknis yang lebih besar lagi.

Oleh karena itu, memahami evolusi arsitektur dan dampaknya terhadap manajemen hutang teknis adalah kunci untuk mengelola kualitas perangkat lunak dalam jangka panjang.

Evolusi Arsitektur Perangkat Lunak

1. Pengertian Arsitektur Perangkat Lunak

Arsitektur perangkat lunak merujuk pada struktur dasar sistem perangkat lunak yang terdiri dari komponen-komponen perangkat lunak dan cara komponen-komponen tersebut saling berinteraksi. Shaw dan Garlan (1996) mendefinisikan arsitektur perangkat

Daftar Pustaka

Bache, M. (2014). Technical Debt And Software Maintenance. *Journal of Software Engineering*, 25(2), 115-128. https://doi.org/10.1016/j.jse.2014.01.002.

Brooks, F. P. (1975). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley.

Cunningham, W. (1992). The WyCash Portfolio Management System. *ACM SIGPLAN Notices*, 27(3), 29-34. https://doi.org/10.1145/143009.143017.

Fowler, M. (2009). *Refactoring: Improving The Design of Existing Code (2nd ed.).* Addison-Wesley.

Hohpe, G., & Woolf, B. (2009). *The Art of Service-Oriented Architecture*. Addison-Wesley.

Kruchten, P. (2004). *The Rational Unified Process: An Introduction*. Addison-Wesley.

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices For Teams, Programs, And The Enterprise*. Addison-Wesley.

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. *NIST Special Publication*. https://doi.org/10.6028/NIST.SP.800-145.

Rerung, R. R., Fauzan, M., & Hermawan, H. (2020). Website Quality Measurement of Higher Education Services Institution Region IV Using Webqual 4.0 Method. *International Journal of Advances in Data Fnd Information Systems*, *1*(2), 89-102.

Sargent, J. (2014). *Managing Technical Debt: A Pragmatic Guide*. Springer.

Sharp, M. (2012). *Software Architecture for Developers*. Leanpub.

Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on An Emerging Discipline*. Prentice Hall.

Stewart, D. & Simmons, M. (2010). *The Business Playground: Where Creativity and Commerce Collide*. Berkeley, AS: New Riders Press.

Tom, D. (2015). The Cost of Technical Debt: An In-Depth Study. *IEEE Software*, 32(3), 36-42. https://doi.org/10.1109/MS.2015.65

PROFIL PENULIS

Gergorius Kopong Pati

Ketertarikan penulis terhadap Evolusi Arsitektur dan Manajemen Hutang Teknis dimulai pada tahun 2023 silam. Hal tersebut membuat penulis memilih untuk masuk ke Atma Jaya dengan memilih Jurusan Kesehatan Masyarakat) dan berhasil lulus pada tahun 2014. Gergorius Kopong Pati telah menunjukkan dedikasi yang tinggi dalam mengejar ilmu pengetahuan serta berkontribusi dalam berbagai kegiatan kampus.

Selama masa perkuliahannya, ia terlibat dalam berbagai kegiatan akademik dan non-akademik yang mendukung pengembangan dirinya, baik sebagai individu maupun profesional. Selain aktif di dunia pendidikan, Penulis memiliki minat yang besar terhadap Teknologi yang ia jadikan sebagai fokus penelitian dan pengembangan diri. Melalui karya-karya yang ditulis, Gergorius bertujuan untuk memberikan kontribusi yang berarti dalam bidang yang ia tekuni.

Email Penulis: gregkopong80@gmail.com.

BAB 10
TREN MASA DEPAN DAN ETIKA DALAM ARSITEKTUR PERANGKAT LUNAK

Lilis Supratman, M.Si.
Universitas Pakuan

Pendahuluan

Perkembangan teknologi informasi yang sangat pesat telah mendorong perubahan signifikan dalam cara sistem perangkat lunak dirancang, dikembangkan, dan diimplementasikan. Arsitektur perangkat lunak memegang peranan krusial sebagai fondasi struktural yang menentukan kualitas, skalabilitas, keamanan, serta keberlanjutan sistem.

Seiring dengan meningkatnya kompleksitas kebutuhan pengguna, tuntutan terhadap sistem yang adaptif, cerdas, dan terintegrasi menjadi semakin tinggi. Oleh karena itu, tren masa depan dalam arsitektur perangkat lunak tidak hanya berfokus pada aspek teknis semata, tetapi juga mencakup dimensi etika, keamanan, dan tanggung jawab sosial.

Transformasi *digital* yang melibatkan kecerdasan buatan (*Artificial Intelligence*/AI), komputasi awan (*cloud computing*), *Internet of Things* (IoT), dan *big data* telah mengubah paradigma pengembangan sistem. Arsitektur perangkat lunak modern dituntut untuk lebih fleksibel, modular, dan resilien agar mampu beradaptasi dengan dinamika lingkungan yang cepat berubah (Bass, Clements, & Kazman, 2021). Di sisi lain, munculnya berbagai risiko baru, seperti pelanggaran privasi, bias algoritma, serta penyalahgunaan data, menuntut adanya perhatian serius terhadap aspek etika dalam perancangan sistem.

Etika dalam arsitektur perangkat lunak tidak hanya berkaitan dengan kepatuhan terhadap regulasi, tetapi juga menyangkut tanggung jawab moral pengembang dalam menciptakan sistem yang adil, transparan, dan dapat dipercaya. Hal ini menjadi semakin relevan ketika sistem perangkat lunak digunakan dalam sektor-sektor krusial seperti kesehatan, pendidikan, keuangan, dan pemerintahan (Floridi *et al.,* 2018).

Oleh karena itu, integrasi antara tren teknologi masa depan dan prinsip etika menjadi kebutuhan yang tidak terpisahkan dalam pengembangan arsitektur perangkat lunak modern. Bab ini membahas secara komprehensif mengenai tren masa depan dalam arsitektur perangkat lunak serta implikasi etika yang menyertainya.

Pembahasan meliputi perkembangan teknologi utama, perubahan paradigma desain arsitektur, tantangan etika, serta rekomendasi strategis dalam merancang sistem perangkat lunak yang berkelanjutan dan bertanggung jawab.

Selayang Pandang Arsitektur Perangkat Lunak

Arsitektur perangkat lunak dapat didefinisikan sebagai struktur fundamental suatu sistem yang terdiri atas komponen, hubungan antar komponen, serta prinsip dan pedoman yang mengatur desain dan evolusinya (Bass *et al.,* 2021). Arsitektur berfungsi sebagai kerangka kerja yang memungkinkan pengembang memahami, mengelola, dan mengembangkan sistem secara sistematis.

Menurut Richards dan Ford (2020), arsitektur perangkat lunak tidak hanya mencerminkan keputusan teknis, tetapi juga keputusan strategis yang berdampak jangka panjang terhadap kinerja, keamanan, dan keberlanjutan sistem. Oleh sebab itu, perancangan arsitektur harus mempertimbangkan berbagai atribut kualitas, seperti *scalability, reliability, maintainability, security,* dan *usability*.

Seiring berkembangnya teknologi, pendekatan arsitektur juga mengalami transformasi. Arsitektur monolitik yang sebelumnya dominan kini mulai bergeser menuju pendekatan yang lebih modular, seperti *microservices* dan *event-driven architecture*. Pergeseran ini memungkinkan sistem menjadi lebih fleksibel, mudah dikembangkan, dan adaptif terhadap perubahan kebutuhan bisnis (Newman, 2021). Perbedaan karakteristik antara arsitektur tradisional dan modern dapat dilihat pada tabel 10.1.

Tabel 10.1: Perbandingan Arsitektur Tradisional dan Arsitektur Modern

Aspek	**Tradisional (Monolitik)**	**Modern (*Microservices* & *Cloud-Native*)**
Struktur	Terpusat	Terdistribusi
Skalabilitas	Rendah	Tinggi
Pengembangan	Lambat	Cepat dan Iteratif
Deployment	Terpusat	*Continuous deployment*

Tabel 10.4: Implikasi Tren dan Etika Bagi Pendidikan dan Industri

Sektor	Implikasi
Pendidikan	Integrasi etika TI
Dosen	Literasi teknologi
Mahasiswa	Tanggung jawab profesi
Industri	*DevSecOps* & *ethical* AI
Pemerintah	Regulasi & *governance*

Sumber: Diolah Penulis.

Tren masa depan dalam arsitektur perangkat lunak menunjukkan arah yang semakin kompleks, adaptif, dan terintegrasi dengan teknologi cerdas. Perkembangan *microservices*, *cloud-native architecture*, AI, IoT, dan *DevSecOps* membawa peluang besar sekaligus tantangan signifikan. Di sisi lain, meningkatnya risiko pelanggaran privasi, bias algoritma, dan ancaman keamanan menuntut perhatian serius terhadap aspek etika.

Oleh karena itu, integrasi prinsip etika dalam perancangan arsitektur perangkat lunak menjadi kebutuhan mendesak. Pendekatan *ethics by design* memungkinkan terciptanya sistem yang tidak hanya unggul secara teknis, tetapi juga bertanggung jawab secara sosial. Dengan demikian, arsitektur perangkat lunak masa depan diharapkan mampu mendukung transformasi digital yang berkelanjutan, adil, dan berorientasi pada kesejahteraan manusia.

Daftar Pustaka

Anderson, R. (2020). *Security Engineering: A Guide To Building Dependable Distributed Systems (3rd ed.).* Wiley.

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... Suter, P. (2017). Serverless Computing: Current Trends And Open Problems. *Research Advances in Cloud Computing*, 1–20.

Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture In Practice (4th ed.).* Addison-Wesley.

Burns, B. (2018). *Designing Distributed Systems*. O'Reilly Media.

Denning, P. J., & Tedre, M. (2019). *Computational Thinking*. MIT Press.

Floridi, L., Cowls, J., Beltrametti, M., Chatila, R., Chazerand, P., Dignum, V., ... Vayena, E. (2018). AI4People An Ethical Framework For A Good AI society. *Minds and Machines*, 28(4), 689–707.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A Vision, Architectural Elements, And Future Directions. *Future Generation Computer Systems*, 29(7), 1645–1660.

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

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art Of Software Testing (3rd Ed.).* Wiley.

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

O'Neil, C. (2016). *Weapons of Math Destruction*. Crown.

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

Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach (4th ed.).* Pearson.

Solove, D. J. (2021). *Understanding Privacy*. Harvard University Press.

Van den Hoven, J., Vermaas, P. E., & Van de Poel, I. (2015). *Handbook of Ethics, Values, And Technological Design*. Springer.

PROFIL PENULIS

Lilis Supratman, M.Si.

Lilis Supratman adalah seorang dosen dari lima bersaudara yang bertempat tinggal di Kota Bogor. Selama mengabdi di Universitas Pakuan Bogor sejak tahun 2007, menulis merupakan hobinya. Tidak kurang dari 11 buku bersertifikat ISBN yang sudah dirilis dan 13 *e-book* yang bersertifikat HaKI. Disiplin ilmu yang digeluti adalah mikrobiologi dengan cakupan materi jamur dan liken.

Aplikasi keilmuan diimplementasikan dalam bentuk kegiatan pengabdian masyarakat dalam berbagai topik yang bersifat multidisiplin (topik *stunting*, media *digital*, cerpen berbasis AI, pembuatan pupuk organik dan pembuatan lembar kerja/*worksheet*). Membaca adalah cara kita menyerap hikmah dari dunia, dan menulis adalah cara kita mengembalikan kebaikan itu kepada dunia. Saat kita membaca, kita memperluas pandangan dan menguatkan hati; saat kita menulis, kita meninggalkan jejak pemikiran agar orang lain ikut merasakan cahaya yang pernah menyinari kita. Tak perlu menjadi hebat dulu untuk mulai menulis, justru dengan menulis kita tumbuh, belajar, dan akhirnya memberi arti. Membaca dan menulis adalah pasangan serasi dalam memahami alam sekitar. Oleh karena itu, pahamilah bahwa "cuplikan media sosial bisa menggiring persepsi. Tapi melalui buku, memberikan pemahaman yang utuh".

Email Penulis: lilis@unpak.ac.id.

ARSITEKTUR PERANGKAT LUNAK

Panduan Komprehensif Perancangan Sistem

Dunia pengembangan perangkat lunak telah berevolusi jauh melampaui sekadar penulisan baris kode. Di era di mana sistem harus menangani jutaan transaksi secara simultan, menjaga keamanan data yang sensitif, dan tetap fleksibel terhadap perubahan bisnis yang sangat cepat, peran Arsitektur Perangkat Lunak menjadi fondasi yang menentukan antara keberhasilan dan kegagalan sebuah produk teknologi. Arsitektur perangkat lunak bukan hanya sekadar rancangan teknis, melainkan juga strategi untuk memastikan kualitas, skalabilitas, dan keberlangsungan sistem. Oleh karena itu, buku ini disusun secara sistematis agar dapat menjadi panduan komprehensif bagi mahasiswa, peneliti, praktisi, maupun pengembang perangkat lunak. Isi buku ini mencakup berbagai aspek mulai dari:

1. Introduksi Arsitektur Perangkat Lunak
2. Karakteristik dan Atribut Kualitas Sistem
3. Analisis Trade-off dan Pengambilan Keputusan Arsitektur
4. Arsitektur Berlapis (Layered Architecture)
5. Arsitektur Microservices
6. Keamanan dalam Arsitektur (Security by Design)
7. Arsitektur Cloud-Native dan Serverless
8. Dokumentasi dan Visualisasi Arsitektur Perangkat Lunak
9. Evolusi Arsitektur dan Manajemen Hutang Teknis
10. Tren Masa Depan dan Etika dalam Arsitektur Perangkat Lunak

sada kurnia pustaka

ISBN 978-634-7522-33-7

9 786347 522337