Tugas Makalah Makassar, 25 September 2025

MAKALAH

ASPECT-ORIENTED SOFTWARE DEVELOPMENT (AOSD)

Dosen Pengampu: Mardiyyah Hasnawi, S.Kom., M.T., MTA.

Disusun oleh:

KELOMPOK III

Alexandria Kayla Kazya Putri Nur (13020230001)

Nurvana Syakir (13020230026)

Nabilah Tikah Mushlihah Thahir (13020230045)

Nazwa Syalaisa Haq (13020230063)

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS MUSLIM INDONESIA

MAKASSAR

2025

DAFTAR ISI

BAB I
PENDAHULUAN

A. Latar Belakang

Di era digital yang terus berkembang pesat, pengembangan perangkat lunak menghadapi tantangan yang semakin kompleks, terutama dalam memenuhi kebutuhan akan sistem yang efisien, mudah dipelihara, dan mampu beradaptasi dengan perubahan skala. Pendekatan tradisional seperti Object-Oriented Programming (OOP) telah menjadi landasan utama sejak tahun 1970-an hingga 1990-an, memberikan struktur berbasis objek yang mempermudah pengelolaan kode. Namun, OOP memiliki keterbatasan signifikan dalam menangani crosscutting concerns, seperti logging, keamanan, transaksi, dan penanganan kesalahan. Masalah ini sering kali menghasilkan scattering—di mana kode terkait kebutuhan lintas tersebar di berbagai modul—serta tangling—di mana logika inti bercampur dengan fungsi pendukung, sehingga menyulitkan pemeliharaan dan pengembangan lebih lanjut, terutama pada sistem berskala besar.

Untuk mengatasi keterbatasan tersebut, Aspect-Oriented Software Development (AOSD) muncul sebagai paradigma inovatif yang diperkenalkan oleh Gregor Kiczales dan timnya pada tahun 1997 melalui konsep Aspect-Oriented Programming (AOP). AOSD memperkenalkan aspect sebagai unit modularisasi baru yang memungkinkan pemisahan crosscutting concerns dari kode utama, sehingga meningkatkan modularitas, reusability, dan kemudahan pemeliharaan perangkat lunak. Seiring waktu, konsep ini berkembang menjadi AOSD pada awal 2000-an, mencakup seluruh siklus hidup pengembangan perangkat lunak, mulai dari analisis kebutuhan hingga evaluasi performa. Penerapan AOSD telah menunjukkan keberhasilannya dalam berbagai domain industri, seperti enterprise systems, e-health, telekomunikasi, dan multimedia, di mana pengelolaan kompleksitas menjadi kunci keberhasilan. Oleh karena itu, kajian mengenai AOSD tidak hanya bersifat teoritis tetapi juga memiliki implikasi praktis yang signifikan dalam mendukung inovasi teknologi perangkat lunak di era modern.

B. Tujuan Penulisan

Makalah ini memiliki tujuan sebagai berikut:

- Menguraikan definisi dan sejarah perkembangan *Aspect-Oriented Software Development* (AOSD) sejak diperkenalkan melalui *Aspect-Oriented Programming* (AOP).
- Menjelaskan fase-fase penerapan AOSD, yang mencakup analisis, desain, dan implementasi, beserta contoh aplikasinya dalam sistem nyata.
- Mengidentifikasi karakteristik utama AOSD yang membedakannya dari metodologi pengembangan perangkat lunak lainnya.
- Menganalisis kelebihan dan kelemahan AOSD berdasarkan hasil penelitian dan penerapannya dalam industri.
- Melakukan perbandingan antara AOSD dan metodologi pengembangan perangkat lunak lainnya, khususnya Waterfall, Agile Unified Process, Component-Based Development, dan Rational Unified Process.
- Menguraikan alat bantu (tools) yang mendukung penerapan AOSD, mulai dari tahap analisis kebutuhan hingga implementasi.

BAB II
KAJIAN TEORITIS

A. Definisi dan Sejarah

Aspect-Oriented Software Development (AOSD) adalah paradigma pengembangan perangkat lunak yang muncul untuk mengatasi keterbatasan pendekatan Object-Oriented Programming (OOP). Dalam OOP, kebutuhan lintas modul atau crosscutting concerns seperti logging, keamanan, dan penanganan kesalahan sering menimbulkan masalah scattering (kode tersebar di banyak bagian) dan tangling (kode bercampur dengan logika inti). AOSD memperkenalkan aspect sebagai unit modularisasi baru yang memungkinkan pemisahan crosscutting concerns dari kode utama, sehingga perangkat lunak menjadi lebih modular, mudah dipelihara, dan reusable[1].

Sejarah AOSD berawal dari dominasi Structured Programming dan Object-Oriented Programming (OOP) pada tahun 1970 hingga 1990-an yang meskipun efektif dalam membangun sistem berbasis objek, tetap belum mampu menangani kompleksitas crosscutting concerns. Untuk menjawab permasalahan ini, Gregor Kiczales dan timnya memperkenalkan Aspect-Oriented Programming (AOP) pada tahun 1997 dengan menghadirkan aspect sebagai unit modularisasi baru. Seiring perkembangan kebutuhan perangkat lunak berskala besar, konsep AOP kemudian diperluas menjadi Aspect-Oriented Software Development (AOSD) pada awal 2000-an. AOSD tidak hanya berfokus pada tahap implementasi, tetapi juga mencakup seluruh siklus hidup pengembangan perangkat lunak, mulai dari analisis kebutuhan, desain, implementasi, pengujian, hingga evaluasi performa[2].

B. Penerapan Model

Aspect Oriented Software Development (AOSD) diterapkan melalui tiga fase utama, yaitu analisis, desain, dan implementasi.

- **Analisis**

  Tahap ini berfokus pada pengidentifikasian kebutuhan sistem sekaligus pemetaan *core concerns* (fungsi inti) dan *intervention concerns* (kebutuhan lintas). Pada fase ini, dilakukan kajian terhadap *use case* untuk menemukan bagian-bagian sistem yang berpotensi menimbulkan penyebaran kode. Dengan cara ini, intervention concerns dapat dikenali sejak awal sehingga pengembang memiliki gambaran lebih jelas dalam memodularisasi sistem pada tahap berikutnya[1].

- **Desain**

Tahap ini lebih pada pemodelan sistem dengan pendekatan berorientasi aspek. Desain dilakukan dengan menentukan kelas-kelas pengendali, hubungan antar komponen, serta titik eksekusi (*join points*) tempat aspek akan diterapkan. Proses ini bertujuan untuk memisahkan kebutuhan inti dari kebutuhan lintas, sehingga kode yang sebelumnya berpotensi tersebar bisa diarahkan menjadi modul aspek yang terstruktur[1].

- **Implementasi**

  Tahap ini merupakan realisasi dari desain yang telah dibuat. Kebutuhan inti dikembangkan dalam bentuk komponen utama, sementara intervention concerns direalisasikan sebagai aspek yang berdiri sendiri. Melalui proses *weaving*, aspek-aspek tersebut digabungkan kembali ke dalam sistem sehingga fungsi lintas dapat berjalan tanpa menambah kompleksitas kode inti[1]

Setiap fase tersebut memiliki peran yang saling melengkapi, mulai dari mengenali *intervention concerns*, merancang pemisahan logika, hingga mengimplementasikan modularisasi aspek. Sebagai contoh, dalam pengembangan **Sistem Manajemen Perpustakaan**, fase analisis digunakan untuk menelusuri kebutuhan inti seperti peminjaman, pengembalian, dan perpanjangan buku serta menemukan *intervention concerns* berupa *CheckingBook* dan *BookUpdating*. Pada fase desain, dirancang kelas pengendali Book_Trust_Controller yang mengoordinasikan interaksi antar kelas, sementara aspek lintas dipisahkan ke dalam modul terpisah. Selanjutnya, pada fase implementasi, aspek-aspek tersebut dimodularisasi dan digabungkan melalui proses *weaving*, sehingga sistem perpustakaan menjadi lebih modular, kode yang semula tersebar dapat dipusatkan, dan pemeliharaan sistem lebih mudah dilakukan[1].

C. Karakteristik Model

Rashid *et al.* dalam artikelnya *Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe* menjelaskan bahwa *Aspect-Oriented Software Development* (AOSD) memiliki sejumlah karakteristik utama yang membedakannya dari metodologi pengembangan perangkat lunak lainnya :

- **Modularisasi Cross-cutting Concerns**

  AOSD digunakan untuk memodularisasi concern lintas(cross-cutting concerns) seperti logging, keamanan, transaksi, dan monitoring, sehingga concern tersebut dipisahkan dari logika inti sistem[3].

- **Penerapan Secara Inkremental**

Dalam praktiknya, AOSD biasanya diperkenalkan secara bertahap. Organisasi cenderung menerapkannya terlebih dahulu pada concern non-inti atau fitur pendukung sebelum menggunakannya pada logika inti[3].

- **Skala Sistem yang Diterapkan**

  Studi menunjukkan bahwa AOSD diterapkan pada sistem berskala menengah hingga besar diberbagai domain, termasuk enterprise systems, e-health, telekomunikasi, workflow, web-based systems, dan multimedia[3]

Karakteristik tersebut menunjukkan bahwa AOSD berperan penting dalam memisahkan concern lintas, memudahkan pengelolaan sistem yang kompleks, serta dapat diadopsi secara realistis dalam proyek industry berskala besar[3].

D. Kelebihan dan Kelemahan

Rashid *et al.* juga menguraikan sejumlah kelebihan dan kelemahan AOSD berdasarkan hasil penelitian dan pengalaman industri :

- **Kelebihan**

  1. **Pemisahan Cross-cutting Concerns**

     Dengan AOSD, concern lintas dapat dipisahkan dari kode utama. Hal ini menghasilkan sistem yang lebih bersih, modular, dan fokus pada logika inti[3].

  2. **Meningkatkan Maintainability dan Reusability**

     Karena concern lintas dikemas dalam *aspects*, perubahan dapat dilakukan di satu tempat saja tanpa harus menyentuh banyak modul. Aspek juga dapat digunakan kembali di proyek lain dengan kebutuhan serupa[3].

  3. **Stabilitas Desain dalam Evolusi Sistem**

     Saat perangkat lunak berkembang, AOSD membantu menjaga stabilitas desain dengan memungkinkan perubahan dilakukan pada aspek tertentu tanpa merombak keseluruhan sistem[3].

  4. **Relevansi Industri pada Skala Besar**

     Penerapan AOSD yang sukses pada sistem menengah hingga besar di berbagai domain menunjukkan bahwa model ini bukan sekadar teori, tetapi memiliki nilai praktis yang tinggi[3].

- **Kelemahan**

  1. **Pointcut Fragility**

     Definisi pointcut yang menghubungkan aspek dengan kode inti bersifat rapuh. Perubahan kecil pada kode inti dapat berdampak luas terhadap aspek, sehingga menimbulkan risiko bug[3].

2. **Kesulitan Debugging**

   Alur eksekusi yang tersembunyi melalui *join points* dan *advice* membuat proses pelacakan kesalahan menjadi lebih kompleks dibandingkan dengan metode tradisional[3].

3. **Interaksi Antar-Aspek**

   Saat beberapa aspek bekerja bersamaan, interaksi yang tidak terduga dapat muncul, menimbulkan efek samping yang sulit diprediksi. Hal ini membutuhkan strategi pengelolaan aspek yang lebih hati-hati[3].

Dengan demikian, AOSD menawarkan keunggulan signifikan dalam modularitas, maintainability, dan kemampuan mengelola sistem kompleks. Namun, kelemahannya dalam hal fragilitas pointcut, kompleksitas debugging, dan potensi interaksi antar-aspek harus dipahami dan diantisipasi agar penerapannya dapat berjalan efektif dalam skala industry[3].

E. Perbedaan dengan Model atau Metodologi Lainnya

- **Perbandingan Aspect Oriented Software Development (AOSD) dan Waterfall Model**

  Menurut Model Waterfall, yang merupakan metodologi untuk mengembangkan proyek secara linear dan kaku, setiap fase harus diselesaikan terlebih dahulu sebelum fase berikutnya dimulai. Model ini tidak fleksibel terhadap perubahan, melakukan analisis di bagian akhir, dan cocok untuk proyek dengan persyaratan yang stabil[4].

  Sebaliknya, Aspect-Oriented Software Development (AOSD) adalah sebuah paradigma pengembangan yang berfokus pada analisis dan modularisasi isu-isu lintas (seperti *logging* dan keamanan) ke dalam aspek. AOSD dapat digabungkan dengan metodologi lain, sehingga menawarkan fleksibilitas dan modularitas yang lebih besar untuk evolusi perangkat lunak. Secara sederhana, AOSD dengan pengembangan yang modular dan dapat beradaptasi lebih selaras dengan metodologi Waterfall serta lebih mandiri dengan teknik pengembangan iteratif yang fleksibel[5].

- **Perbandingan Aspect Oriented Software Development (AOSD) dan Agile Unified Process**

  Agile Unified Process (AUP) adalah metodologi Agile-UP yang menekankan pada iterasi, pembuatan rancangan, kerja tim, serta adaptasi terhadap perubahan [6].

Sebaliknya, Aspect-Oriented Software Development (AOSD) adalah sebuah paradigma untuk mengintegrasikan isu-isu lintas (seperti *logging* dan keamanan) ke dalam aspek-aspek modular. Meskipun terdapat perbedaan, AOSD dapat memperkuat AUP dengan meningkatkan modularitas dan kualitas kode. Melalui integrasi AOSD, isu-isu lintas dapat ditangani secara langsung dalam iterasi AUP, sehingga membantu menjaga desain tetap bersih, mengurangi kompleksitas, dan meningkatkan kemampuan AUP dalam menangani evolusi perangkat lunak[5].

- **Perbandingan Aspect Oriented Software Development (AOSD) dan Compnent-based Development**

  Component-Based Software Development (CBSD) memecah fungsionalitas ini ke dalam komponen-komponen yang dapat digunakan kembali, tetapi juga menangani isu-isu lintas (seperti *logging* dan keamanan) yang ada dan tercermin dalam kode[7].

  Aspect-Oriented Software Development (AOSD) mengatasi masalah ini dengan memecah isu-isu lintas ke dalam aspek-aspek terpisah. Menurut penelitian ini, AOSD dapat diintegrasikan ke dalam CBSD—misalnya melalui model *Fractal Aspect Component (FAC)*—untuk menjangkau concern yang lebih komprehensif, memungkinkan pengembangan yang lebih menyeluruh dari perspektif fungsional maupun lintas, serta meningkatkan ketahanan dan evolusi perangkat lunak[5].

- **Perbandingan Aspect Oriented Software Development (AOSD) dan Rational Unified Proses**

  Rational Unified Process (RUP) adalah sebuah kerangka kerja untuk mengembangkan perangkat lunak secara iteratif yang berbasis pada arsitektur. RUP disusun dalam beberapa fase (*Inception, Elaboration, Construction, Transition*) dan dipandu oleh *use case*, Meskipun demikian, RUP tidak secara khusus menangani isu-isu lintas secara modular[8].

  Aspect-Oriented Software Development (AOSD) adalah sebuah paradigma yang fleksibel yang mengintegrasikan isu-isu lintas (seperti *logging* dan keamanan) ke dalam aspek-aspek tertentu. Artikel ini menggambarkan Aspect-Oriented Software Process (AOSP) sebagai integrasi AOSD ke dalam RUP, dengan menyoroti aktivitas-aktivitas berorientasi aspek pada setiap fase RUP. Integrasi ini memungkinkan RUP untuk menangani isu-isu kompleksitas lintas, sehingga menghasilkan desain dan kode yang lebih tangguh, mudah dipahami, serta dapat berevolusi[5].

F. Alat Bantu Model atau Metode

Dalam penerapan *Aspect-Oriented Software Development (AOSD)*, terdapat sejumlah alat bantu (*tools*) dan aplikasi pendukung yang digunakan untuk membantu proses pengembangan perangkat lunak. Alat bantu ini digunakan sejak tahap kebutuhan, desain, hingga implementasi, sehingga memudahkan pemisahan *cross-cutting concerns*

Beberapa alat bantu yang digunakan antara lain :

- **WEAVR**
  Sebuah model weaving tool yang digunakan pada proyek Motorola untuk modularisasi tracing, debugging, dan penanganan timeout.WEAVR memungkinkan pengembang melakukan weaving aspek langsung pada model UML sebelum menghasilkan kode.
- **EA-Miner**
  Aplikasi yang memanfaatkan Teknik Natural Language Processing(NLP) untuk menambang dokumen kebutuhan perangkat lunak.EA-Miner membantu menemukan cross-cutting concerns sejak tahap requirement sehingga aspek bisa ditangani lebih awal.
- **Theme/UML**
  Alat bantu pemodelan berbasis UML yang digunakan untuk mendesain aspek sejak tahap perancangan. Dengan Theme/UML, pengembang dapat memvisualisasikan concern lintas secara eksplisit di dalam diagram desain

Dengan adanya alat bantu tersebut, AOSD dapat diimplementasikan lebih sistematis, mulai dari analisis kebutuhan (EA-Miner), desain (Theme/UML), hingga implementasi berbasis model (WEAVR). Hal ini menunjukkan bahwa AOSD telah memiliki dukungan teknologi nyata dalam siklus pengembangan perangkat lunak[3].

BAB III
PENUTUP

A. Kesimpulan

Berdasarkan pembahasan dalam makalah ini, *Aspect-Oriented Software Development* (AOSD) hadir sebagai solusi untuk mengatasi keterbatasan *Object-Oriented Programming* (OOP) dalam menangani kebutuhan lintas (*crosscutting concerns*), seperti logging, keamanan, dan penanganan kesalahan. Sejak diperkenalkan oleh Gregor Kiczales pada tahun 1997 melalui *Aspect-Oriented Programming* (AOP) dan berkembang menjadi AOSD pada awal 2000-an, paradigma ini menawarkan cara kerja baru dengan memisahkan kebutuhan lintas ke dalam *aspect*. Proses penerapannya meliputi analisis, desain, hingga implementasi, dengan dukungan alat bantu seperti WEAVR, EA-Miner, dan Theme/UML yang memudahkan pengelolaan sistem berskala menengah hingga besar, misalnya pada enterprise systems, e-health, dan telekomunikasi.

Karakteristik utama AOSD, seperti pemisahan kebutuhan lintas dan penerapan secara bertahap, memberi keunggulan dalam hal kemudahan pemeliharaan, penggunaan ulang kode, dan menjaga stabilitas desain saat sistem berkembang. Meski begitu, ada juga tantangan seperti kerentanan *pointcut*, kesulitan dalam proses debugging, dan kemungkinan terjadinya interaksi antar-aspek yang tidak terduga. Jika dibandingkan dengan metode lain seperti Waterfall, AUP, CBD, dan RUP, AOSD lebih fleksibel dan lebih mampu menangani kompleksitas, terutama bila dipadukan dengan pendekatan pengembangan iteratif. Dengan demikian, AOSD bukan hanya konsep teoritis, tetapi juga strategi penting dalam menghadapi tantangan pengembangan perangkat lunak modern.

DAFTAR PUSTAKA

[1] F. SoleimanianGharehchopogh, E. Amini, and B. Zebardast, "Aspect-Oriented Software Development based Solution for Intervention Concerns Problems: Case Study," *Int. J. Comput. Appl.*, vol. 63, no. 4, pp. 16–25, 2013, doi: 10.5120/10453-5157.

[2] M. K. Abid and Mohibullah Khan, "Complexity in the adaptation of aspect-oriented software Development," *Int. J. Inf. Syst. Comput. Technol.*, vol. 1, no. 1, pp. 13–20, 2022, doi: 10.58325/ijisct.001.01.0013.

[3] A. Rashid *et al.*, "Aspect-oriented software development in practice: Tales from AOSD-Europe," *Computer (Long. Beach. Calif).*, vol. 43, no. 2, pp. 19–26, 2010, doi: 10.1109/MC.2010.30.

[4] Shamsulhuda Khan and Shubhangi Mahadik, "A Study on Fintech Develop in India," *Int. J. Adv. Res. Sci. Commun. Technol.*, no. July 2022, pp. 399–402, 2022, doi: 10.48175/ijarsct-5696.

[5] M. Khaari and R. Ramsin, "Process patterns for aspect-oriented software development," *17th IEEE Int. Conf. Work. Eng. Comput. Syst. ECBS 2010*, no. January 2010, pp. 241–250, 2010, doi: 10.1109/ECBS.2010.33.

[6] M. Z. Than, "An Analysis on Agile Unified Process (AUP) Framework," no. April, 2022, doi: 10.13140/RG.2.2.20260.55682.

[7] N. Pessemier, L. Seinturier, L. Duchien, and T. Coupaye, "A component-based and aspect-oriented model for software evolution," *Int. J. Comput. Appl. Technol.*, vol. 31, no. 1–2, pp. 94–105, 2008, doi: 10.1504/IJCAT.2008.017722.

[8] A. Anwar, "A Review of RUP (Rational Unified Process)," *Int. J. Softw. Eng.*, vol. 5, no. 2, pp. 8–24, 2014, [Online]. Available: http://www.cscjournals.org/library/manuscriptinfo.php?mc=IJSE-142