Migrasi dari Arsitektur Monolitik ke Arsitektur Mikroservis Arsitektur: Tinjauan Literatur Sistematis

Hossam Hassan, Manal A. Abdel-Fattah, Wael Mohamed
Departemen Sistem Informasi, Universitas Helwan, Mesir

Abstrak—Migrasi dari sistem perangkat lunak monolitik ke arsitektur layanan mikro modern merupakan proses penting untuk meningkatkan skalabilitas, pemeliharaan, dan kinerja sistem perangkat lunak. Studi ini melakukan tinjauan literatur sistematis untuk mengeksplorasi berbagai metodologi, teknik, dan algoritma yang digunakan dalam migrasi sistem monolitik ke arsitektur layanan mikro modern. Lebih lanjut, studi ini menggarisbawahi peran kecerdasan buatan dalam meningkatkan efisiensi dan efektivitas proses migrasi dengan meneliti literatur terkini untuk mengidentifikasi pola, tantangan, dan solusi optimal yang signifikan. Selain itu, studi ini menekankan pentingnya migrasi sistem monolitik ke layanan mikro dengan mensintesis berbagai studi penelitian yang memungkinkan fleksibilitas, toleransi kesalahan, dan skalabilitas independen yang lebih besar. Temuan ini menawarkan wawasan berharga bagi para peneliti dan praktisi di industri perangkat lunak. Selain itu, artikel ini memberikan panduan praktis tentang implementasi metodologi berbasis AI dalam evolusi arsitektur perangkat lunak. Terakhir, kami menyoroti arah penelitian masa depan dalam menyediakan teknik otomatisasi untuk migrasi arsitektur perangkat lunak.

Kata kunci—Migrasi perangkat lunak; evolusi perangkat lunak; arsitektur monolitik; arsitektur layanan mikro; tinjauan literatur sistematis

I. PENDAHULUAN

Selama dekade terakhir, arsitektur perangkat lunak telah berubah secara signifikan. Perangkat lunak awalnya bersifat monolitik, mengintegrasikan semua komponen ke dalam satu unit. Namun, seiring dengan semakin kompleksnya sistem perangkat lunak, muncul keterbatasan dalam skalabilitas, fleksibilitas, dan pemeliharaan. Hal ini mendorong penyelidikan pola arsitektur perantara seperti arsitektur mikro-kernel dan arsitektur berorientasi layanan (SOA). Arsitektur mikro-kernel memiliki sistem inti minimal dengan hanya fungsi-fungsi terpenting sambil mendelegasikan fungsionalitas tambahan ke komponen modular di ruang pengguna, sedangkan

SOA berfokus pada penguraian perangkat lunak menjadi layanan-layanan yang terhubung secara longgar. Evolusi ini membuka jalan bagi arsitektur microservices baru, yang membagi perangkat lunak menjadi komponen-komponen kecil, otonom, dan hanya memenuhi satu kebutuhan.

Sebagian besar bisnis perangkat lunak di seluruh dunia telah mengadopsi arsitektur microservice modern karena skalabilitas, pemeliharaan, fleksibilitas, dan kemudahan pengembangannya yang meningkat secara signifikan [1], [2]. Selain itu, banyak bisnis perangkat lunak, seperti Amazon, Uber, Netflix, dan Spotify, mengadopsi pendekatan arsitektur ini, dan transisi ke microservice sedang berlangsung [3]. Arsitektur microservice dapat diwakili oleh kumpulan layanan kecil, masing-masing beroperasi dalam prosesnya sendiri dan berinteraksi menggunakan protokol ringan seperti HTTP (Hypertext Transfer Protocol), yang dikembangkan berdasarkan kebutuhan bisnis dan disampaikan secara independen [2], [4].

Arsitektur microservices memecahkan banyak masalah arsitektur monolitik. masalah sistem. Beberapa manfaat utama microservices meliputi skalabilitas, yang meningkatkan pemanfaatan sumber daya dan kinerja puncak, dan fleksibilitas, yang memungkinkan tim pengembang untuk memilih alat dan teknologi terbaik untuk setiap layanan, menghasilkan solusi yang lebih disesuaikan dan efisien, serta toleransi kesalahan, yang mencegah kegagalan sistem mempengaruhi semua layanan, meningkatkan ketahanan dan keandalan [3].

Selama migrasi ke arsitektur microservice, beberapa gudang perangkat lunak yang memiliki sistem berbasis monolit mencoba untuk menguraikannya menjadi implementasi berbasis microservice yang koheren. Tujuan dari penguraian ini adalah untuk membantu arsitek perangkat lunak dalam mengidentifikasi kandidat microservice dengan menganalisis domain aplikasi, kebutuhan bisnis, kode sumber, dan informasi terkait versi [5], [6], [7].

Mengingat demikian, vendor perangkat lunak telah melakukan upaya untuk melakukan migrasi manual dari ekosistem aplikasi monolitik ke ekosistem aplikasi berorientasi microservice. Skalabilitas, independensi komponen, manajemen data, komunikasi layanan, penyebaran, dan pemantauan adalah beberapa upaya tersebut [1], [8]. Proses migrasi ini bersifat subjektif, membutuhkan penilaian manusia, dan rentan terhadap kesalahan. Pendapat ahli diperlukan untuk proses ini, begitu pula keahlian insinyur perangkat lunak dalam ekstraksi microservice dan pelestarian kualitas sistem [9]. Selain itu, ekstraksi microservice menjadi proses yang kompleks karena tidak ada metode yang jelas atau mudah untuk mendefinisikan batasan microservice. Tahap pertama dan paling menantang dalam memecah aplikasi monolitik melibatkan identifikasi batasan microservice. Identifikasi yang tidak memadai dapat menyebabkan sistem yang lebih kompleks dengan kualitas yang lebih rendah [1].

Untuk mengatasi kesenjangan ini, terdapat peningkatan minat dalam menggunakan algoritma AI dan ML untuk memfasilitasi proses migrasi. Metode berbasis AI, seperti teknik berbasis pencarian dan algoritma pengelompokan, memungkinkan untuk secara otomatis menemukan komponen layanan mikro dan meningkatkan dekomposisi sistem.
Namun, mayoritas penelitian gagal mengidentifikasi microservice yang relevan dan potensial, dan mereka kesulitan menentukan jumlah kandidat microservice yang tepat sekaligus memastikan granularitas dan keterkaitan yang longgar [10].

Makalah ini bertujuan untuk melakukan kajian yang komprehensif dan sistematis. Tinjauan pustaka ini bertujuan untuk menganalisis dan mengidentifikasi metode dan teknik terpenting yang digunakan untuk memfasilitasi proses migrasi perangkat lunak dari sistem monolitik ke layanan mikro modern. Tinjauan ini juga bertujuan untuk membandingkan hasilnya secara rinci melalui tinjauan pustaka sistematis (SLR).

yang dapat berfungsi sebagai dasar untuk mengembangkan solusi yang efektif.

Bagian-bagian selanjutnya dari makalah ini mengikuti struktur tersebut: Bagian II mencakup latar belakang dan motivasi, sedangkan Bagian III menyajikan metodologi penelitian. Bagian IV memberikan gambaran umum tentang keadaan pengetahuan dan pemahaman saat ini mengenai proses migrasi perangkat lunak.
Bagian V menyajikan temuan dari tinjauan pustaka ini.
Bagian VI berisi kesimpulan dari tinjauan pustaka ini dan menawarkan saran untuk pekerjaan dan perbaikan di masa mendatang.

II. LATAR BELAKANG

Arsitektur monolitik dan arsitektur layanan mikro adalah dua metodologi berbeda untuk merancang dan membangun sistem perangkat lunak. Berbagai faktor memengaruhi pilihan di antara keduanya. Setiap arsitektur memiliki kelebihan dan kekurangannya masing-masing, dan penentuan harus dilakukan dengan mempertimbangkan persyaratan dan batasan khusus proyek. Pada subbagian berikut, kami akan memberikan informasi tambahan mengenai persamaan, perbedaan, kelebihan, dan kekurangan dari kedua gaya arsitektur tersebut.

A. Arsitektur Monolitik

Unit yang diimplementasikan secara tunggal disebut sebagai monolit. Sistem monolitik memerlukan implementasi simultan dari semua fungsi. Implementasi semua kode sebagai proses terpadu, mengkonsolidasikan semua kode ke dalam satu proses, merupakan ciri arsitektur monolitik [2].

Monolit modular, versi baru dari monolit proses tunggal, membagi proses tunggal menjadi beberapa modul berbeda yang dikembangkan secara terpisah; namun, penyebaran memerlukan kombinasi modul-modul ini. Ini mungkin merupakan solusi optimal untuk beberapa perusahaan dan gudang perangkat lunak karena mendefinisikan batasan modul dengan baik, menyediakan sejumlah besar pekerjaan paralel, mengatasi kompleksitas yang terkait dengan arsitektur layanan mikro terdistribusi, dan mengadopsi topologi penyebaran yang lebih sederhana [2], [11]. Perbedaan antara monolit monolitik dan monolit modular

Arsitektur perangkat lunak dapat ditunjukkan pada Gambar 1. Shopify adalah contoh perusahaan yang menggunakan teknik ini sebagai pengganti dekonstruksi microservice, dan tampaknya cukup efektif bagi mereka.

Sayangnya, monolit belakangan ini menjadi simbol yang dihindari dan sering dikaitkan dengan sistem lama.
Namun, hal ini sebenarnya merupakan pilihan yang layak berdasarkan persyaratan dan spesifikasi sistem. Beberapa keuntungan dari arsitektur monolitik tercantum di bawah ini [2], [12]:

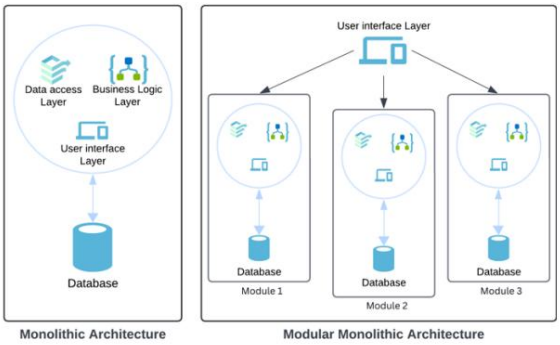
1) Topologi penyebaran yang lebih cepat dan mudah: Menghindari berbagai masalah sistem terdistribusi.

2) Alur kerja untuk pengembang lebih mudah dikelola.

3) Pemantauan dan pemecahan masalah yang lebih sederhana: Pengujian ujung-ke-ujung disederhanakan.

4) Penggunaan kembali kode cukup sederhana, tanpa duplikasi apa pun.

Namun demikian, masalah penting pada arsitektur monolit modular adalah bahwa basis data tidak memiliki tingkat dekomposisi yang sama dengan kode, sehingga menyulitkan pemisahan monolit di masa mendatang [2].



Gambar 1. Menunjukkan perbedaan antara arsitektur perangkat lunak monolitik dan modular.

B. Arsitektur Layanan Mikro

Microservices dirancang agar memiliki keterkaitan yang longgar dan dapat diimplementasikan secara independen, tetapi mungkin masih bergantung pada pola koordinasi yang berbeda untuk menangani transaksi terdistribusi dan komunikasi antar layanan, yang dirancang berdasarkan domain bisnis tertentu dan berpotensi dirilis secara terpisah. Sebuah layanan adalah modul yang memiliki fungsi spesifik dan memungkinkan layanan lain untuk mengaksesnya melalui jaringan. Dengan menggabungkan modul-modul ini, gudang perangkat lunak dapat menciptakan sistem perusahaan yang lebih rumit. Setiap microservice dapat mewakili aspek spesifik dari sistem. Ketika digabungkan, microservices ini membentuk sistem perusahaan yang lengkap. Dengan kata lain, mereka adalah jenis arsitektur berorientasi layanan yang memiliki sudut pandang tertentu tentang bagaimana batasan layanan harus didefinisikan dengan kemampuan untuk diimplementasikan secara independen. Gambar 2 menunjukkan

Contoh konstruksi arsitektur microservice.

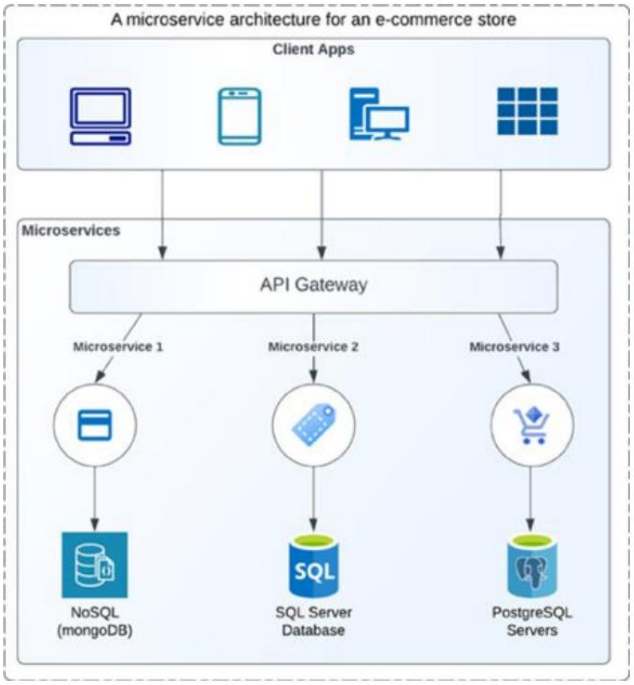
Sebuah microservice tunggal tampak seperti kotak hitam. Satu atau lebih titik akhir jaringan, seperti API SOAP atau REST, menampung fungsionalitas bisnis menggunakan protokol yang sesuai. Melalui titik akhir jaringan ini, konsumen—microservice atau program—mengakses kemampuan ini. Dunia luar menyembunyikan elemen implementasi seperti teknologi layanan dan penyimpanan data. Dalam kebanyakan kasus, desain microservice membungkus basis data mereka sendiri alih-alih menggunakan basis data umum.

Arsitektur microservices menyembunyikan informasi di dalam komponen-komponennya dan memberikan informasi minimal melalui antarmuka eksternal. Implementasi microservice tersembunyi dapat diperbarui secara bebas selama perubahan tersebut tidak menimbulkan modifikasi yang tidak kompatibel pada antarmuka jaringan yang diekspos nya. Perubahan yang dilakukan dalam batas microservice tidak boleh memengaruhi konsumen hulu, sehingga memungkinkan rilis fungsionalitas yang terpisah. Batas layanan yang jelas dan konsisten yang tidak berubah dengan implementasi internal menghasilkan kopling yang lebih longgar dan kohesi yang lebih besar. Beberapa keuntungan arsitektur microservice tercantum di bawah ini:

1) Kemandirian dalam penerapan: Memungkinkan modifikasi dan penerapan tanpa bergantung pada layanan mikro lainnya.

2) Model berbasis domain bisnis: Mempermudah menghadirkan fungsionalitas baru dan menggabungkan kembali layanan mikro untuk memberikan kemampuan baru kepada konsumen.

3) Memiliki kendali atas statusnya sendiri: Microservices harus menghindari ketergantungan pada basis data bersama. Sebaliknya, ia perlu meminta data dari microservice lain untuk mengaksesnya.



Gambar 2. Arsitektur layanan mikro untuk sistem belanja online.

III. METODOLOGI PENELITIAN

Untuk mencapai tujuan kami dalam memigrasikan arsitektur perangkat lunak secara otomatis dari monolitik ke microservice, SLR digunakan untuk menganalisis proses migrasi. Investigasi ini mencakup penggunaan AI dan teknik lain untuk mengotomatiskan proses tersebut. SLR dipilih dalam penelitian ini karena keunggulannya dibandingkan tinjauan literatur konvensional dalam hal akurasi, efektivitas, dan organisasi. Ini membantu dalam mengidentifikasi tujuan kami, menilai hasilnya, dan mengklasifikasikannya ke dalam kategori. SLR ini terdiri dari tiga langkah utama, yaitu sebagai berikut [13], [14], dan [15]:

1) Perencanaan dan persiapan ulasan.
2) Melakukan peninjauan.
3) Melaporkan hasil ulasan.

A. Fase 1: Tinjauan Perencanaan

Dimulai dengan langkah-langkah ini, fase ini mencakup inti dari SLR:

1) Nyatakan dengan jelas pertanyaan-pertanyaan yang akan dijawab oleh penelitian ini:

RQ1: Metode AI mana yang efektif untuk migrasi arsitektur microservice otomatis?

RQ2: Apa saja hambatan dan kendala utama yang dihadapi organisasi selama proses migrasi, dan bagaimana solusi berbasis AI dapat mengatasi tantangan ini?

RQ3: Apa kriteria yang digunakan untuk menilai efektivitas dan keberhasilan strategi migrasi berbasis AI?

2) Memilih repositori penelitian yang tepat: Dalam SLR ini, berbagai perpustakaan digital daring digunakan, termasuk namun tidak terbatas pada IEEE Xplore, Google Scholar, Science Direct, Springer, dan MDPI.

3) Menetapkan kriteria penelitian: Rangkaian kata kunci dipilih berdasarkan kata kunci SLR dan alternatif kata kunci tersebut yang ditemukan dalam Evolusi Arsitektur Perangkat Lunak. Seperti yang terlihat pada Tabel I, rangkaian kata kunci ini dibagi menjadi dua kategori yang berbeda.

4) Proses Migrasi Perangkat Lunak.
5) Dari Monolitik ke Mikroservis.

TABEL I. KATA KUNCI PENCARIAN UNTUK TINJAUAN LITERATUR SISTEMATIS

KELOMPOK	String Pencarian
Perangkat lunak Migrasi	(“Migrasi Perangkat Lunak” ATAU “Arsitektur Perangkat Lunak” “Evolusi” ATAU “Transformasi Arsitektur Perangkat Lunak” ATAU “Evolusi Perangkat Lunak” ATAU “Arsitektur Perangkat Lunak”)
	DAN
monolitik menjadi Layanan mikro	(("Warisan", "Monolitik", "Sistem Monolitik", "Aplikasi Lapisan Tunggal", "Monolitik Modular")) DAN (("Mikroservis", "Pola mikroservis", "Arsitektur mikroservis", "Arsitektur berorientasi layanan (SOA)")

6) Memberikan definisi yang jelas tentang inklusi & eksklusi: Pedoman untuk kriteria inklusi dan eksklusi untuk hal ini SLR didirikan oleh Kitchenhem [13].

Kriteria inklusi berikut ini tercantum:

IC1: Publikasi jurnal atau presentasi konferensi diperlukan untuk seleksi penelitian.

IC2: Studi harus berfokus pada proses migrasi perangkat lunak, khususnya migrasi dari arsitektur monolitik ke arsitektur microservice.

IC3: AI, ML, dan algoritma atau solusi lain untuk perangkat lunak Migrasi harus diterapkan.

IC4: Studi yang dipublikasikan antara tahun 2018 dan 2024. Mikroservis telah diperkenalkan sebelumnya, tetapi studi terbaru menunjukkan implementasi, tantangan, dan inovasi terkini.

Kriteria pengecualian berikut ini tercantum:

EC1: Studi yang gagal menjawab pertanyaan penelitian.

EC2: Studi yang mengabaikan proses migrasi perangkat lunak.

EC3: Studi yang tidak menyertakan microservices.

EC4: Publikasi yang diterbitkan sebelum tahun 2018.

7) Menetapkan standar untuk mengukur kualitas: Fase ini penting karena kami membandingkan dan memeriksa kualitas studi yang dipilih dengan tujuan, pertanyaan penelitian, dan sasaran kami.

B. Fase 2: Melakukan Peninjauan:

1) Seleksi data primer: Pada tahap ini, metode penyaringan digunakan untuk menerapkan kriteria pencarian dan menentukan kriteria inklusi dan eksklusi. Proses seleksi studi primer telah dimulai. Pendekatan Tollgate diterapkan untuk meningkatkan efektivitas dan efisiensi proses seleksi secara sistematis dan terorganisir [13].

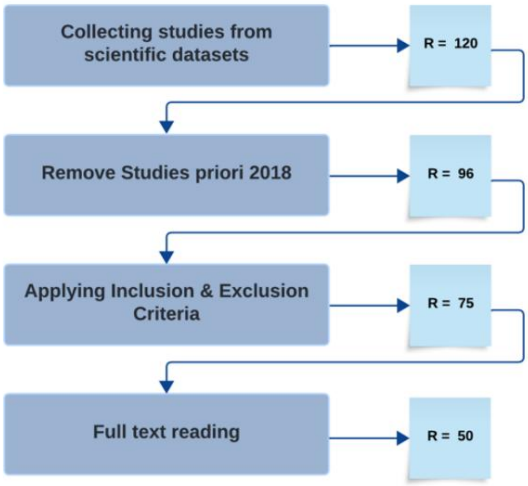
2) Ekstraksi data: Pemilihan studi dipandu oleh kriteria spesifik, termasuk metodologi penelitian, publikasi.

tahun, jenis studi, dan batasan atau pembatasan apa pun yang dikenakan pada studi tersebut.

3) Sintesis data: Studi-studi yang dikumpulkan dinilai dan dibandingkan dengan pertanyaan penelitian dan tujuan studi kami.

C. Fase 3: Pelaporan Tinjauan:

Selama fase ini, studi-studi terpilih diverifikasi dan dibandingkan dengan kriteria kualitas. Gambar 3 menunjukkan proses SLR, yang mengindikasikan kumpulan studi yang terorganisir dengan baik yang tersedia untuk diskusi dan investigasi, dengan R mewakili jumlah studi penelitian di setiap langkah.



Gambar 3. Proses seleksi SLR.

IV. KONDISI TERKINI

Bagian ini mengkaji studi-studi yang dipilih untuk dianalisis, ditinjau, dan didiskusikan.

Jin et al. [16] mengusulkan metode ekstraksi microservice berorientasi fungsional (FoME) yang menggunakan pengelompokan jejak eksekusi dan mengklasifikasikan entitas kode sumber sesuai dengan fungsinya. Para penulis mengevaluasi metode mereka terhadap tiga metode (LIMBO, WCA, dan MEM) di empat proyek open-source. Temuan mereka menunjukkan bahwa FoME menghasilkan microservice dengan kohesivitas yang sebanding dengan metode lain dan mencapai kopling yang jauh lebih longgar. Namun, metode mereka bergantung pada kasus uji berkualitas tinggi, dengan upaya di masa mendatang diarahkan pada otomatisasi penuh proses tersebut.

Sellami et al. [1] mengusulkan metode yang disebut MSExtractor, yang mengidentifikasi microservices sebagai masalah optimasi multi-objektif menggunakan algoritma evolusi berbasis indikator (IBEA) sambil mempertimbangkan ketergantungan struktural dan semantik dalam kode sumber. Mereka melakukan benchmark terhadap tujuh sistem perangkat lunak untuk menilai efektivitas metode mereka. Temuan mereka menunjukkan bahwa MSExtractor
mengungguli algoritma pengelompokan lainnya seperti FoME dan MEM. disebutkan dalam [16], [17]. Namun demikian, metode mereka terbatas pada empat aplikasi web dan kurang generalisasi. Pekerjaan selanjutnya harus mempertimbangkan metrik evaluasi non-fungsional.

Velepucha et al. [18] berkonsentrasi pada penguraian arsitektur microservice. Mereka membandingkan berbagai konsep, Menyusun daftar pola arsitektur microservice seperti yang ditunjukkan pada Tabel II, dan membahas kelebihan dan kekurangan microservice dibandingkan arsitektur monolitik. Pekerjaan di masa mendatang dan keterbatasannya meliputi adaptasi micro-frontend, migrasi otomatis proses dekomposisi, pembatasan hasil pada pendekatan berorientasi objek, dan evaluasi lebih lanjut terhadap literatur.

TABEL II. DAFTAR POLA YANG DIGUNAKAN DALAM DEKOMPOSISI MIKROLAYANAN PROSES

Pola / Deskripsi
Domain-Driven: Mengembangkan sistem perangkat lunak yang berfokus pada logika bisnis.
Penemuan layanan: Menangani interaksi dan komunikasi layanan.
Berbasis data: Merancang sistem berdasarkan perilaku atau struktur data. Data adalah kuncinya.
Backend untuk frontend: Mengembangkan layanan yang disesuaikan untuk klien frontend.
Adaptor microservice: Mentransformasikan data fungsionalitas microservice.
Aplikasi Strangler: Migrasi bertahap ke layanan mikro.
Layanan mikro data bersama: Berbagi dan mengelola data layanan mikro.
Layanan mikro agregator: Menggabungkan data dari beberapa layanan mikro.

Di sisi lain, Kazanaviÿius et al. menghasilkan model konseptual yang bertujuan untuk memigrasikan basis data monolitik ke dalam sistem persistensi poliglota multi-model [19]. Mereka menilai bukti konsep mereka menggunakan definisi atribut kualitas standar ISO/IEC 25012:2008. Temuan mereka menunjukkan bahwa metode yang mereka usulkan dapat secara efektif mentransisikan penyimpanan data dari arsitektur monolitik ke arsitektur layanan mikro.
Arah pengembangan ke depan mencakup kebutuhan untuk mengotomatisasi solusi dan fakta bahwa adopsi model mereka bergantung pada satu aplikasi tunggal.

Nordli et al. [20] berfokus pada vendor solusi monolitik; mereka kesulitan untuk mengkonversi produk monolitik menjadi solusi SaaS berbasis cloud multi-tenant karena banyak klien, terutama perusahaan besar, menginginkan produk yang disesuaikan. Para penulis menyajikan bukti konsep yang menguraikan pendekatan gabungan untuk mengubah monolit menjadi SaaS berbasis cloud yang dapat disesuaikan. Temuan mereka menunjukkan bahwa pendekatan migrasi yang didorong oleh kustomisasi dapat mengarahkan monolit untuk menjadi SaaS. Keterbatasan mereka termasuk tidak tersedianya dataset, kebutuhan akan evaluasi ahli dan penggunaan sistem dunia nyata untuk menggeneralisasi hasilnya, dan persyaratan untuk proses migrasi otomatis.

Velepucha et al. [21] melakukan SLR dan memberikan daftar tantangan dan manfaat yang muncul ketika melakukan proses migrasi, seperti yang ditunjukkan pada Tabel III. Beberapa keterbatasan dan
Pekerjaan selanjutnya mencakup perlunya menganalisis pro dan kontra. dari setiap arsitektur, serta mengotomatiskan proses migrasi.

TABEL III. TANTANGAN YANG DIHADAPI SELAMA MIGRASI PERANGKAT LUNAK

Masalah/Tantangan	
Menggunakan alat yang tepat selama proses migrasi.	Lakukan migrasi microservices secara keseluruhan tanpa memecahnya menjadi bagian-bagian yang terpisah.
Penataan ulang pemangku kepentingan diperlukan saat mengimplementasikan layanan mikro.	Tantangan dalam mengidentifikasi dan mendesain layanan mikro.
Berkeinginan untuk mentransisikan semua program monolitik ke layanan mikro.	Pastikan konsistensi terjaga saat melakukan transisi antar basis data.
Berupaya mengintegrasikan teknologi baru ke dalam aplikasi monolitik.	

Faustino et al. [11] melakukan studi kasus transformasi sistem ke microservice menggunakan arsitektur monolitik modular. Mereka menemukan bahwa arsitektur monolitik modular dapat menyederhanakan proses migrasi. Selain itu, hal ini mengatasi masalah yang berkaitan dengan optimasi kinerja, konsistensi bertahap, dan komunikasi antar-microservice.

Namun, keterbatasan metode mereka pada satu studi kasus migrasi menghambat kemampuan generalisasinya.

Blinowski et al. [12] membandingkan kinerja dan skalabilitas arsitektur monolitik dan mikroservis dengan mengimplementasikan aplikasi web referensi dengan dua teknologi dan gaya arsitektur yang berbeda. Selain itu, mereka memilih tiga skenario penyebaran yang berbeda. Temuan mereka menunjukkan bahwa arsitektur monolitik mengungguli mikroservis pada satu mesin, dengan Java menangani tugas-tugas yang membutuhkan komputasi intensif lebih efisien daripada .NET. Untuk layanan non-komputasi pada mesin dengan daya terbatas, penskalaan vertikal Azure terbukti lebih hemat biaya daripada penskalaan horizontal. Dalam studi selanjutnya, penulis bermaksud untuk meningkatkan kompleksitas sistem benchmarking mereka, memperluas aplikasinya di berbagai platform cloud, dan memasukkan lebih banyak metrik kinerja.

Bastidas Fuertes et al. [22] menggunakan Transpiler pada lapisan back-end model desain arsitektur perangkat lunak untuk secara otomatis mengubah logika bisnis dari satu kode sumber ke beberapa versi yang setara. Mereka menguji model tersebut untuk kinerja, skalabilitas, dan keandalan dalam berbagai skenario dan membandingkannya dengan model desain perangkat lunak yang ada untuk mengidentifikasi pro dan kontranya. Hasil mereka menunjukkan bahwa model yang diusulkan bertujuan untuk menghemat biaya, mengoptimalkan proses pengembangan, dan meningkatkan efektivitas platform multi-bahasa pemrograman.
Meskipun demikian, para penulis mengakui perlunya penyempurnaan efektivitas model untuk menggeneralisasi hasilnya.

Mazzara et al. [8] menyajikan studi kasus untuk mengilustrasikan keuntungan transformasi monolitik menjadi microservice pada skalabilitas. Studi kasus ini berfokus pada sistem FX Core, yang sangat penting bagi Danske Bank. Mereka membandingkan kedua arsitektur tersebut, dan hasilnya menunjukkan bahwa microservice telah menghasilkan peningkatan skalabilitas dan secara efektif menyelesaikan masalah signifikan yang ditimbulkan oleh monolitik. Namun, temuan mereka kurang wawasan, validasi, dan verifikasi.

Selain itu, studi kasus ini menggunakan metodologi agile, yang membatasi kemampuan generalisasi hasilnya.

Assunção et al. [10] mengadopsi pendekatan untuk mendesain ulang fitur menjadi layanan mikro dengan menggunakan teknik berbasis pencarian untuk menilai secara kuantitatif kemungkinan desain ulang fitur monolitik sebagai layanan mikro berdasarkan empat kriteria: kopling, kohesi, overhead jaringan, dan modularisasi fitur. Untuk mengevaluasi temuan mereka, wawancara dengan delapan pengembang sistem monolitik dilakukan untuk mendapatkan umpan balik mereka. Temuan mereka menunjukkan bahwa pendekatan mereka menunjukkan hasil positif dan mendorong eksplorasi lebih lanjut.

Untuk membenarkan kemampuan konfigurasi, arah penelitian selanjutnya mencakup generalisasi hasil dan pengusulan kriteria spesifik yang terkait dengan variabilitas.

Teguh Prasandy dkk. [23] menyajikan metode modularisasi kode sumber aplikasi, basis data, dan server cloud untuk mengidentifikasi persiapan yang diperlukan untuk melakukan transisi yang sukses ke microservices. Temuan mereka

Penelitian ini menunjukkan bahwa migrasi ke microservices dapat menghadirkan tantangan dan memengaruhi pemegang saham, khususnya analis sistem dan pengembang. Selain itu, sangat penting untuk mengisolasi blok program selama penerapan dan menentukan waktu unggah untuk mencegah kegagalan. Mereka menggunakan alat Postman untuk menilai REST API baik sebagai klien REST maupun sebagai aplikasi. Penelitian selanjutnya akan mencakup penilaian kapasitas server cloud, dan evaluasi diperlukan untuk menggeneralisasi temuan.

Dengan menghitung metrik (latensi, throughput, skalabilitas, penggunaan CPU, penggunaan memori, dan penggunaan jaringan) yang dibutuhkan untuk membandingkan aplikasi sumber dan target, Fondazione et al. [24] mengembangkan metode untuk menentukan apakah migrasi ke microservices bermanfaat. Temuan mereka menunjukkan bahwa monolitik bekerja dengan baik pada sistem kecil hingga menengah, yang biasanya didefinisikan oleh ukuran dan kompleksitas keseluruhan proyek. Rasio skalabilitas yang jauh lebih tinggi dari sistem microservice mendukung hipotesis bahwa sistem ini berkinerja lebih baik daripada desain monolitik untuk sistem dengan terlalu banyak pengguna bersamaan, terutama dalam hal menangani lebih banyak lalu lintas. Selain itu, para peneliti melakukan eksperimen benchmarking untuk mengevaluasi hasil mereka. Pekerjaan selanjutnya meliputi pengujian pada sistem nyata, menggunakan bahasa pemrograman alternatif, dan memastikan keamanan.

Abgaz et al. [25] melakukan SLR dengan memeriksa 35 studi. Hasil mereka menunjukkan bahwa proses memecah monolit menjadi layanan mikro masih dalam tahap awal, dan terdapat kekurangan teknik untuk mengintegrasikan data statis, dinamis, dan evolusioner. Kurangnya dukungan alat yang memadai juga terlihat jelas. Penulis melakukan SLR mereka menggunakan prinsip-prinsip Barbara Kitchenham sebagai panduan, seperti yang kami ilustrasikan di Bagian III. Penulis menyarankan untuk fokus pada penyebaran layanan mikro dan menstandarisasi pengukuran analisis.

Tapia et al. [26] menilai kinerja dan korelasi aplikasi monolitik dan mikroservis. Mereka melakukan uji stres pada hasil mereka menggunakan karakteristik dan spesifikasi perangkat keras yang sama. Selanjutnya, model matematika menggunakan metode regresi non-parametrik memverifikasi temuan studi mereka. Hasil mereka menunjukkan bahwa aplikasi monolitik dan mikroservis dapat melayani berbagai situasi teknologi.

Namun, arsitektur microservices meningkatkan efisiensi sumber daya perangkat keras, penghematan biaya, dan produktivitas. Arah pengembangan di masa depan mencakup peningkatan keamanan informasi dan memerangi serangan siber. Selain itu, diperlukan alat otomatisasi untuk migrasi.

Kuryazov et al. [27] mengusulkan model konseptual untuk menyelesaikan masalah migrasi dari arsitektur monolitik ke microservices, khususnya langkah-langkah dekomposisi. Model konseptual mereka masih dalam tahap awal dan membutuhkan lebih banyak evaluasi dan pengujian di industri nyata. Mereka perlu mengembangkan metode untuk mengevaluasi perangkat lunak menggunakan pengukuran kohesi dan kopling, yang akan menyederhanakan analisis monolitik.
Mereka perlu menganalisis sistem dan memperkirakan upaya migrasi. Selain itu, mereka harus membuat alat yang memfasilitasi ekstraksi metadata perangkat lunak dan logika bisnis.

Auer et al. [28] menyajikan kerangka kerja pendukung keputusan untuk perusahaan perangkat lunak yang ingin beralih ke layanan mikro.
Kerangka kerja mereka didasarkan pada pemeriksaan serangkaian karakteristik dan metrik, yang mereka kumpulkan dan tinjau melalui wawancara dengan para ahli. Temuan mereka memberikan data.

dan pengukuran yang dapat digunakan perusahaan untuk menilai adopsi microservices. Pekerjaan selanjutnya meliputi validasi kerangka kerja, mengidentifikasi ukuran yang dapat diterapkan secara otomatis yang dapat dengan mudah mengurangi subjektivitas pengambilan keputusan, dan menambahkan teknologi cloud-native serta arsitektur micro-frontend.

Daoud et al. [7] menggunakan proses bisnis untuk mengidentifikasi kebutuhan, data, dan semantik untuk menangkap ketergantungan antar proses ini, serta teknik pengelompokan kolaboratif untuk merekomendasikan layanan mikro. Hasil menunjukkan bahwa pendekatan ini mengungguli pendekatan serupa untuk identifikasi layanan mikro dan menyoroti pentingnya proses bisnis. Arah masa depan adalah sebagai berikut: menghasilkan hubungan aktivitas baru dengan memanfaatkan pembelajaran mesin yang kuat, termasuk NLP, dan mengevaluasi berbagai model ketergantungan aktivitas untuk identifikasi layanan mikro. Identifikasi layanan mikro dapat dipengaruhi oleh masalah keamanan, yang dapat menjadi perkembangan yang menarik.

Hasan et al. [29] menyajikan kumpulan metrik arsitektur perangkat lunak, termasuk kopling, kompleksitas, kohesi, dan ukuran, untuk menilai pemeliharaan desain arsitektur microservice. Hasil menunjukkan bahwa kriteria metrik yang disarankan lebih sesuai untuk implementasi di lingkungan industri. Di sisi lain, studi kasus dari sektor industri dunia nyata perlu dianalisis dan diterapkan pada metrik yang disarankan untuk menilai efektivitasnya. Selanjutnya, metodologi berbasis alat harus dikembangkan untuk mengevaluasi kualitas arsitektur microservice potensial.

Oumoussa et al. [30] melakukan tinjauan literatur sistematis yang menyoroti area kritis yang membutuhkan perhatian lebih, seperti alat identifikasi otomatis yang ditingkatkan dan standar evaluasi yang terstandarisasi. Temuan mereka menunjukkan bahwa banyak teknik yang ada untuk mengidentifikasi microservices; namun, teknik-teknik tersebut seringkali berfokus pada tantangan tertentu dan mengabaikan tantangan lainnya. Selain itu, masih minim penelitian yang berfokus pada solusi untuk mengatasi masalah migrasi.

Abdellatif et al. [31] melakukan analisis komprehensif terhadap 41 studi. Penelitian mereka bertujuan untuk mengidentifikasi berbagai masukan, proses, keluaran, dan kegunaan metodologi identifikasi layanan untuk memodernisasi perangkat lunak monolitik. Temuan mereka menunjukkan bahwa kategorisasi tersebut selaras dengan pengalaman para profesional industri dan memberikan bantuan berharga kepada praktisi di lingkungan industri dunia nyata. Arah masa depan mencakup usulan pendekatan untuk mengidentifikasi layanan berdasarkan jenisnya, yang meningkatkan potensi untuk menggunakannya kembali di beberapa tingkatan: aplikasi, perusahaan, dan bisnis, selain menggeneralisasi hasil.

Li et al. [32] mengusulkan teknik untuk mengidentifikasi microservice dengan memanfaatkan Unified Model Language (UML), yang diturunkan dari kode sumber. Kemudian, kelas dan diagram urutan dianalisis, menggunakannya sebagai input untuk teknik pengelompokan untuk mengidentifikasi microservice potensial. Selain itu, eksperimen dilakukan untuk mengevaluasi model yang diusulkan dan membandingkannya dengan metode terkini. Temuan mereka menunjukkan bahwa model yang diusulkan mengungguli model yang ada. Namun, hasil mereka tidak dapat digeneralisasikan karena model yang diusulkan mengabaikan distribusi microservice dan kriteria kualitas.

Fokus utama Gomes Barbosa dkk. [33] adalah untuk mengidentifikasi potensi microservice dari prosedur basis data, khususnya menargetkan aplikasi monolitik yang dikembangkan pada tahun 1980-an dan 1990-an yang menggunakan prosedur basis data. Bukti konsep mereka berkontribusi pada identifikasi kode duplikat, meningkatkan pemeliharaan sistem. Untuk arah selanjutnya, penulis merekomendasikan penggunaan algoritma pembelajaran mesin untuk mengotomatisasi proses sepenuhnya dan metode pengujian blackbox/whitebox untuk memverifikasi dan memvalidasi microservice yang diekstrak.

Al-Debagy et al. [34] menguraikan arsitektur monolitik menjadi arsitektur microservices menggunakan model jaringan saraf (code2vec). Temuan mereka menunjukkan hasil yang lebih baik dibandingkan dengan algoritma lain. Selain itu, penulis memvalidasi hasil mereka dengan menggunakan pengukuran kualitas seperti tingkat pesan (CHM) dan kohesi pada tingkat domain (CHD). Arah masa depan dapat mencakup pengembangan dan pengujian lebih lanjut dari model yang diusulkan dengan bahasa pemrograman lain, serta pelatihan.

Jin et al. [35] mengusulkan metode Identifikasi Kandidat Layanan Berorientasi Fungsionalitas (FoSCI), yang menggunakan teknik pengelompokan atom fungsional berbasis pencarian untuk mengidentifikasi kandidat layanan dari jejak eksekusi sistem monolitik. Para penulis menilai metode mereka dengan rangkaian evaluasi layanan 8 metrik untuk menganalisis fungsionalitas, modularitas, dan kemampuan evolusi. Selain itu, para penulis mengevaluasi FoSCI terhadap metode lain (LIMBO, WCA, dan MEM) untuk mengevaluasi dampak cakupan jejak eksekusi pada kinerja. Hasil mereka menunjukkan bahwa FoSCI mengungguli metode yang dibandingkan.
Namun, metode mereka memprioritaskan fungsionalitas di atas atribut kualitas lainnya seperti kinerja, keamanan, dan keandalan, yang berpotensi mendapat manfaat dari perluasan di masa mendatang.

Desai et al. [36] memperkenalkan metode Graph Neural Network untuk refactoring sistem monolitik, yang disebut COGCN (Clustering and Outlier-Aware Graph Convolution Net). COGCN ini menggabungkan representasi node, deteksi outlier, dan clustering ke dalam kerangka kerja yang kohesif. Untuk menilai kualitas cluster, empat metrik struktural dan yang diinginkan: modularitas, modularitas struktural, distribusi non-ekstrem (NED), dan jumlah antarmuka (IFN) digunakan. Hasil mereka menunjukkan bahwa COGCN bekerja lebih baik daripada metode lain seperti GCN, Node2Vector, dan DeepWalk dengan meningkatkan kualitas cluster dan menemukan outlier. Pekerjaan selanjutnya akan mencakup penentuan otomatis jumlah microservice dan menangani bahasa pemrograman prosedural.

Kalia et al. [37] mengembangkan pendekatan yang disebut Mono2Micro. Pendekatan ini menggunakan kasus penggunaan bisnis yang terdefinisi dengan baik, dekomposisi spasial-temporal, dan relasi panggilan run-time untuk memisahkan kelas aplikasi secara fungsional. Metode ini bekerja lebih baik daripada metode lain dalam metrik yang terdefinisi dengan baik yang spesifik untuk domain tersebut, seperti FoSCI, MEM, CoGCN, dan Bunch [17], [35], [36], [38]. Alat mereka dievaluasi dengan menggunakan beberapa kriteria untuk memverifikasi efisiensinya. Arah masa depan berfokus pada elaborasi kriteria kualitas, memberikan bantuan lebih lanjut untuk mengembangkan kasus penggunaan yang efektif bagi praktisi, dan menggeneralisasi temuan dengan mendukung berbagai bahasa pemrograman.

Francisco et al. [39] melakukan tinjauan terhadap 20 makalah yang membahas proses migrasi ke microservices. Hasil mereka menunjukkan bahwa mayoritas solusi bergantung pada desain

aspek, grafik ketergantungan sistem, dan algoritma pengelompokan.
Selain itu, sebagian besar studi bergantung pada grafik dan mengkategorikannya ke dalam layanan mikro; 70% berfokus pada sistem berbasis web, sebagian besar menggunakan Java sebagai bahasa pemrograman utama.
Namun, penelitian mereka menghadapi keterbatasan karena bergantung pada satu mesin pencari dan bias peneliti individu.
Selain itu, dokumen ini menganjurkan berbagai macam metodologi migrasi dan investigasi migrasi basis data yang lebih luas.

Justas et al. [40] berkonsentrasi pada hambatan dan pendekatan untuk memigrasikan perangkat lunak ke layanan mikro dengan meninjau dan menganalisis berbagai teknik migrasi; namun, mereka tidak menyebutkan metrik evaluasi khusus yang digunakan untuk menilai tinjauan mereka. Hasil mereka menekankan bahwa proses migrasi itu kompleks dan mahal, tanpa solusi yang cocok untuk semua. Penelitian masa depan mencakup fokus pada pengembangan teknik migrasi standar untuk mengatasi berbagai masalah yang ditimbulkan oleh sistem lama.

Ren et al. [41] mengintegrasikan analisis statis dan dinamis untuk memeriksa karakteristik sistem monolitik. Mereka menggunakan pengelompokan fungsi untuk memfasilitasi migrasi dan pengelompokan hierarkis untuk mengidentifikasi kandidat layanan mikro.
Mereka menguji metode mereka dengan membandingkan kinerja di empat aplikasi benchmark dan memvalidasi algoritma migrasi dengan 12 aplikasi industri dan sumber terbuka.
Temuan mereka menunjukkan bahwa metode yang mereka usulkan secara efektif memigrasikan aplikasi monolitik ke layanan mikro dengan akurasi tinggi dan biaya kinerja rendah. Namun, analisis statis yang tidak lengkap karena kurangnya pemanggilan fungsi dan interaksi pengguna mungkin masih membatasi kelengkapan migrasi.

Fritzsch et al. [42] melakukan studi kualitatif menggunakan wawancara semi-terstruktur dalam konteks migrasi ke microservices. Para penulis melakukan 16 wawancara komprehensif dengan spesialis dari 10 perusahaan perangkat lunak di 14 kasus migrasi. Temuan mereka menunjukkan bahwa motivasi terbesar untuk migrasi adalah pemeliharaan dan skalabilitas.

Selain itu, banyak organisasi memilih pembangunan ulang total daripada pembubaran kode program. Masalah utama meliputi identifikasi pemotongan layanan yang tepat dan pengembangan kemampuan dalam teknologi baru. Namun, prosedur pengambilan sampel, yang hanya berfokus pada 14 kasus dan individu yang berlokasi di Jerman, membatasi penelitian mereka dan memengaruhi kemampuan generalisasinya.

Kalske et al. [43] melakukan tinjauan literatur yang berfokus pada migrasi arsitektur dan tantangan terkait. Temuan mereka menunjukkan bahwa organisasi menggunakan microservices untuk mengurangi kompleksitas, meningkatkan skalabilitas, dan menyelesaikan tantangan kepemilikan kode. Meskipun demikian, restrukturisasi dan dekopling monolit yang terhubung erat tetap menjadi tantangan yang signifikan. Namun, tinjauan mereka kurang kredibel dan verifikasi hasilnya, dan tidak ada pedoman untuk melakukan tinjauan tersebut. Arah masa depan mereka mencakup pemahaman tentang bagaimana berbagai struktur organisasi memengaruhi efektivitas adopsi microservice.

Vainio et al. [44] menggabungkan studi kasus dan tinjauan literatur untuk mengekstrak fungsionalitas dari sistem monolitik, kemudian menggunakan microservices untuk mendemonstrasikan manfaat dan tantangan dunia nyata yang realistis untuk proses migrasi. Temuan mereka menunjukkan keuntungan signifikan dalam skalabilitas, pemeliharaan, dan kemampuan untuk menggunakan beberapa bahasa pemrograman untuk layanan yang berbeda. Selain itu, penerapan microservices secara independen meningkatkan stabilitas dan kinerja sistem.
Namun, studi kasus mereka tidak secara menyeluruh meneliti kompleksitas arsitektur dan implikasi keamanan dari pengelolaan beberapa layanan mikro dalam sistem yang luas, yang menunjukkan perlunya penelitian tambahan tentang topik-topik ini.

Eski et al. [45] menggunakan metodologi pengelompokan berbasis grafik yang memanfaatkan kopling kode statis dan evolusioner untuk menghasilkan layanan mikro dari aplikasi monolitik.
Mereka mengevaluasi metode mereka dengan menilai dua proyek, yang mengungkapkan tingkat keberhasilan 89%; namun, layanan tertentu memerlukan sub-clustering manual karena ukurannya. Mereka berencana untuk melanjutkan penelitian mereka dengan menambahkan bobot pada edge graf ke metode mereka, mengotomatiskan ambang batas sub-clustering, dan mengujinya pada lebih banyak proyek menggunakan algoritma yang berbe

Su et al. [46] melakukan tinjauan literatur sistematis dengan memeriksa 32 studi untuk mendapatkan wawasan tentang bagaimana perusahaan perangkat lunak bermigrasi dari arsitektur microservices kembali ke arsitektur monolitik. Studi mereka mengidentifikasi biaya, kompleksitas, skalabilitas, kinerja, dan faktor organisasi sebagai alasan utama untuk kembali ke arsitektur monolitik. Arah penelitian selanjutnya mencakup menyelidiki fenomena ini di berbagai industri dan melakukan penilaian empiris untuk menggeneralisasi hasilnya.

Bandara et al. [47] mengembangkan sebuah toolkit yang menggunakan fungsi iteratif fitness untuk mengidentifikasi microservice dalam sistem monolitik. Toolkit mereka menggunakan metrik kualitas layanan, termasuk fungsionalitas, komposisi, kemandirian, dan penggunaan. Mereka membandingkan alat tersebut dengan identifikasi microservice manual. Hasil mereka menunjukkan bahwa alat mereka menghasilkan microservice seperti identifikasi manual. Arah masa depan mencakup dukungan untuk lebih banyak bahasa pemrograman dan pola arsitektur, model pembelajaran mesin, dan pengetahuan konteks untuk meningkatkan ekstraksi microservice.

Michael Ayas dkk. [48] meneliti migrasi ke microservices melalui wawancara dengan 19 individu dari 16 organisasi. Hasil penelitian mereka menunjukkan bahwa proses migrasi bersifat iteratif dan berlangsung pada dua tingkatan: arsitektur dan tingkat sistem. Selain itu, mereka mengkategorikan aktivitas utama ke dalam empat fase: merancang arsitektur, mengubah sistem, menyiapkan artefak pendukung, dan mengimplementasikan artefak teknis. Studi ini mengakui bias peneliti dan keterwakilan sampel. Arah penelitian selanjutnya meliputi pemeriksaan lebih banyak jalur migrasi dan validasi hasil di berbagai organisasi.

Taibi et al. [49] memperkenalkan kerangka kerja penambangan proses untuk membantu dekomposisi sistem monolitik. Kerangka kerja mereka mengidentifikasi proses bisnis berdasarkan jejak log, mengelompokkan proses yang serupa, dan menggunakan metrik untuk mengevaluasi kualitas dekomposisi. Seorang arsitek perangkat lunak memvalidasi kerangka kerja mereka dengan membandingkan hasilnya dengan dekomposisi manual. Kerangka kerja mereka mengungkapkan alternatif dekomposisi dan masalah perangkat lunak yang terlewatkan oleh analisis manual dalam studi kasus industri. Arah masa depan meliputi otomatisasi proses, validasi metodologi, dan integrasi pola untuk konektivitas layanan mikro.

Silva et al. [50] melakukan dua studi kasus. Studi ini mengidentifikasi langkah-langkah migrasi dan tantangan dari sistem lama ke sistem baru.

microservices. Temuan mereka mengungkapkan empat isu utama: 1) mengidentifikasi fungsionalitas dalam modul besar; 2) mendefinisikan batasan microservice yang optimal; 3) memilih fitur untuk dikonversi menjadi microservices; dan 4) menganalisis granularitas dan kohesi kandidat microservice. Langkah selanjutnya termasuk menyempurnakan dan mendukung pedoman praktis untuk bermigrasi ke arsitektur microservice, serta melakukan survei kepada praktisi industri untuk mempelajari lebih lanjut tentang tantangan migrasi.

Kholy et al. [51] memperkenalkan kerangka kerja manajemen basis data untuk arsitektur layanan mikro (MDMA). Kerangka kerja mereka menunjukkan bahwa MDMA secara signifikan mengurangi waktu eksekusi dan ukuran transfer data dibandingkan dengan pendekatan terpusat dan menunjukkan kinerja yang lebih unggul seiring meningkatnya volume permintaan. Selain itu, hal ini meningkatkan fleksibilitas dan ketahanan, mengurangi dampak kegagalan layanan dan memfasilitasi transfer data selama penyebaran layanan.

Penelitian selanjutnya dapat mencakup evaluasi kerangka kerja di berbagai lingkungan dunia nyata dan memperluas penerapannya ke berbagai arsitektur layanan mikro.

Antunes et al. [52] menyelidiki migrasi aplikasi dunia nyata ke arsitektur micro-frontend. Hasil menunjukkan bahwa micro-frontend meningkatkan fleksibilitas, skalabilitas tim, dan migrasi inkremental. Meskipun demikian, mereka mengamati peningkatan kompleksitas dalam pengelolaan dependensi, lingkungan, debugging, dan pengujian. Studi masa depan harus menyelidiki proyek dan metodologi implementasi lainnya, dengan fokus pada penyederhanaan manajemen dependensi dan pengujian integrasi.

Maria et al. [53] menyelidiki migrasi menuju micro-frontend. Studi mereka menunjukkan bagaimana cara berhasil mengimplementasikan ulang aplikasi halaman tunggal (SPA) menggunakan arsitektur micro-frontend dengan mengadopsi Webpack untuk menggabungkan modul dan Cypress untuk pengujian. Temuan mereka menunjukkan peningkatan dalam kolaborasi tim, penyebaran independen, dan kinerja. Studi lebih lanjut mencakup pengelolaan dependensi, peningkatan integrasi, dan evaluasi kinerja.

Fritzsch et al. [54] menggunakan SLR untuk mengklasifikasikan 10 pendekatan yang ada berdasarkan teknik dekomposisi. Temuan mereka mengidentifikasi kurangnya pendekatan yang berlaku universal dengan dukungan alat yang memadai. Penelitian selanjutnya akan menggabungkan analisis kode statis dengan data runtime, membuat metrik kualitas dekomposisi, dan mengotomatiskan proses migrasi.

Lauretis [55] menyajikan strategi untuk bermigrasi ke microservice. Penulis menyoroti potensi manfaat seperti peningkatan skalabilitas, pemeliharaan, dan evolusi bagi perusahaan. Strategi mereka terdiri dari lima langkah, termasuk analisis fungsi, identifikasi bisnis, analisis bisnis, penugasan fungsionalitas, dan pembuatan microservice. Namun, strategi mereka masih dalam tahap awal. Arah masa depan meliputi otomatisasi proses migrasi dan pengujian strategi pada sistem nyata untuk mengumpulkan data kinerja dan statistik.

Nunes et al. [56] menggunakan konteks transaksional untuk mengusulkan strategi migrasi microservices. Mereka menggunakan kode statis Analisis dilakukan untuk mengidentifikasi entitas domain dan algoritma pengelompokan diterapkan untuk mengelompokkannya. Perbandingan dengan dekomposisi ahli menghasilkan hasil yang menjanjikan. Pekerjaan selanjutnya akan meningkatkan metode dan memperluas hasilnya, sambil membatasi penerapannya pada kerangka kerja dan alat tertentu.

Santos et al. [57] mengusulkan metrik kompleksitas untuk mengukur transisi ke microservices berdasarkan empat ukuran kesamaan yang memeriksa dekomposisi entitas. Ukuran-ukuran ini berfokus pada himpunan baca dan tulis, urutan akses, dan biaya pelonggaran konsistensi transaksional. Metrik mereka dievaluasi menggunakan tiga sistem monolitik. Metrik kompleksitas mereka berhasil mengidentifikasi dekomposisi yang paling kompleks.

Arah pengembangan ke depan mencakup penggunaan kombinasi data dinamis dan statis, serta eksperimen lebih lanjut dengan berbagai sistem yang lebih beragam.

Ma et al. [58] mengusulkan Identifikasi Layanan Mikro menggunakan Analisis untuk Akses Basis Data (MIADA) untuk memperhitungkan pentingnya "Basis Data Per Layanan" dalam desain layanan mikro.
Pendekatan mereka memfasilitasi pengelompokan titik akhir layanan untuk identifikasi layanan mikro. Dua proyek perangkat lunak berorientasi layanan (PlanApproval dan CoCoME) mengevaluasi hasilnya, menunjukkan bahwa MIADA dapat berhasil merekomendasikan klaster titik akhir layanan untuk layanan mikro. Arah masa depan mencakup menghasilkan hasil dan meningkatkan MIADA.

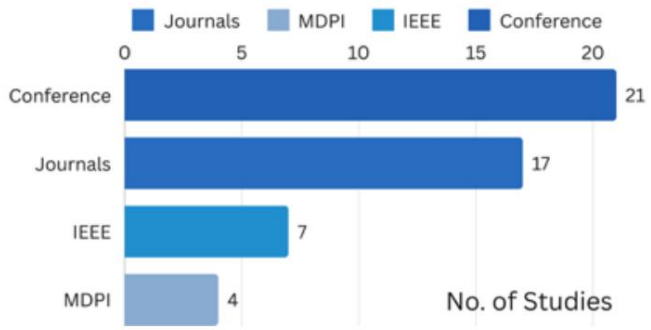
Ghofrani et al. [59] mengkategorikan tantangan migrasi menjadi inersia, kecemasan, dan konteks berdasarkan 17 wawancara ahli semi-terstruktur. Hambatan yang paling signifikan adalah kecemasan migrasi, diikuti oleh inersia dan konteks. Selanjutnya, mereka memberikan saran untuk mengatasi hambatan-hambatan ini. Arah masa depan mencakup evaluasi solusi mereka dan pembuatan metrik kualitas.

Ghofrani et al. [60] melakukan survei terhadap para ahli industri untuk mengidentifikasi tantangan dalam arsitektur microservices. Mereka
Mereka mengidentifikasi tantangan-tantangan kritis, termasuk kurangnya notasi, metode, dan kerangka kerja untuk desain layanan mikro, serta kurangnya alat untuk memilih artefak pihak ketiga. Selain itu, mereka memprioritaskan keamanan, waktu respons, dan kinerja daripada ketahanan dan toleransi kesalahan. Pekerjaan selanjutnya mencakup perluasan ruang lingkup dan menyarankan solusi untuk kesenjangan ini.

Razzaq et al. [61] menyediakan studi pemetaan sistematis tentang migrasi menuju microservices. Tinjauan mereka menekankan manfaat utama seperti: termasuk penyebaran independen, skalabilitas, dan mekanisme ringan setelah migrasi.
Selain itu, penelitian ini mengidentifikasi tantangan migrasi dan faktor-faktor keberhasilan yang membantu memandu strategi migrasi. Arah penelitian selanjutnya meliputi, mengidentifikasi solusi efektif untuk area-area ini, serta menyediakan riset yang lebih mendalam untuk layanan mikro dalam teknologi yang sedang berkembang.

V. HASIL

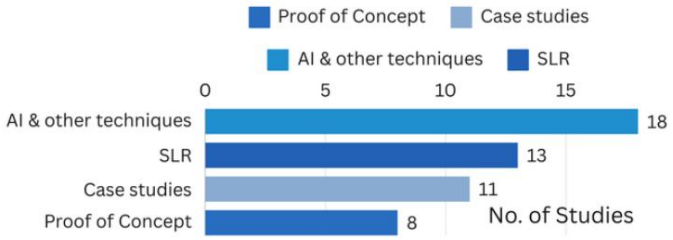
Seperti yang ditunjukkan pada Gambar 4, SLR ini menganalisis 50 studi yang diterbitkan antara tahun 2018 dan 2024 yang berfokus pada proses migrasi dari arsitektur monolitik ke arsitektur microservice. Selanjutnya, Tabel IV memberikan gambaran umum temuan dari penelitian literatur ini, yang mencakup topik-topik berikut:



Gambar 4. Klasifikasi studi yang diteliti dalam SLR ini.

A. Algoritma AI dan Teknik Lainnya:

Beberapa penulis menekankan pentingnya memberikan bukti konsep untuk mendukung ide-ide mereka dan memberikan wawasan untuk proses migrasi. Beberapa studi melakukan studi kasus industri dunia nyata, mencoba untuk menilai dan melacak perbedaan kinerja, upaya, skalabilitas, dan pemeliharaan antara arsitektur monolitik dan mikroservis serta membandingkannya. Para peneliti melakukan penelitian ekstensif tentang AI dan teknik algoritmik lainnya, termasuk algoritma evolusi berbasis indikator (IBEA), jaringan saraf, algoritma berbasis pencarian, algoritma pengelompokan, kopling, dan kohesi. Selain itu, beberapa studi menggunakan SLR untuk meningkatkan pemahaman mereka tentang arsitektur mikroservis. Gambar 5 menyajikan distribusi studi, menunjukkan penggunaan AI dan teknik pengelompokan lainnya dalam 18 studi, SLR dalam 13 studi, bukti konsep dalam 8 studi, dan studi kasus dalam studi yang tersisa.

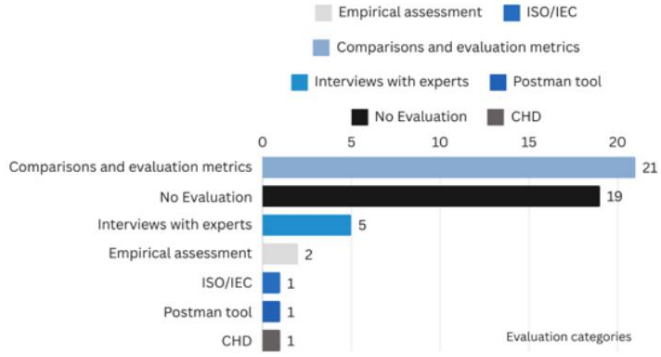


Gambar 5. Distribusi studi berdasarkan metodologi.

B. Teknik Evaluasi:

Studi-studi utama dievaluasi dan divalidasi menggunakan berbagai teknik, termasuk penilaian empiris menggunakan tolok ukur sistem perangkat lunak, definisi atribut kualitas standar ISO/IEC 25012:2008, perbandingan kinerja, skalabilitas, dan keandalan, serta perbandingan desain arsitektur.

Studi lain menggunakan kohesi pada tingkat domain (CHD), melakukan wawancara dengan pakar industri perangkat lunak, dan menggunakan alat Postman untuk evaluasi dan pengujian. Gambar 6 menyajikan kategori evaluasi.



Gambar 6. Kategorisasi metode evaluasi yang diteliti dalam SLR ini.

C. Keterbatasan dan Pekerjaan di Masa Depan

Sebagian besar studi di bidang migrasi perangkat lunak, khususnya yang berfokus pada layanan mikro, kurang melakukan validasi dan verifikasi model mereka untuk sistem perangkat lunak dunia nyata, sehingga menyebabkan kekurangan dalam menggeneralisasi hasil studi tersebut. Kesenjangan ini menekankan perlunya penelitian empiris lebih lanjut dan implementasi aktual untuk menguji model migrasi dalam berbagai skenario dunia nyata guna memastikan kekokohan dan keandalannya untuk penggunaan yang lebih luas. Berikut adalah daftar arah dan area analisis serta investigasi di masa mendatang:

1) Pertimbangkan kriteria non-fungsional.

2) Perspektif arsitek dan pengembang perangkat lunak.

3) Diperlukan alat otomatis untuk dekomposisi manual.

4) Menggeneralisasi hasil dengan menguji pada industri dunia nyata.

5) Dampak server cloud terhadap proses migrasi sangat signifikan. Hyperscale seperti Azure, AWS, dan GCP menawarkan fitur unik seperti opsi skalabilitas, alokasi sumber daya yang fleksibel, dan alat DevOps.

6) Mengadopsi micro-frontend bersamaan dengan microservices.

Diskusi dan jawaban selanjutnya sesuai dengan hal-hal berikut: Pertanyaan penelitian diuraikan dalam Bagian III:

RQ1: Terdapat kekurangan yang signifikan dalam penelitian tentang metode AI dan ML dalam migrasi perangkat lunak ke layanan mikro.
domain. Hanya sedikit penelitian yang menggunakan algoritma seperti algoritma evolusi berbasis indikator (IBEA), teknik berbasis pencarian, dan algoritma pengelompokan. Penelitian lain mengandalkan perbandingan dan masukan ahli untuk menilai kriteria kualitas perangkat lunak. Penelitian yang tersisa menawarkan bukti konsep tetapi tidak memberikan solusi komprehensif. Solusi komprehensif akan melibatkan metode atau algoritma otomatis yang memanfaatkan AI untuk menyederhanakan transisi dari proses monolitik ke proses layanan mikro sambil mempertahankan kinerja, skalabilitas, dan ketersediaan. Selain itu, sistem perangkat lunak industri dunia nyata harus mengevaluasi solusi tersebut.

RQ2: Tantangan dan hambatan paling signifikan dalam domain ini muncul dari bagaimana gudang perangkat lunak dan arsitek

Memecah logika dan basis data sistem perangkat lunak menjadi puluhan layanan mikro yang saling terhubung, mempersulit jaminan komunikasi yang efektif, konsistensi data, dan pelestarian integritas sistem secara keseluruhan. Lebih lanjut, penting untuk mempertimbangkan cara layanan-layanan ini akan berinteraksi dan berkoordinasi satu sama lain. Selain itu, arsitektur layanan mikro harus memenuhi kriteria non-fungsional seperti kinerja, skalabilitas, keamanan, dan keandalan. Seperti yang disarankan dalam [7], AI dapat membantu dengan menggunakan teknik ML yang ampuh seperti NLP untuk menganalisis kode, dokumentasi, dan persyaratan sistem, serta teknik pengelompokan yang kompleks untuk menguraikan logika sistem menjadi beberapa layanan. Untuk mengatasi kriteria non-fungsional, mungkin bermanfaat untuk memantau dan memanfaatkan teknologi, seperti komputasi awan dan metodologi DevOps.

RQ3: Evaluasi migrasi perangkat lunak dari arsitektur monolitik ke arsitektur mikroservis sebagian besar didasarkan pada perbandingan kedua arsitektur dan meminta pendapat serta rekomendasi dari para ahli. Meskipun demikian, model AI dan algoritma lainnya perlu dievaluasi dan diuji secara ketat di industri nyata untuk mengumpulkan umpan balik dari organisasi dan pengguna akhir, sehingga memungkinkan kita untuk memantau kinerja dan memperoleh wawasan yang relevan.

VI. KESIMPULAN

Studi ini melakukan tinjauan literatur sistematis untuk mengeksplorasi tantangan, hambatan, dan peningkatan utama dalam industri perangkat lunak, khususnya migrasi dari arsitektur monolitik ke arsitektur layanan mikro. Meskipun terdapat peningkatan minat dalam mengembangkan AI dan teknik algoritmik lainnya untuk memfasilitasi proses migrasi ini, bidang ini masih dalam tahap awal, dan masih kekurangan solusi komprehensif dan menyeluruh yang dapat mengatasi tantangan tersebut. kompleksitas penuh dari proses tersebut. Sebagian besar studi yang ada berfokus pada kerangka kerja teoretis, bukti konsep, dan penilaian ahli daripada memverifikasi dan memvalidasi pendekatan ini melalui implementasi di dunia nyata.

Tabel IV menyajikan ringkasan tinjauan literatur ini, yang mencakup daftar studi, masing-masing menjelaskan metode yang digunakan, teknologi evaluasi, hasil, keterbatasan, dan pekerjaan di masa mendatang. Bagian V membahas jawaban rinci untuk tiga pertanyaan penelitian. Keduanya akan membantu pengembang perangkat lunak dan penulis lain dengan menyediakan dasar untuk solusi komprehensif dalam mengatasi tantangan migrasi.

Hasil studi literatur ini menyoroti kesenjangan yang signifikan dalam pembuatan alat migrasi yang efektif dan validasi solusi tersebut dalam industri perangkat lunak. Hal ini menekankan perlunya penelitian di masa mendatang untuk fokus pada pengembangan, evaluasi, dan validasi alat otomatis di lingkungan dunia nyata. Lebih lanjut, proses migrasi harus menekankan persyaratan non-fungsional seperti kinerja, skalabilitas, keandalan, dan keamanan. Mengintegrasikan metodologi DevOps dengan konsep komputasi awan sangat penting untuk mengoptimalkan keunggulan arsitektur microservice.

Dengan mengidentifikasi kesenjangan ini, tinjauan ini menekankan pentingnya menangani dimensi teoritis dan praktis dari proses migrasi, serta menyediakan peta jalan untuk penelitian masa depan yang bertujuan untuk meningkatkan efisiensi, efektivitas, dan skalabilitas migrasi perangkat lunak. Penyelesaian tantangan ini pada akhirnya akan meningkatkan kemampuan industri untuk bermigrasi dari sistem monolitik ke arsitektur berbasis layanan mikro modern.

TABEL IV. RINGKASAN SINGKAT TEMUAN PENELITIAN DARI LITERATUR

R	Teknik yang digunakan	Hasil	Keterbatasan dan pekerjaan di masa mendatang
[1]	IBEA	Pengujian empiris menunjukkan bahwa pendekatan yang diusulkan lebih baik. efektif.	Mempertimbangkan evaluasi non-fungsional dan perangkat lunak menggunakan.
[19]	Bukti konsep	Migrasi penyimpanan data monolitik ke layanan mikro dapat diterapkan.	Diperlukan otomatisasi, dan hasilnya tidak dapat digeneralisasikan.
[18]	Tinjauan Pustaka	Pola untuk arsitektur microservices. Perbandingan microservices dan arsitektur monolitik: pro dan kontra.	Diperlukan evaluasi dan dekomposisi otomatis.
[20]	Pendekatan gabungan	Migrasi berbasis kustomisasi dapat mengarahkan sistem monolitik menjadi SaaS.	Menggeneralisasi hasil dan mengotomatiskan proses migrasi.
[21]	Tinjauan Pustaka	Daftar tantangan dan manfaat yang muncul dalam proses migrasi.	Analisis kelebihan dan kekurangan arsitektur serta otomatiskan migrasi.
[11]	Studi Kasus	Monolit modular membantu migrasi, konsistensi, dan komunikasi.	Generalisasikan dan terapkan hasilnya pada studi kasus lain.
[12]	Studi Kasus	Arsitektur monolitik mengungguli arsitektur microservices, Java lebih unggul dalam hal komputasi.	Meningkatkan kompleksitas sistem dan melakukan deployment ke berbagai cloud.
[22]	Studi Kasus	Model mereka mengoptimalkan proses pengembangan.	Generalisasi hasil tersebut memerlukan perbaikan model.
[8]	Studi Kasus	Arsitektur microservices meningkatkan skalabilitas dan memecahkan masalah arsitektur monolitik. masalah.	Memperluas pembuktian konsep untuk menggeneralisasi hasilnya.
[10]	Teknik berbasis pencarian	Strategi yang menjanjikan ini dapat diterapkan pada sistem monolitik lainnya.	Model tersebut perlu dapat dikonfigurasi dan digeneralisasi.
[23]	Studi Kasus	Migrasi tersebut memengaruhi para pemangku kepentingan. Waktu unggah memengaruhi penerapan.	Dukungan aplikasi server cloud dan generalisasi temuan.
[24]	Studi Kasus	Arsitektur monolitik cocok untuk aplikasi kecil hingga menengah, tetapi arsitektur microservices lebih mudah diskalakan.	Pengujian di dunia nyata. Memastikan keamanan di antara layanan mikro.
[25]	Tinjauan Pustaka	Migrasi microservices masih dalam tahap awal, dengan sedikit metode yang tersedia.	Pertimbangkan metrik penyebaran dan identifikasi layanan mikro.
[26]	Studi Kasus	Arsitektur microservices meningkatkan efisiensi perangkat keras, produktivitas, dan biaya.	Pengembangan alat migrasi dan pengamanan microservices sangat dibutuhkan.
[27]	Bukti konsep	Model konseptual ini menyelesaikan tahapan migrasi akibat kerusakan sistem.	Diperlukan alat ekstraksi logika bisnis perangkat lunak.

[28] Kerangka pengambilan keputusan	Kumpulan alat ukur adopsi microservices yang lengkap untuk perusahaan.	Memvalidasi kerangka kerja dan menambahkan teknologi cloud-native.
[7] Teknik pengelompokan	Arsitektur microservices bekerja lebih baik dan mengurangi tekanan pada proses bisnis.	Menggunakan AI untuk mengidentifikasi layanan mikro dan masalah keamanan.
[29] Kopling dan kohesi Kriteria metrik yang disarankan lebih baik untuk penggunaan industri.		Mengevaluasi penilaian microservice menggunakan studi kasus dunia nyata.
[30] Tinjauan Pustaka	Tekankan standar identifikasi dan evaluasi otomatis.	Studi-studi tersebut berfokus pada tantangan spesifik dan kurangnya alat bantu migrasi.
[31] Tinjauan Pustaka	Studi ini menunjukkan bahwa kategorisasi membantu para profesional di bidang industri.	Pengelompokan layanan berdasarkan jenisnya dapat membantu menggeneralisasi hasil.
[32] Teknik pengelompokan	Hasil penelitian menunjukkan bahwa model yang diusulkan memiliki kinerja yang lebih baik daripada model lainnya.	Menggeneralisasi hasil.
[33] Bukti konsep	Model tersebut menemukan duplikasi dan meningkatkan kemudahan pemeliharaan.	Otomatisasi menggunakan ML dan black/white box untuk validasi.
[34] Jaringan saraf	Algoritma yang diusulkan berkinerja lebih baik daripada algoritma lainnya.	Menguji model dalam bahasa lain dan melatihnya menggunakan kode sumber.
[37] Teknik pengelompokan Mono2Micro mengungguli metode lain.		Jelaskan secara rinci kriteria kualitas dan kembangkan penggunaan yang efektif. kasus.
[35] Teknik berbasis pencarian	FoSCI lebih unggul dalam kualitas layanan, fungsionalitas, dan modularitas.	Performa, keamanan, dan keandalan diabaikan.
[36] Teknik pengelompokan COGCN meningkatkan kualitas klaster dan identifikasi outlier.		Menentukan jumlah microservice dan bahasa prosedural.
[16] Teknik pengelompokan FoME menyediakan layanan mikro yang kohesif dengan kopling yang lebih longgar.		Pastikan penggunaan kasus uji berkualitas tinggi dan otomatiskan prosesnya.
[39] Tinjauan Pustaka	Sebagian besar penelitian menggunakan desain, grafik ketergantungan, dan pengelompokan. Hasil yang bias dan lebih banyak metode migrasi dianjurkan.	
[40] Tinjauan Pustaka	Migrasi itu rumit dan mahal, tanpa solusi tunggal.	Metode migrasi standar untuk mengatasi masalah sistem lama.
[41] Teknik pengelompokan	Metode migrasi microservice mereka akurat dengan biaya rendah.	Menggeneralisasi hasil dengan mengadopsi interaksi pengguna.
[42] Studi kualitatif	Pemeliharaan dan skalabilitas mendorong migrasi.	Kemampuan generalisasi terbatas karena sampel Jerman yang terdiri dari 14 orang.
[43] Tinjauan Pustaka	Demi kesederhanaan, skalabilitas, dan kepemilikan, perusahaan menggunakan microservices.	Hasil mereka kurang kredibilitas dan verifikasi.
[44] Studi Kasus	Arsitektur microservices mudah diskalakan, dipelihara, stabil, dan multibahasa.	Keamanan dan kompleksitas arsitektur memerlukan penelitian lebih lanjut.
[45] Teknik pengelompokan	Metode mereka menunjukkan tingkat keberhasilan ekstraksi microservice sebesar 89%.	Pengujian pada berbagai algoritma pengelompokan untuk meningkatkan model.
[46] Tinjauan Pustaka	Biaya, kompleksitas, dan organisasi mendorong terjadinya Reverting monolitik.	Menganalisis fenomena tersebut di berbagai industri dan melakukan generalisasi.
[47] Perangkat Alat	Toolkit tersebut menghasilkan layanan mikro seperti identifikasi manual.	Mendukung lebih banyak bahasa, pola, model ML, dan konteks.
[48] Studi kualitatif	Migrasi terjadi pada dua tingkatan: tingkat arsitektur dan tingkat sistem.	Bias peneliti dan keterwakilan sampel memengaruhi temuan.
[49] Penambangan proses	Kerangka kerja ini mengungguli analisis manual dalam studi kasus industri.	Mengotomatiskan proses, memvalidasi, dan mengintegrasikan lebih banyak pola.
[50] Studi Kasus	Ukuran, batasan, fitur, dan kohesi modul merupakan tantangan.	Lakukan survei kepada spesialis industri dan sempurnakan migrasi layanan mikro.
[51] Kerangka kerja MDMA MDMA meningkatkan waktu eksekusi, transfer data, fleksibilitas.		Menguji kerangka kerja di dunia nyata dan memperluas penggunaannya.
[52] Studi Kasus	Micro-frontend meningkatkan migrasi, skalabilitas, dan fleksibilitas.	Ketergantungan, debugging, dan pengujian memerlukan penelitian lebih lanjut.
[53] Studi Kasus	SPA meningkatkan penerapan, kolaborasi, dan kinerja.	Mengelola ketergantungan, meningkatkan integrasi dan kinerja.
[54] Tinjauan Pustaka	Kurangnya pendekatan universal dengan dukungan alat yang memadai.	Otomatiskan migrasi dengan analisis kode statis dan data runtime.
[55] Bukti konsep	Meningkatkan skalabilitas, pemeliharaan, dan evolusi perusahaan. Mengotomatiskan proses migrasi dan pengujian pada sistem nyata.	
[56] Teknik pengelompokan	Perbandingan dengan dekomposisi ahli menghasilkan hasil yang menjanjikan.	Perbaiki metode dan generalisasikan hasilnya.
[57] Teknik pengelompokan	Metrik mereka berhasil mengidentifikasi dekomposisi yang kompleks.	Dengan menggunakan data dinamis dan statis, serta investigasi lebih lanjut.
[58] Teknik pengelompokan	MIADA dapat merekomendasikan klaster titik akhir layanan dengan sukses.	Arah ke depan mencakup menghasilkan hasil dan meningkatkan MIADA.
[59] Studi kualitatif	Kecemasan, keengganan untuk berubah, dan konteks merupakan hambatan terbesar dalam migrasi.	Mengevaluasi solusi yang mereka usulkan dan membuat metrik kualitas.
[60] Tinjauan Pustaka	Notasi dan metode desain microservices merupakan tantangan kritis.	Memperluas cakupan dan menyarankan solusi untuk kesenjangan ini.
[61] Tinjauan Pustaka	Menekankan pada penerapan independen, skalabilitas, dan bobot yang ringan.	Menyediakan riset dalam teknologi baru untuk menemukan solusi.

REFERENSI

[1] K. Sellami, A. Ouni, MA Saied, S. Bouktif, dan MW Mkaouer, “Meningkatkan ekstraksi microservice menggunakan pencarian evolusioner,” Inf Softw Technol, vol. 151, Nov. 2022, doi: 10.1016/j.infsof.2022.106996.

[2] S. Newman, “Membangun Layanan Mikro EDISI KEDUA Mendesain Sistem Berbutir Halus.”

[3] F. Ponce, J. Soldani, H. Astudillo, dan A. Brogi, “Bau dan Refactoring untuk Keamanan Microservices: Tinjauan Literatur Multivokal,” April 2021, http:// [On line]. Tersedia: arxiv.org/abs/2104.13303

[4] MG de Almeida dan ED Canedo, “Autentikasi dan Otorisasi dalam Arsitektur Layanan Mikro: Tinjauan Literatur Sistematis,” Ilmu Terapan (Swiss), vol. 12, no. 6, Mar. 2022, doi: 10.3390/app12063023.

[5] I. Saidani, A. Ouni, MW Mkaouer, dan A. Saied, “Menuju Ekstraksi Mikroservis Otomatis Menggunakan Pencarian Evolusi Multiobjektif,” dalam Lecture Notes in Computer Science (termasuk subseri Lecture Notes in Artificial Intelligence dan Lecture Notes in Bioinformatics), Springer, 2019, hlm. 58–63. doi: 10.1007/978-3-030-33702-5_5.

[6] S. Li et al., “Pendekatan berbasis aliran data untuk mengidentifikasi layanan mikro dari aplikasi monolitik,” Jurnal Sistem dan Perangkat Lunak, vol. 157, November 2019, doi: 10.1016/j.jss.2019.07.008.

[7] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, dan A. El Fazziki, “Pendekatan identifikasi layanan mikro berbasis multi-model,” Jurnal Arsitektur Sistem, vol. 118, Sep. 2021, doi: 10.1016/j.sysarc.2021.102200.

[8] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, ST Larsen, dan S. Dustdar, “Layanan Mikro: Migrasi Sistem Kritis Misi,” IEEE Trans Serv Comput, vol. 14, no. 5, hlm. 1464–1477, 2021, doi: 10.1109/TSC.2018.2889087.

[9] P. Di Francesco, P. Lago, dan I. Malavolta, “Migrasi Menuju Arsitektur Layanan Mikro: Survei Industri,” dalam Prosiding - Konferensi Internasional Arsitektur Perangkat Lunak IEEE ke-15, ICSA 2018, Institute of Electrical and Electronics Engineers Inc., Juli 2018, hlm. 29–38. doi: 10.1109/ICSA.2018.00012.

[10] WKG Assunção et al., “Strategi Multi-Kriteria untuk Mendesain Ulang Fitur Warisan sebagai Layanan Mikro: Studi Kasus Industri,” dalam Prosiding - Konferensi Internasional IEEE 2021 tentang Analisis, Evolusi, dan Rekayasa Ulang Perangkat Lunak, SANER 2021, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, hlm. 377–387. doi: 10.1109/SANER50967.2021.00042.

[11] D. Faustino, N. Gonçalves, M. Portela, dan A. Rito Silva, “Migrasi bertahap dari arsitektur monolit ke arsitektur layanan mikro: Evaluasi kinerja dan upaya migrasi,” Evaluasi Kinerja, vol. 164, Mei 2024, doi: 10.1016/j.peva.2024.102411.

[12] G. Blinowski, A. Ojdowska, dan A. Przybylek, “Monolitik vs. Arsitektur Layanan Mikro: Evaluasi Kinerja dan Skalabilitas,” IEEE Access, vol. hlm. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[13] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, dan S. Linkman, “Tinjauan literatur sistematis dalam rekayasa perangkat lunak - Tinjauan literatur sistematis,” doi: 10.1016/j.infsof.2008.09.009. Januari Tahun 2009.

[14] Y. Xiao dan M. Watson, “Panduan Melakukan Tinjauan Literatur Sistematis,” 01 Maret 2019, SAGE Publications Inc. doi: 10.1177/0739456X17723971.

[15] Z. Bai dan T. Wasson, “Melakukan Tinjauan Literatur Sistematis dalam Sistem Informasi: Analisis Pedoman,” 2019. [Online]. Tersedia di: https://www.researchgate.net/publication/340686721

[16] W. Jin, T. Liu, Q. Zheng, D. Cui, dan Y. Cai, “Ekstraksi Layanan Mikro Berorientasi Fungsionalitas Berdasarkan Pengelompokan Jejak Eksekusi,” dalam Prosiding - Konferensi Internasional IEEE 2018 tentang Layanan Web, ICWS 2018 - Bagian dari Kongres Dunia IEEE 2018 tentang Layanan, Institute of Electrical and Electronics Engineers Inc., Sep. 2018, hlm. 211–218. doi: 10.1109/ICWS.2018.00034.

[17] G. Mazlami, J. Cito, dan P. Leitner, “Ekstraksi Layanan Mikro dari Arsitektur Perangkat Lunak Monolitik,” dalam Prosiding - IEEE ke-24 tahun 2017 Konferensi Internasional tentang Layanan Web, ICWS 2017, Institute of Electrical and Electronics Engineers Inc., September 2017, hlm. 524–531. doi: 10.1109/ICWS.2017.61.

[18] V. Velepucha dan P. Flores, “Survei tentang Arsitektur Layanan Mikro: Prinsip, Pola, dan Tantangan Migrasi,” IEEE Access, vol. 11, hlm. 88339–88358, 2023, doi: 10.1109/ACCESS.2023.3305687.

[19] J. Kazanaviÿius, D. Mažeika, dan D. Kalibatienÿ, “Pendekatan untuk Migrasi Basis Data Monolit ke dalam Persistensi Poliglot Multi-Model Berdasarkan Arsitektur Layanan Mikro: Studi Kasus untuk Basis Data Mainframe,” Ilmu Terapan (Swiss), vol. 12, no. 12, Juni 2022, doi: 10.3390/app12126189.

[20] ET Nordli, SG Haugeland, PH Nguyen, H. Song, dan F. Chauvel, “Migrasi monolit ke layanan mikro berbasis cloud untuk SaaS yang dapat disesuaikan,” Inf Softw Technol, vol. 160, Agustus 2023, doi: 10.1016/j.infsof.2023.107230.

[21] V. Velepucha dan P. Flores, “Monolit ke microservices - Masalah dan Tantangan Migrasi: Sebuah SMS,” dalam Prosiding - Konferensi Internasional ke-2 tentang Sistem Informasi dan Teknologi Perangkat Lunak, ICI2ST 2021, Institute of Electrical and Electronics Engineers Inc., Mar. 2021, 135–142. doi: 10.1109/ICI2ST51859.2021.00027. hlm.

[22] A. Bastidas Fuertes, M. Pérez, dan J. Meza, “Model Desain Arsitektur Berbasis Transpiler untuk Lapisan Back-End dalam Pengembangan Perangkat Lunak,” Applied Sciences (Swiss), vol. 13, no. 20, Okt. 2023, doi: 10.3390/app132011371.

[23] Teguh Prasandy, Titan, Dina Fitria Murad, dan Taufik Darwis, “Migrasi Aplikasi dari Monolit ke Mikroservis,” dalam Konferensi Internasional Manajemen dan Teknologi Informasi (ICIMTech) 2020, Bandung, Indonesia: IEEE, 2020, hlm. 726–731. doi: 10.1109/ICIMTech50083.2020.9211252.

[24] AB Fondazione et al., “Migrasi dari Monolit ke Mikroservis : Tolok Ukur Studi Kasus”, doi: 10.13140/RG.2.2.27715.14883.

[25] Y. Abgaz et al., “Dekomposisi Aplikasi Monolit Menjadi Arsitektur Layanan Mikro: Tinjauan Sistematis,” IEEE Transactions on Software Engineering, vol. 49, no. 8, hlm. 4213–4242, Agustus 2023, doi: 10.1109/TSE.2023.3287297.

[26] F. Tapia, M. ángel Mora, W. Fuertes, H. Aules, E. Flores, dan T. Toulkeridis, “Dari sistem monolitik ke layanan mikro: Studi perbandingan kinerja,” Applied Sciences (Swiss), vol. 10, no. 17, Sep. 2020, doi: 10.3390/app10175797.

[27] D. Kuryazov, D. Jabborov, dan B. Khujamuratov, “Menuju Dekomposisi Aplikasi Monolitik menjadi Layanan Mikro,” dalam Konferensi Internasional IEEE ke-14 tentang Aplikasi Teknologi Informasi dan Komunikasi, AICT 2020 - Prosiding, Institute of Electrical and Electronics Engineers Inc., Okt. 2020. doi: 10.1109/AICT50176.2020.9368571.

[28] F. Auer, V. Lenarduzzi, M. Felderer, dan D. Taibi, “Dari sistem monolitik ke Mikroservis: Kerangka penilaian,” Inf Softw Technol, vol. 137, Sep. 2021, doi: 10.1016/j.infsof.2021.106600.

[29] MH Hasan, M. Hafeez Osman, NI Admodisastro, dan S. Muhammad, “Dari Monolit ke Mikroservis: Mengukur Kemudahan Pemeliharaan Arsitektur,” 2023. [Online]. Tersedia di: www.ijacsa.thesai.org

[30] I. Oumoussa dan R. Saidi, “Evolusi Identifikasi Layanan Mikro dalam Dekomposisi Monolit: Tinjauan Sistematis,” IEEE Access, vol. 12, hlm. 23389–23405, 2024, doi: 10.1109/ACCESS.2024.3365079.

[31] M. Abdellatif dkk., “Taksonomi pendekatan identifikasi layanan untuk modernisasi sistem perangkat lunak lama,” Jurnal Sistem dan Perangkat Lunak, vol. 173, Mar. 2021, doi: 10.1016/j.jss.2020.110868.

[32] J. Li, H. Xu, X. Xu, dan Z. Wang, “Metode Baru untuk Mengidentifikasi Layanan Mikro dengan Mempertimbangkan Ekspektasi Kualitas dan Kendala Penyebaran,” https://doi.org/10.1142/S021819402250019X, vol. 32, no. 3, hlm. 417–437, Apr. 2022, doi: 10.1142/S021819402250019X.

[33] MH Gomes Barbosa dan PHM Maia, “Menuju Identifikasi Kandidat Microservice dari Aturan Bisnis yang Diimplementasikan dalam Prosedur Tersimpan,” dalam Prosiding - Konferensi Internasional IEEE 2020 tentang Arsitektur Perangkat Lunak, ICSA-C 2020, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, hlm. 41–48. doi: 10.1109/ICSA-C50368.2020.00015.

[34] O. Al-Debagy dan P. Martinek, “Metode Dekomposisi Layanan Mikro Melalui Penggunaan Representasi Terdistribusi Kode Sumber,” Scalable Computing, vol. 22, 1, hlm. 39–52, 2021, doi: 10.12694:/scpe.v22i1.1836.PAK.

[35] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, dan Q. Zheng, “Identifikasi Kandidat Layanan dari Sistem Monolitik Berdasarkan Jejak Eksekusi,” IEEE Transactions on Software Engineering, vol. 47, no. 5, hlm. 987–1007, Mei 2021, doi: 10.1109/TSE.2019.2910531.

[36] U. Desai, S. Bandyopadhyay, dan S. Tamilselvam, “Jaringan Neural Graf untuk Mengurangi Outlier untuk Refactoring Aplikasi Monolit,” Feb. 2021, [Online]. Tersedia: http://arxiv.org/abs/2102.03827

[37] AK Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, dan D. Banerjee, “Mono2Micro: Alat praktis dan efektif untuk menguraikan aplikasi Java monolitik menjadi layanan mikro,” dalam ESEC/FSE 2021 - Prosiding Pertemuan Bersama ACM ke-29 Konferensi dan Simposium Rekayasa Perangkat Lunak Eropa tentang Fondasi Rekayasa Perangkat Lunak, Association for Computing Machinery, Inc, Agustus 2021, hlm. 1214–1224. doi: 10.1145/3468264.3473915.

[38] BS Mitchell dan S. Mancoridis, “Tentang Modularisasi Otomatis dari Sistem Perangkat Lunak yang Menggunakan Alat Bunch.”

[39] P. Francisco, M. Gastón, dan A. Hernán, Migrasi dari arsitektur monolitik ke layanan mikro: Tinjauan Cepat. IEEE, 2019. doi: 10.1109/SCCC49216.2019.8966423.

[40] K. Justas dan M. Dalius, “Migrasi Perangkat Lunak Warisan ke Arsitektur Layanan Mikro,” Institut Insinyur Listrik dan Elektronika, 2019, hlm. 32. doi: 10.1109/eStream.2019.8732170.

[41] Z. Ren et al., “Migrasi aplikasi web dari struktur monolitik ke arsitektur microservices,” dalam ACM International Conference Proceeding Series, Association for Computing Machinery, Sep. 2018. doi: 10.1145/3275219.3275230.

[42] J. Fritzsch, J. Bogner, S. Wagner, dan A. Zimmermann, “Migrasi Layanan Mikro di Industri: Niat, Strategi, dan Tantangan,” dalam Prosiding - Konferensi Internasional IEEE 2019 tentang Pemeliharaan dan Evolusi Perangkat Lunak, ICSME 2019, Institute of Electrical and Electronics Engineers Inc., Sep. 2019, hlm. 481–490. doi: 10.1109/ICSME.2019.00081.

[43] M. Kalske, N. Mäkitalo, dan T. Mikkonen, “Tantangan Saat Beralih dari Arsitektur Monolit ke Arsitektur Mikroservis,” dalam Lecture Notes in Computer Science (termasuk subseri Lecture Notes in Artificial Intelligence dan Lecture Notes in Bioinformatics), Springer Verlag, 2018, hlm. 32–47. doi: 10.1007/978-3-319-74433-9_3.

[44] M. Vainio dan T. Antti-Pekka, “Manfaat dan tantangan dalam migrasi dari arsitektur monolitik ke arsitektur microservice,” 2021. [Online]. Tersedia di: http://www.cs.helsinki.fi/

[45] S. Eski dan F. Buzluca, “Pendekatan ekstraksi otomatis - Transisi ke arsitektur microservices dari aplikasi monolitik,” dalam ACM International Conference Proceeding Series, Association for Computing Machinery, 2018. doi: 10.1145/3234152.3234195.

[46] R. Su, X. Li, dan D. Taibi, “Dari Microservice ke Monolith: Tinjauan Literatur Multivokal †,” 01 April 2024, Multidisciplinary Digital Publishing Institute (MDPI). doi: 10.3390/electronics13081452.

[47] C. Bandara dan I. Perera, “Transforming monolithic systems to microservices - An analysis toolkit for legacy code evaluation,” dalam 20th International Conference on Advances in ICT for Emerging Regions, ICTer 2020 - Proceedings, Institute of Electrical and Electronics Engineers Inc., Nov. 2020, doi: 10.1109/ICTer51097.2020.9325443. hlm. 95–100.

[48] H. Michael Ayas, P. Leitner, dan R. Hebig, “Perjalanan Migrasi Menuju Layanan Mikro,” dalam Lecture Notes in Computer Science (termasuk subseri Lecture Notes in Artificial Intelligence dan Lecture Notes in Bioinformatics), Springer Science and Business Media Deutschland GmbH, 2021, hlm. 20–35. doi: 10.1007/978-3-030-91452-3_2.

[49] D. Taibi dan K. Systä, “Dari sistem monolitik ke layanan mikro: Kerangka kerja dekomposisi berdasarkan penambangan proses,” dalam CLOSER 2019 - Prosiding Konferensi Internasional ke-9 tentang Komputasi Awan dan Ilmu Layanan, SciTePress, 2019, hlm. 153–164. doi: 10.5220/0007755901530164.

[50] HHOS Da Silva, GF De Carneiro, dan MP Monteiro, “Menuju peta jalan untuk migrasi sistem perangkat lunak lama ke arsitektur berbasis layanan mikro,” dalam CLOSER 2019 - Prosiding Konferensi Internasional ke-9 tentang Komputasi Awan dan Ilmu Layanan, SciTePress, 2019, hlm. 37–47. doi: 10.5220/0007618400370047.

[51] M. El Kholy dan A. El Fatatry, “Kerangka Kerja untuk Interaksi antara Basis Data dan Arsitektur Layanan Mikro,” IT Prof, vol. 21, no. 5, hlm. 57–63, September 2019, doi: 10.1109/MITP.2018.2889268.

[52] F. Antunes, MJ Dias De Lima, M. Antônio, P. Araújo, D. Taibi, dan M. Kalinowski, “Meneliti Manfaat dan Keterbatasan Migrasi ke Arsitektur Micro-Frontends,” 2024.

[53] A. Maria dan C. Fulvio, “Menjelajahi Transisi Arsitektur Perangkat Lunak: Dari Aplikasi Monolitik ke Mikrofrontend yang ditingkatkan dengan pustaka Webpack dan Pengujian Cypress,” Juli 2024.

[54] J. Fritzsch, J. Bogner, A. Zimmermann, dan S. Wagner, “Dari Monolit ke Mikroservis: Klasifikasi Pendekatan Refactoring,” 2018. doi: arXiv:1807.10059.

[55] L. De Lauretis, “Dari arsitektur monolitik ke arsitektur layanan mikro,” dalam Prosiding - Simposium Internasional IEEE ke-30 Lokakarya Rekayasa Keandalan Perangkat Lunak 2019, ISSREW 2019, Institute of Electrical and Electronics Engineers Inc., Okt. 2019, hlm. 93–96. doi: 10.1109/ISSREW.2019.00050.

[56] L. Nunes, N. Santos, dan A. Rito Silva, “Dari arsitektur monolit ke arsitektur layanan mikro: Pendekatan berdasarkan konteks transaksional,” dalam Lecture Notes in Computer Science (termasuk subseri Lecture Notes in Artificial Intelligence dan Lecture Notes in Bioinformatics), 2019. doi: 10.1007/978-3-030-29983-5_3.

[57] N. Santos dan A. Rito Silva, “Metrik kompleksitas untuk migrasi arsitektur microservices,” dalam Prosiding - Konferensi Internasional IEEE ke-17 tentang Arsitektur Perangkat Lunak, ICSA 2020, Institute of Electrical and Electronics Engineers Inc., Mar. 2020, hlm. 169–178. doi: 10.1109/ICSA47634.2020.00024.

[58] SP Ma, TW Lu, dan CC Li, “Migrasi Monolit ke Mikroservis berdasarkan Analisis Permintaan Akses Basis Data,” dalam Prosiding - Konferensi Internasional IEEE ke-16 tentang Rekayasa Sistem Berorientasi Layanan, SOSE 2022, Institute of Electrical and Electronics Engineers Inc., 2022, hlm. 11–18. doi: 10.1109/SOSE55356.2022.00008.

[59] J. Ghofrani dan A. Bozorgmehr, “Migrasi ke Layanan Mikro: Hambatan dan Solusi,” 2019, hlm. 269–281. doi: 10.1007/978-3-030-32475-9_20.

[60] J. Ghofrani dan D. Lübke, “Tantangan Arsitektur Layanan Mikro: Survei tentang Keadaan Praktik,” 2018. [Online]. Tersedia: http://ceur-ws.org/Vol-2072

[61] A. Razzaq dan SAK Ghayyur, “Studi pemetaan sistematis: Era baru arsitektur perangkat lunak dari arsitektur monolitik ke arsitektur microservice—kesadaran dan tantangan,” 01 Maret 2023, John Wiley and Sons Inc. doi: 10.1002/cae.22586.