OO Metrics Proposed by Lorenz and Kidd

DISUSUN OLEH:

(H071221008)	NURHALIZA ALAWIAH SYAH
(H071221038)	SURYA AGUS NANRO
(H071221071)	SALDAN RAMA

PROGRAM STUDI SISTEM INFORMASI
DEPARTEMEN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS HASANUDDIN
2024

Object-Oriented Metrics

Object-Oriented (OO) metrics yang diusulkan oleh Mark Lorenz dan Jeff Kidd adalah pendekatan pengukuran untuk mengevaluasi manajemen dan kualitas desain perangkat lunak berbasis objek. Metrik ini terbagi menjadi dua kategori utama:

1. Project Metrics digunakan untuk aspek manajerial, seperti mengukur ukuran aplikasi (melalui jumlah kelas utama dan subsistem) dan estimasi kebutuhan sumber daya manusia, misalnya menggunakan metrik Person-Days Per Class untuk menghitung waktu yang dibutuhkan untuk setiap kelas, serta Classes Per Developer untuk memperkirakan kapasitas kerja pengembang.
2. Design Metrics berfokus pada menilai kompleksitas dan kualitas desain perangkat lunak. Ini mencakup ukuran metode seperti Lines of Code (LOC) dan Message Sends, ukuran kelas (jumlah atribut dan metode), serta kompleksitas hierarki pewarisan dan internal metode.

Setiap metrik dilengkapi dengan panduan berupa deskripsi, ambang batas nilai, dan tindakan yang disarankan untuk mengatasi anomali. Pendekatan ini dirancang berdasarkan pengalaman langsung mereka dalam proyek berbasis Smalltalk dan C++, sehingga memberikan solusi praktis untuk pengembangan perangkat lunak berbasis objek yang efisien dan terukur.

Cara Kerja OO Metrics

1. Kategori Metrics

Lorenz dan Kidd membagi metrik OO menjadi beberapa kategori utama:

- Internal Metrics: Metrik ini berfokus pada karakteristik internal suatu kelas, mengevaluasi kompleksitas dan struktur dalam. Contoh yang paling sering digunakan adalah:
    - Weighted Methods per Class (WMC): Mengukur jumlah metode dalam sebuah kelas, dengan memberikan bobot pada kompleksitas setiap metode. Semakin tinggi WMC, semakin kompleks dan sulit dipelihara kelas tersebut.
- External Metrics: Metrik eksternal mengevaluasi hubungan antar kelas dalam sistem. Interaksi yang terlalu erat antara kelas-kelas dapat mengurangi modularitas dan meningkatkan ketergantungan. Salah satu contohnya adalah:
    - Coupling Between Objects (CBO): Mengukur jumlah hubungan antara satu kelas dengan kelas lain. Nilai CBO yang tinggi menunjukkan ketergantungan antar kelas yang tinggi, sehingga sulit untuk diuji dan diperbaiki.
- Inheritance Metrics: Metrik pewarisan mengevaluasi penggunaan hierarki pewarisan dalam desain perangkat lunak. Contohnya:

    - Depth of Inheritance Tree (DIT): Mengukur kedalaman sebuah kelas dalam pohon pewarisan. Meskipun pewarisan memperkenalkan kembali penggunaan kode, kedalaman yang terlalu dalam dapat mempersulit pemahaman desain.
- Size Metrics: Ukuran sistem dihitung untuk memberikan gambaran tentang seberapa besar atau kompleks sebuah aplikasi. Metrik ini termasuk:
    - Jumlah Kelas Utama, Subsistem, dan Kelas Pendukung: Memberikan estimasi tingkat pekerjaan dan staf yang diperlukan untuk menyelesaikan proyek.

2. Formula Metrik

Setiap metrik memiliki rumus tertentu untuk menghitung nilai-nilai yang relevan. Misalnya:

- WMC dihitung dengan menjumlahkan bobot dari semua metode dalam sebuah kelas.
- DIT dihitung dengan menghitung jumlah level dari kelas dasar hingga kelas turunan.
- CBO dihitung dengan menghitung jumlah kelas lain yang digunakan oleh kelas tertentu.

3. Atribut Metrik

Setiap metrik yang dijelaskan oleh Lorenz dan Kidd memiliki tujuh atribut yang dirancang untuk memberikan wawasan yang lengkap:

- Name (nama)
- Meaning (arti)
- Project results (hasil proyek) – memberikan representasi statistik yang membantu visualisasi dan penggunaan metrik.
- Affecting Factors (faktor yang memengaruhi) – mencakup faktor-faktor yang mungkin memengaruhi nilai metrik.
- Related Metrics (metrik terkait) – hubungan antara metrik yang saling memengaruhi atau terkait dalam analisis.
- Thresholds (batasan nilai) – berdasarkan pengalaman, penulis memberikan rentang nilai yang dapat dianggap wajar atau tidak diinginkan.
- Suggested Actions (tindakan yang disarankan) – tindakan yang disarankan jika metrik berada di luar ambang batas yang diinginkan.

Implementasi OO Metrics

Langkah-langkah Implementasi

1. **Pemilihan Metrik**

   Langkah pertama dalam implementasi adalah menentukan metrik mana yang paling relevan untuk digunakan berdasarkan tujuan evaluasi sistem perangkat lunak. Tujuan ini dapat beragam, seperti meningkatkan maintainability (kemudahan pemeliharaan), mengurangi kompleksitas desain, meningkatkan modularitas, atau memastikan kualitas kode.

   - Contoh Pertimbangan

     Jika fokus utama adalah memahami kompleksitas suatu kelas, maka metrik seperti *Weighted Methods per Class (WMC)* sangat relevan. Namun, jika tujuannya adalah mengevaluasi hubungan antar kelas, *Coupling Between Objects (CBO)* menjadi pilihan yang tepat.

   - Pendekatan Sistematis

     Sebelum memilih metrik, perlu dilakukan pemetaan kebutuhan terhadap atribut kualitas perangkat lunak, seperti efisiensi, keterbacaan kode, atau fleksibilitas desain. Setelah itu, pilihlah metrik yang dapat memberikan wawasan mendalam untuk masing-masing atribut tersebut.

   - Dokumentasi

     Setiap metrik yang dipilih perlu dicatat lengkap dengan alasan pemilihannya untuk memudahkan komunikasi dalam tim pengembang.

2. **Pengumpulan Data**

   Setelah metrik ditentukan, langkah berikutnya adalah mengumpulkan data dari sistem perangkat lunak yang akan dianalisis. Data ini meliputi informasi mengenai struktur kode, hierarki kelas, dan hubungan antar objek. Proses ini melibatkan:

   - **Penggunaan Alat Analisis Statis**

     Alat seperti SonarQube, JDepend, atau NDepend dapat digunakan untuk mengekstrak informasi penting dari kode sumber secara otomatis. Alat ini juga mampu memberikan laporan yang detail terkait struktur kelas, metode, dan ketergantungan antar objek.

   - **Pengumpulan Manual (Opsional)**

     Dalam beberapa kasus, analisis manual mungkin diperlukan untuk memastikan data yang diperoleh sesuai dengan kebutuhan metrik, terutama jika sistem tidak memiliki dokumentasi yang baik.

   - **Validasi Data**

Pastikan data yang dikumpulkan lengkap dan akurat. Misalnya, jika Anda menghitung jumlah metode untuk WMC, pastikan semua metode, termasuk metode bawaan dari kelas dasar, sudah dihitung.

3. **Perhitungan Metrik**

   Setelah data tersedia, langkah selanjutnya adalah menerapkan rumus untuk setiap metrik yang telah dipilih. Proses ini melibatkan:

   - **Penerapan Formula**

     Gunakan formula spesifik untuk setiap metrik. Misalnya:

     - Untuk WMC, jumlahkan kompleksitas semua metode dalam suatu kelas.
     - Untuk DIT, hitung jumlah level dalam hierarki pewarisan dari akar hingga kelas target.
     - Untuk CBO, hitung jumlah kelas lain yang memiliki hubungan langsung dengan kelas yang dianalisis.

   - **Penggunaan Alat Otomasi**

     Sebagian besar alat analisis metrik sudah dilengkapi dengan kemampuan menghitung metrik secara otomatis. Namun, tetap perlu diverifikasi bahwa alat tersebut menggunakan formula yang sesuai dengan standar yang digunakan dalam proyek.

   - **Pendokumentasian Hasil**

     Catat hasil perhitungan metrik secara rinci untuk setiap kelas atau komponen yang dianalisis. Dokumentasi ini akan menjadi dasar untuk langkah berikutnya, yaitu analisis dan interpretasi.

4. **Analisis dan Interpretasi**

   Tahap akhir adalah menganalisis hasil metrik yang telah dihitung untuk memahami kondisi kualitas perangkat lunak. Analisis ini mencakup:

   - **Perbandingan dengan Ambang Batas**

     Setiap metrik biasanya memiliki ambang batas tertentu yang dianggap wajar. Misalnya:

     - WMC dengan nilai tinggi (>20) mungkin menunjukkan bahwa kelas terlalu kompleks dan memerlukan refactoring.
     - DIT yang terlalu dalam (>5) dapat menandakan hierarki yang sulit dipahami dan memerlukan penyederhanaan.

    - CBO dengan nilai tinggi menunjukkan ketergantungan antar kelas yang perlu dikurangi.
- **Identifikasi Area Masalah**

  Nilai metrik yang melebihi ambang batas adalah indikasi adanya masalah dalam desain perangkat lunak. Identifikasi ini memungkinkan tim untuk fokus pada area yang memerlukan perbaikan.

- **Rencana Perbaikan**

  Berdasarkan hasil analisis, buat rencana perbaikan yang terarah. Misalnya:

    - Untuk kelas dengan WMC tinggi, tim dapat memecah metode menjadi lebih kecil atau mengurangi tanggung jawab kelas.
    - Untuk sistem dengan CBO tinggi, desain ulang ketergantungan antar kelas dapat dilakukan untuk meningkatkan modularitas.
    - Untuk hierarki dengan DIT tinggi, tim dapat menyederhanakan pewarisan untuk meningkatkan keterbacaan dan pemahaman kode.

Alat untuk Implementasi

Beberapa alat yang dapat digunakan untuk menghitung OO metrics antara lain:

- SonarQube: Alat analisis statis yang dapat mengukur berbagai metrik kualitas perangkat lunak.
- JDepend: Alat untuk menganalisis dependensi antar paket Java.
- NDepend: Alat untuk menganalisis kualitas kode .NET.

Contoh Implementasi pada Proyek Nyata

Lorenz dan Kidd, dalam penelitian mereka, mengaplikasikan metrik perangkat lunak berbasis objek pada proyek nyata yang menggunakan bahasa pemrograman Smalltalk dan C++. Proyek ini memberikan data empiris yang berguna untuk mengevaluasi efektivitas metrik dalam mendukung pengembangan perangkat lunak berbasis objek. Implementasi ini mencakup beberapa aspek penting, mulai dari pengukuran ukuran aplikasi hingga estimasi kebutuhan sumber daya manusia.

1. Pengukuran Ukuran Aplikasi

Salah satu tujuan utama dari penerapan metrik adalah untuk menghitung ukuran aplikasi. Lorenz dan Kidd menggunakan beberapa metrik untuk mendapatkan gambaran yang komprehensif tentang ukuran dan kompleksitas sistem, antara lain:

- **Jumlah Kelas Utama**: Mengukur jumlah kelas yang menjadi inti atau komponen utama dalam aplikasi. Kelas utama ini biasanya bertanggung jawab atas fungsi-fungsi utama dari perangkat lunak.
- **Jumlah Subsistem**: Subsistem adalah kelompok kelas yang bekerja sama untuk menyelesaikan tugas tertentu dalam aplikasi. Dengan mengukur jumlah subsistem, pengembang dapat memahami tingkat modularitas sistem.
- **Jumlah Kelas Pendukung**: Kelas pendukung adalah kelas yang melayani fungsi tambahan untuk mendukung operasi kelas utama. Pengukuran ini memberikan wawasan tentang bagian-bagian pendukung yang mungkin membutuhkan optimisasi atau pengurangan beban kerja.

Pengukuran ini tidak hanya memberikan informasi tentang skala aplikasi, tetapi juga membantu manajemen dalam memahami beban kerja yang diperlukan untuk pengembangan dan pemeliharaan.

2. Estimasi Kebutuhan Sumber Daya

Selain pengukuran ukuran aplikasi, Lorenz dan Kidd juga memanfaatkan metrik untuk memperkirakan kebutuhan sumber daya manusia. Dua metrik utama yang digunakan adalah:

1. **Person-Days Per Class**: Metrik ini mengukur jumlah hari kerja yang diperlukan untuk menyelesaikan pengembangan atau pemeliharaan satu kelas. Dengan menghitung total jumlah kelas dalam aplikasi dan mengalikan dengan nilai ini, manajemen dapat memperkirakan jumlah waktu yang dibutuhkan untuk menyelesaikan proyek.
2. **Classes Per Developer**: Metrik ini memberikan estimasi jumlah kelas yang dapat ditangani oleh satu pengembang dalam waktu tertentu. Dengan membandingkan jumlah kelas dalam sistem dengan jumlah pengembang yang tersedia, manajemen dapat memastikan bahwa distribusi beban kerja adil dan efisien.

Kedua metrik ini memberikan wawasan yang sangat berguna untuk perencanaan staf dan alokasi sumber daya. Dengan memperkirakan kebutuhan staf secara akurat, manajemen dapat menghindari risiko keterlambatan proyek akibat kekurangan tenaga kerja atau kelebihan beban pada tim tertentu.

3. Manfaat yang Dihasilkan

Implementasi metrik ini memberikan manfaat nyata dalam pengelolaan proyek perangkat lunak, di antaranya:

- **Pengelolaan Beban Kerja yang Lebih Baik**: Dengan mengetahui jumlah kelas, subsistem, dan kelas pendukung, manajemen dapat merencanakan beban kerja dengan lebih baik, memastikan tidak ada bagian proyek yang terabaikan.
- **Peningkatan Akurasi Estimasi**: Dengan metrik seperti Person-Days Per Class, tim pengembang dapat memberikan estimasi waktu yang lebih realistis, mengurangi risiko terjadinya overrun waktu dan biaya.

- **Pemahaman Lebih Mendalam tentang Kompleksitas Sistem**: Dengan memahami jumlah subsistem dan tingkat modularitas, pengembang dapat mengevaluasi dan meningkatkan desain sistem untuk mencapai efisiensi yang lebih tinggi.
- **Perbaikan Proses Pengembangan**: Hasil analisis dari metrik ini dapat digunakan sebagai masukan untuk menyempurnakan proses pengembangan, seperti optimisasi hierarki pewarisan, pengurangan kompleksitas metode, atau peningkatan modularitas.

4. Aplikasi dalam Proyek Lain

Contoh yang diberikan oleh Lorenz dan Kidd menunjukkan bagaimana metrik ini dapat diterapkan tidak hanya dalam proyek Smalltalk dan C++, tetapi juga dalam proyek lain yang menggunakan paradigma berorientasi objek. Dengan alat analisis modern seperti SonarQube, JDepend, atau NDepend, pengembang dapat dengan mudah mengimplementasikan metrik ini untuk bahasa pemrograman populer lainnya, seperti Java, Python, atau C#.

Penggunaan metrik ini tidak hanya membantu dalam manajemen proyek tetapi juga dalam memastikan bahwa perangkat lunak yang dikembangkan memiliki kualitas tinggi, mudah dipelihara, dan dapat diandalkan dalam jangka panjang.

Contoh Kasus Penggunaan OO Metrics

Contoh Kasus: **Evaluasi dan Optimisasi Sistem Pemesanan Online Berbasis Objek**

Latar Belakang Proyek

Sebuah perusahaan e-commerce sedang mengembangkan sistem pemesanan online berbasis objek untuk meningkatkan efisiensi operasional mereka. Sistem ini terdiri dari beberapa modul utama, seperti modul pengelolaan pelanggan, modul pemrosesan pesanan, dan modul manajemen inventaris. Tim pengembang ingin mengevaluasi kualitas desain sistem menggunakan *Object-Oriented Metrics* yang diusulkan oleh Mark Lorenz dan Jeff Kidd, dengan fokus pada pemeliharaan, modularitas, dan kompleksitas sistem.

Penerapan Object-Oriented Metrics

1. **Pengukuran Internal Metrics (WMC)**
   Modul pemrosesan pesanan dianalisis menggunakan *Weighted Methods per Class (WMC)* untuk mengidentifikasi kelas dengan tingkat kompleksitas tinggi.
   - **Hasil Temuan**

Salah satu kelas, *OrderProcessor*, memiliki WMC yang tinggi (30), karena mengelola berbagai metode kompleks seperti validasi pesanan, penghitungan diskon, dan integrasi pembayaran.

- **Tindakan Perbaikan**

  Kelas *OrderProcessor* dipecah menjadi beberapa kelas kecil, seperti *PaymentHandler* dan *DiscountCalculator*, untuk mengurangi kompleksitas dan meningkatkan modularitas. Hal ini membuat kode lebih mudah diuji dan dipelihara.

2. **Pengukuran Inheritance Metrics (DIT)**
   Hierarki pewarisan modul manajemen inventaris dianalisis menggunakan *Depth of Inheritance Tree (DIT)*.
   - **Hasil Temuan**

     Salah satu kelas, *InventoryProduct*, memiliki DIT sebesar 6, menunjukkan hierarki pewarisan yang terlalu dalam, yang membuat pemahaman desain menjadi sulit.

   - **Tindakan Perbaikan**

     Hierarki ini disederhanakan dengan menggabungkan beberapa kelas turunan yang memiliki fungsi serupa, sehingga DIT berkurang menjadi 4 tanpa kehilangan fungsionalitas.

3. **Pengukuran External Metrics (CBO)**
   Modul pengelolaan pelanggan dianalisis menggunakan *Coupling Between Objects (CBO)* untuk mengevaluasi hubungan antar kelas.
   - **Hasil Temuan**

     Kelas *CustomerManager* memiliki CBO sebesar 12, karena memiliki ketergantungan pada terlalu banyak kelas lain, seperti *OrderHistory*, *FeedbackHandler*, dan *RecommendationEngine*.

   - **Tindakan Perbaikan**

     Kelas ini diubah untuk memanfaatkan pola desain *Facade* yang mengurangi langsung hubungan antar kelas, sehingga nilai CBO turun menjadi 6.

4. **Pengukuran Size Metrics**
   Sistem dihitung jumlah kelas utama, subsistem, dan kelas pendukungnya.
   - **Hasil Temuan**

     Sistem terdiri dari 20 kelas utama, 5 subsistem, dan 40 kelas pendukung. Data ini digunakan untuk memperkirakan beban kerja proyek. Berdasarkan metrik *Person-Days Per Class*, setiap kelas membutuhkan rata-rata 3 hari kerja,

sehingga total waktu pengembangan sistem diperkirakan memakan waktu sekitar 180 hari kerja untuk seluruh tim.

Hasil dan Manfaat

1. **Peningkatan Maintainability**: Dengan WMC yang lebih rendah, kelas menjadi lebih modular dan mudah dimengerti oleh pengembang baru.
2. **Struktur Hierarki Lebih Sederhana**: Dengan menyederhanakan pewarisan, DIT yang lebih rendah membuat desain lebih intuitif dan mengurangi kemungkinan bug.
3. **Modularitas Lebih Baik**: Dengan pengurangan nilai CBO, sistem menjadi lebih fleksibel terhadap perubahan dan pengujian menjadi lebih mudah dilakukan.
4. **Efisiensi Pengembangan**: Perhitungan beban kerja menggunakan *Person-Days Per Class* membantu manajemen dalam mengalokasikan sumber daya secara efektif.

Kesimpulan

Melalui penerapan *Object-Oriented Metrics* oleh Lorenz dan Kidd, perusahaan berhasil meningkatkan kualitas desain perangkat lunak mereka. Evaluasi metrik ini tidak hanya mengidentifikasi masalah potensial tetapi juga memberikan panduan konkret untuk perbaikan, memastikan bahwa sistem pemesanan online lebih efektif, dapat diandalkan, dan siap untuk kebutuhan masa depan.

Referensi

Lorenz, M., & Kidd, J. (1994). *Object-Oriented Software Metrics*. In *Proceedings of the 1994 IEEE International Conference on Software Engineering* (pp. 302–308). IEEE.

Rodriguez, D., & Harrison, R. (2001). *OO Metrics*. Retrieved from University of Reading.

Dogo Rangsang Research Journal (2019). *Object Oriented Metrics Software Programming*. Retrieved from Journal Dogorangsang.

Bansal, S., & Kumar, S. (2016). "A Review on Object Oriented Software Metrics." *International Journal of Computer Applications*, 139(7), 1-5.

Kostecki, J. A. (1995). Book review: Object-Oriented Software Metrics by Mark Lorenz and Jeff Kidd. ACM SIGSOFT Software Engineering Notes, 20(1), 91–93. https://doi.org/10.1145/225907.773556

Sharma, M. K. (2018). A COMPREHENSIVE INTERPRETATION OF OBJECT ORIENTED METRICS FOR QUALITY REFINEMENT IN SOFTWARE DEVELOPMENT. *International Journal of Advanced Research in Computer Science*, *9*(2), 390–396. https://doi.org/10.26483/ijarcs.v9i2.5836

Harrison, R., Counsell, S., & Nithi, R. (2002). An overview of object-oriented design metrics. *An Overview of Object-oriented Design Metrics*, 230–235. https://doi.org/10.1109/step.1997.615494