



MENGUKUR PERANGKAT LUNAK

Andi Wahyu Rahardjo Emanuel

Mengukur Perangkat Lunak

Andi Wahyu Rahardjo Emanuel

Penerbit ANDI

Mengukur Perangkat Lunak

Oleh: Andi Wahyu Rahardjo Emanuel

Hak Cipta © 2025 pada Penulis

Editor : Robertus Ari
Desain tata letak : Ery Hermawan S
Desain sampul : Ferryan Nugroho P.
Korektor : Dewani H

Hak Cipta dilindungi Undang-Undang.

Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apa pun, baik secara elektronik maupun mekanis, termasuk memfotokopi, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penerbit.

Diterbitkan oleh:

Penerbit ANDI, Yogyakarta

Anggota IKAPI: 005/DIY/85

Dicetak oleh:

Percetakan CV Andi Offset, Yogyakarta

Isi di luar tanggung jawab Percetakan

Ed. I, Cetakan I: 2025

xii + 132 hlm.; 16 x 23 cm

ISBN 978-623-01-5000-5

ISBN 978-623-01-5001-2 (PDF)

DDC'23: 005.3

Program

Persembahan

*Buku ini saya persembahkan
untuk keluarga tercinta
Bernadeta Ariyanti Retno Wulandari dan
Marcelinus Dimas Satrio Yudanto*

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Prakata

Puji syukur kami aturkan kepada Tuhan Yang Maha Kuasa karena karunia-Nya kami telah berhasil menyelesaikan buku *Mengukur Perangkat Lunak* ini. Buku ini kami hadirkan sebagai wujud nyata kontribusi kami dalam dunia Pengukuran Perangkat Lunak (*Software Metrics*) yang merupakan salah satu bidang ilmu di Rekayasa Perangkat Lunak (*Software Engineering*) yang masih sering terabaikan sampai sekarang, khususnya di Indonesia.

Buku ini kami buat sebagai referensi bagi siapa saja, khususnya bagi mahasiswa, pemula, praktisi, dan profesional yang ingin tahu tentang berbagai pengukuran yang memungkinkan dalam sebuah proyek perangkat lunak. Sifat perangkat lunak yang tidak dapat dilihat secara langsung memang menimbulkan tantangan tersendiri untuk mengukurnya, dibandingkan dengan berbagai proyek lain yang bisa dilihat secara langsung. Buku ini diharapkan membantu memberikan perspektif yang mencukupi tentang aspek-aspek apa saja yang dapat dilakukan dalam pengukuran perangkat lunak mulai dari metrik produk, metrik kualitas, metrik proses, metrik orang dan juga berbagai perangkat yang dapat dipergunakan untuk mengukur perangkat lunak.

Besar harapan kami buku ini dapat memberikan kontribusi kepada masyarakat. Kami menyadari bahwa buku ini kemungkinan masih memiliki kekurangan di sana sini yang akan kami perbaiki pada revisi-revisi yang akan datang. Kritik dan saran atas buku ini akan sangat kami harapkan untuk kemajuan bersama dalam bidang ilmu pengetahuan dan teknologi, khususnya bidang rekayasa perangkat lunak.

Klaten, 1 Juli 2025

Andi Wahyu Rahardjo Emanuel

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Daftar Isi

Persembahan	iii
Prakata	v
Daftar Isi	vii
Daftar Tabel	ix
Daftar Gambar	xi
Bab 1 Pendahuluan	1
A. Definisi Perangkat Lunak	1
B. Tantangan Pengembangan Perangkat Lunak	6
C. Pentingnya Mengukur Kualitas dan Kuantitas Perangkat Lunak	13
D. Definisi Metrik Perangkat Lunak	15
Bab 2 Metrik Produk	19
A. Pendahuluan	19
B. Metrik Ukuran	20
C. Metrik Usaha, Biaya dan Waktu	32
D. Metrik Bahasa Pemrograman Berorientasi Objek	36
Bab 3 Metrik Kualitas	53
A. Pendahuluan	53
B. Metrik Kinerja (<i>Performance</i>)	56
C. Metrik Keamanan (<i>Security</i>)	60
D. Metrik Kebergunaan (<i>Usability</i>)	63
E. Metrik Keandalan (<i>Reliability</i>)	69
F. Metrik Pemeliharaan (<i>Maintainability</i>)	71
G. Metrik Portabilitas (<i>Portability</i>)	76
H. Metrik Waktu-Nyata (<i>Real-Time</i>)	77
Bab 4 Metrik Proses	79
A. Pendahuluan	79
B. Metrik Pemantauan Pembentukan Cacat (<i>Defect</i> atau <i>Bug</i>)	90

C. Metrik Perubahan Kebutuhan (<i>Requirement Changes</i>)	93
D. Metrik Waktu Pemrograman (<i>Programming Time</i>)	93
E. Metrik Proses Evolusi	94
Bab 5 Metrik Orang	95
A. Pendahuluan	95
B. Metrik <i>Person-Month</i>	97
C. Metrik LOC per Jam	98
D. Metrik FP per Jam	99
E. Metrik Kebutuhan per Jam	99
Bab 6 Perangkat Lunak Pengukur Metrik	101
A. Pendahuluan	101
B. Perangkat Lunak Pendukung	102
C. Perangkat Penghitung Metrik	109
D. Perangkat Lingkungan Pengembangan Terintegrasi (<i>Integrated Development Environment - IDE</i>)	118
Bab 7 Penutup	125
Daftar Pustaka	127

Daftar Tabel

Tabel 2.1 Konversi FP ke LOC (QSM, 2025).....	25
Tabel 3.1 Laju Insiden Keamanan di Berbagai Industri.....	63
Tabel 3.2 Standar Industri untuk Waktu Penerapan Tambalan.....	63

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

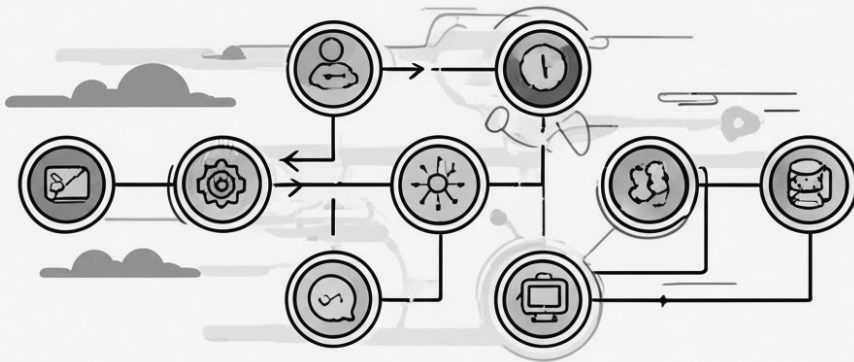
Daftar Gambar

Gambar 1.1 Penyusun Perangkat Lunak	2
Gambar 1.2 Tiga Komponen Penyusun Sistem Informasi	4
Gambar 1.3 Komponen dari Teknologi Informasi.....	6
Gambar 1.4 Empat Kekangan Pengembangan Perangkat Lunak	7
Gambar 1.5 4P dalam Pengembangan Sistem Perangkat Lunak.....	9
Gambar 1.6 Manajemen Proyek Perangkat Lunak.....	11
Gambar 1.7 Siklus Hidup Pengembangan Perangkat Lunak <i>Waterfall</i>	12
Gambar 2.1 Situs Laman Web dari IFPUG	27
Gambar 2.2 Informasi Jejak Memori di Sistem Operasi Windows 11	28
Gambar 2.3 Tampilan Jejak Memori Aplikasi di Ubuntu.....	29
Gambar 2.4 Jejak Memori Aplikasi di Android.....	30
Gambar 2.5 Jejak Memori Aplikasi Browser Safari di MacOS.....	31
Gambar 2.6 Model Estimasi COCOMO II.....	33
Gambar 2.7 Situs Simulasi COCOMO II	35
Gambar 2.8 Ilustrasi Sebuah Diagram Kelas	37
Gambar 2.9 Contoh <i>Use Case Diagram</i>	39
Gambar 2.10 Contoh <i>Activity Diagram</i> Pembelian Tiket oleh Penumpang.....	40
Gambar 2.11 Contoh Sequence Diagram Pengenalan Wajah.....	41
Gambar 2.12 <i>Coupling</i> Antar-Kelas	46
Gambar 2.13 Contoh Package Diagram	47
Gambar 2.14 Pewarisan dalam Kelas Diagram	48
Gambar 2.15 Ilustrasi Kohesi dalam Kelas	52
Gambar 3.1 Ilustrasi Sistem Pengolah Transaksi (TPS)	58
Gambar 3.2 Situs Laman Web Yayasan OWASP.....	61
Gambar 3.3 Laman Awal Situs Web UEQ-Online	67
Gambar 4.1 Model Pengembangan Perangkat Lunak <i>Waterfall</i>	80
Gambar 4.2 Pengembangan Perangkat Lunak Model – V	81
Gambar 4.3 Model Pengembangan Perangkat Lunak <i>Spiral</i>	83
Gambar 4.4 Model Pengembangan Perangkat Lunak <i>Prototyping</i>	84

Gambar 4.5 Model Pengembangan Perangkat Lunak <i>Extreme Programming</i>	86
Gambar 4.6 Contoh Papan Kanban.....	87
Gambar 4.7 Model Pengembangan Perangkat Lunak SCRUM	89
Gambar 6.1 Situs XAMPP	103
Gambar 6.2 Tampilan Depan XAMPP	104
Gambar 6.3 Tampilan PHPMyAdmin.....	105
Gambar 6.4 Laman Situs Ubuntu.....	106
Gambar 6.5 Tampilan Sistem Operasi Ubuntu	107
Gambar 6.6 Situs Temurin JDK oleh Adoptium	108
Gambar 6.7 Situs OpenJDK.....	109
Gambar 6.8 Situs <i>SonarQube</i>	110
Gambar 6.9 Tampilan SonarQube Community Edition	111
Gambar 6.10 Situs JavaNCSS.....	112
Gambar 6.11 Situs CCCC.....	113
Gambar 6.12 Situs PMD	114
Gambar 6.13 Situs Perangkat CKJM	115
Gambar 6.14 Situs SLOCCount	116
Gambar 6.15 Situs Jdepend	117
Gambar 6.16 Laman Web Situs Visual Studio 2022.....	118
Gambar 6.17 Situs Eclipse IDE	120
Gambar 6.18 Situs Apache Ant.....	121
Gambar 6.19 Situs Apache Maven	122
Gambar 6.20 Situs Gradle.....	123

BAB 1

Pendahuluan

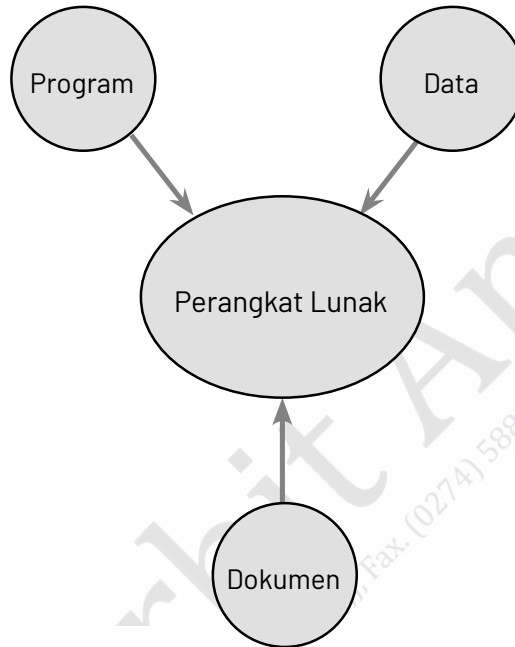


A. Definisi Perangkat Lunak

Perangkat lunak atau *software* adalah sebuah produk yang dirancang dan dibangun oleh seorang pengembang atau pemrogram perangkat lunak (*software engineer*). Perangkat lunak tersebut perlu dipasang pada sebuah perangkat keras (*hardware*) untuk dapat bekerja. Setidaknya terdapat tiga komponen yang menyusun sebuah sistem perangkat lunak:

1. Aplikasi/program: adalah sekumpulan perintah yang disusun untuk dapat mengolah data di dalam sebuah komputer atau laptop. Sekumpulan perintah ini bisa berupa berkas hasil kompilasi ataupun skrip program.
2. Data: adalah sekumpulan fakta yang dituliskan dalam bentuk digital agar dapat diolah oleh aplikasi/program. Data ini biasanya bisa disimpan dalam berkas teks ataupun dalam sistem basis data.

3. Dokumen: sekumpulan dokumentasi mengenai penggunaan ataupun pemeliharaan dari aplikasi agar dapat berfungsi sesuai tujuannya. Dokumen-dokumen ini misalnya manual pengguna, manual pemeliharaan, manual instalasi, dan lain-lainnya.



Gambar 1.1 Penyusun Perangkat Lunak

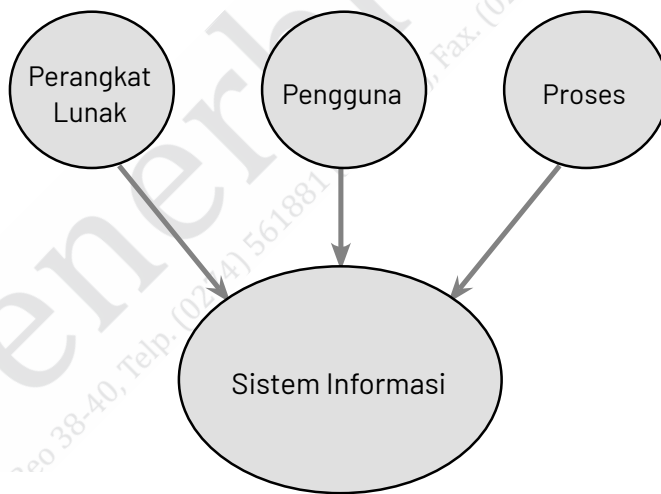
Sebagai contoh misalnya aplikasi perdagangan elektronik atau *e-commerce* seperti *Tokopedia*, *Shopee*, *Zalora*, atau *Blibli* yang sekarang ini populer, aplikasi ini juga memiliki 3 komponen seperti yang diilustrasikan di Gambar 1.1. Komponen pertama adalah aplikasi/program itu sendiri yang terdiri dari program yang berada di server *Tokopedia*, *Shopee*, *Zalora*, atau *Blibli*, dan di handphone kita. Selain program aplikasi-aplikasi tersebut juga memiliki data yaitu berbagai macam informasi tentang barang-barang yang dijual beserta foto, deskripsi, penjual, dan juga data-data pembeli. Komponen ketiga adalah dokumen yang berisi berbagai macam informasi mengenai bagaimana cara mendaftar, bagaimana cara membeli, bagaimana melaporkan kalau ada keluhan, dan lain-lainnya. Jadi, tanpa ketiga komponen tersebut, niscaya aplikasi *Tokopedia*, *Shopee*, *Zalora*, atau *Blibli* tersebut tidak akan berfungsi dengan baik.

Sistem perangkat lunak dapat kita temui dalam berbagai bentuk dan ukuran. Kegunaan dari perangkat lunak tersebut juga berbagai macam untuk menunjang kelancaran pekerjaan dan tugas dari penggunanya. Beberapa contoh jenis-jenis perangkat lunak yang dapat kita temui sehari-hari, antara lain:

1. Sistem Operasi: sistem operasi adalah perangkat lunak yang terpasang pada sebuah komputer atau laptop yang berfungsi mengelola berbagai macam sumber daya yang ada pada perangkat keras di komputer atau laptop tersebut. Contoh Sistem Operasi adalah *Windows 11*, *Ubuntu Linux*, *MacOS*, *Android*, *IOS*, dan lain-lainnya. Berbagai aplikasi jenis lainnya akan dapat dipasang dan berfungsi dengan baik pada komputer atau laptop yang telah dipasang dengan Sistem Operasi ini.
2. Perangkat lunak produktivitas: perangkat lunak produktivitas biasanya dipakai untuk para pekerja atau pelajar/mahasiswa agar bisa lebih produktif saat bekerja dan belajar. Sebagai contohnya adalah perangkat lunak pengolah kata seperti *Microsoft Word*, pengolah data seperti *Microsoft Excel*, pengolah presentasi seperti *Microsoft PowerPoint*, pengolah diagram seperti *draw.io* dan lain-lainnya.
3. Perangkat lunak permainan/gim: perangkat lunak yang dipasang pada komputer atau laptop atau konsol untuk alat hiburan, khususnya permainan yang sifatnya berkompetisi. Contohnya adalah *Genshin Impact*, *Black Myth Wukong*, *Minecraft*, *Fortnite*, dan lain-lainnya. Beberapa gim tertentu misalnya dalam kategori gim AAA biasanya membutuhkan perangkat keras khusus misalnya GPU (*Graphical Processing Unit*) untuk dapat bekerja dengan baik.
4. Perangkat lunak multimedia: perangkat lunak yang dipergunakan untuk menghasilkan atau menikmati berbagai berkas multimedia seperti foto, video, suara, dan lain-lain. Contoh aplikasi ini misalnya *CapCut*, *Adobe Creative Cloud*, *Blender*, dan lain-lain.
5. Perangkat lunak media sosial: perangkat lunak yang memudahkan dalam menyebarkan konten-konten yang dibuat oleh pengguna untuk dapat dinikmati atau dipergunakan oleh pengguna lainnya. Contoh aplikasi ini misalnya *Facebook*, *Youtube*, *Vidio*, *Instagram*, *TikTok*, dan lain-lain.

6. Perangkat lunak perdagangan elektronik (*e-commerce*): perangkat lunak yang dibangun untuk menunjang berbagai transaksi dan perdagangan elektronik dalam berbagai bentuknya. Contoh aplikasi ini misalnya *Shopee*, *Tokopedia*, *Blibli*, *Zalora*, *Gojek*, *Grab*, dan lain-lain.
7. Perangkat lunak yang ditanamkan pada perangkat keras (*embedded software*): perangkat lunak ini tidak terlihat secara langsung tetapi ditanam dalam perangkat keras untuk mendukung kinerja dari perangkat tersebut. Mesin-mesin atau alat-alat modern biasanya memiliki perangkat lunak jenis ini misalnya pada mesin cuci satu pintu, kulkas, pendingin ruangan, mobil, dan lain-lainnya.
8. Dan lain-lain.

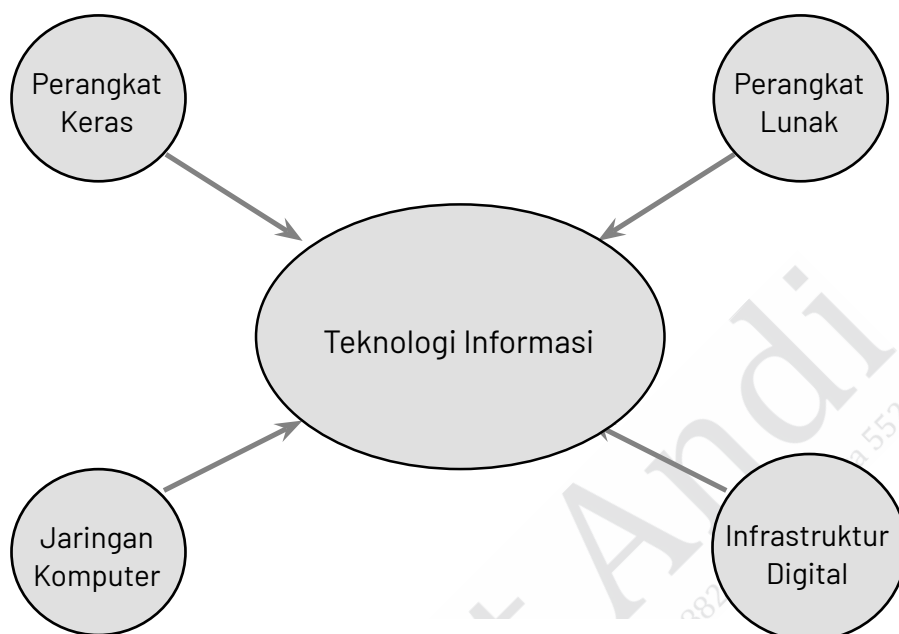
Selain istilah perangkat lunak, juga terdapat istilah Sistem Informasi yang ruang lingkupnya lebih luas. Sistem Informasi merupakan gabungan dari sistem perangkat lunak, para pengguna, dan juga proses yang ditujukan untuk mencapai tujuan dari diinginkan dari suatu perusahaan atau organisasi seperti yang digambarkan dalam Gambar 1.2 berikut ini.



Gambar 1.2 Tiga Komponen Penyusun Sistem Informasi

Contoh dari sebuah sistem informasi adalah Sistem Informasi Sekolah yang terdiri dari 3 bagian: aplikasi, pengguna, dan proses. Bagian pertama adalah aplikasi yang dikembangkan oleh pengembang perangkat lunak untuk sekolah tersebut. Aplikasi tersebut kemudian dipakai oleh para penggunanya yaitu: murid, guru, kepala sekolah, dan lain-lain untuk menjalankan berbagai macam proses yang seharusnya terjadi dalam sebuah sekolah. Proses-proses tersebut misalnya proses pendaftaran siswa baru, proses pengajaran, proses pelaporan hasil belajar, dan berbagai macam proses lainnya. Kombinasi dari aplikasi, pengguna dan proses-proses tersebut akan menunjang pelaksanaan pendidikan di sekolah agar bisa berjalan dengan lebih efisien dan efektif. Dapat dilihat pada penjelasan dan kedua Gambar 1.1 dan 1.2 di atas bahwa salah satu komponen atau bagian yang penting adalah aplikasi atau program itu sendiri. Aplikasi merupakan bagian yang dikembangkan oleh tim pengembang perangkat lunak yang dituntut untuk menghasilkan produk perangkat lunak yang berkualitas tinggi.

Selain berbagai istilah di atas, juga terdapat istilah Teknologi Informasi yang juga sering muncul di berbagai media. Teknologi Informasi merupakan gabungan dari perangkat keras (*hardware*), perangkat lunak (*software*), jaringan komputer, dan berbagai infrastruktur digital yang dipergunakan untuk menerima, memproses, menyebarkan, menyimpan, dan mengamankan data atau informasi. Perangkat lunak tidak bisa berdiri sendiri tanpa adanya perangkat keras di mana perangkat lunak tersebut dipasang dan dipakai oleh penggunanya. Perangkat keras dan perangkat lunak tersebut dihubungkan oleh jaringan komputer dan berbagai infrastruktur digital lainnya untuk mengolah data atau informasi agar menjadi berguna bagi siapa saja yang membutuhkannya. Ilustrasi dari Teknologi Informasi dapat dilihat pada Gambar 1.3.



Gambar 1.3 Komponen dari Teknologi Informasi

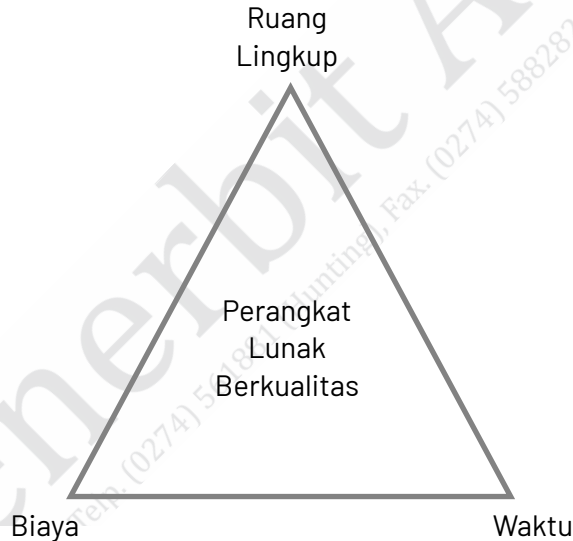
B. Tantangan Pengembangan Perangkat Lunak

Ketergantungan manusia pada Teknologi Informasi semakin hari semakin besar dan memengaruhi hampir pada setiap aspek kehidupan. Sebagai akibatnya, perangkat lunak semakin lama dituntut untuk menjadi semakin besar, semakin kompleks, dan juga diharapkan untuk semakin lama menjadi semakin pintar. Dalam proses pembuatan sistem perangkat lunak biasanya dikekang oleh 4 aspek utama seperti yang digambarkan pada Gambar 1.4, yaitu:

1. Ruang Lingkup: sebuah proyek perangkat lunak memiliki ruang lingkup yang menentukan kemampuan-kemampuan apa saja yang perlu dimiliki oleh sistem perangkat lunak tersebut. Ruang lingkup ini juga yang menentukan apa yang bisa dilakukan dan apa yang tidak bisa dilakukan oleh perangkat lunak.
2. Waktu: pengerjaan proyek perangkat lunak biasanya memiliki waktu yang terbatas. Sebuah proyek pembuatan perangkat lunak skala kecil bisa membutuhkan waktu satu sampai tiga bulan untuk

penyelesaiannya, sedangkan proyek pembuatan perangkat lunak skala menengah mungkin membutuhkan waktu dari tiga bulan sampai dengan satu tahun, sedangkan proyek pembuatan perangkat lunak skala besar atau yang super besar membutuhkan waktu tahunan untuk menyelesaikannya.

3. Anggaran: anggaran atau bujet harus dialokasikan dalam mengerjakan sebuah proyek. Kisaran biaya pembangunan perangkat lunak bisa mulai dari skala jutaan rupiah sampai dengan miliaran tergantung besar perangkat lunak yang dikembangkan.
4. Kualitas: sebuah proyek perangkat lunak dikatakan berkualitas apabila memenuhi atau melebihi ekspektasi dari para penggunanya. Aspek kualitas ini dapat diukur dan dibahas dengan lebih terperinci di Bab 3.



Gambar 1.4 Empat Kekangan Pengembangan Perangkat Lunak

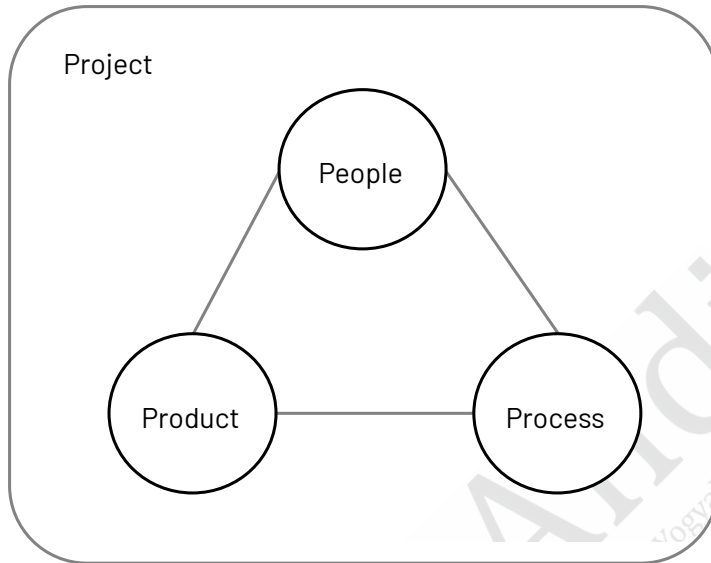
Adalah suatu tantangan yang besar bagi tim pengembang perangkat lunak untuk dapat menyeimbangkan keempat kekangan tersebut di atas karena sifat ketiganya yang cenderung bertolak belakang. Berdasarkan bagaimana tim pengembang perangkat lunak dapat mengendalikan keempat kekangan tersebut, maka sebuah proyek perangkat lunak dapat dikategorikan menjadi tiga kemungkinan:

1. Berhasil: apabila keempat aspek di atas yaitu ruang lingkup, waktu, biaya, dan kualitas semuanya dapat dicapai sesuai yang direncanakan. Artinya, aplikasi yang dikembangkan memiliki ruang lingkup yang diharapkan, dengan anggaran dan waktu sesuai prediksi dan juga memiliki kualitas sesuai harapan juga.
2. Agak atau kurang berhasil: apabila satu atau lebih dari keempat aspek tersebut harus dikorbankan untuk bisa meraih kekangan-kekangan yang lainnya. Misalnya semua ruang lingkup dari proyek tercapai tetapi waktunya molor dan biayanya membengkak. Atau misalnya karena waktu dan biaya tidak bisa dikompromikan, maka ruang lingkup dari perangkat lunak dikurangi.
3. Gagal: apabila keempat aspek di atas semuanya tidak dapat dicapai dan perangkat lunak tersebut akhirnya tidak dapat dipergunakan sama sekali.

Sebuah organisasi global yang dinamakan *Standish Group* (<http://www.standishgroup.com>), secara rutin mengadakan studi mengenai tingkat keberhasilan dari proyek pengembangan perangkat lunak. Sampai dengan hasil studi terakhir di tahun 2020 terungkap bahwa tingkat keberhasilan dari proyek perangkat lunak masih relatif kecil meskipun terus mengalami peningkatan di setiap tahunnya. Fakta tingkat keberhasilan yang rendah ini juga menjadi alasan berkembangnya proses pengembangan perangkat lunak secara cepat seperti *Extreme Programming*, *Scrum*, *Kanban*, dan lain-lainnya.

Pemantauan perkembangan proyek perangkat lunak pada dasarnya sulit untuk dilakukan karena sifatnya yang berbeda dibandingkan dengan proyek-proyek lainnya yang cenderung kasatmata. Hasil dari proyek perangkat lunak yang berupa program yang harus dibuat oleh tim pengembang memiliki sifat yang tidak bisa disentuh dengan indra kita (*intangible*), jadi perlu metode-metode khusus yang harus dikembangkan untuk memastikan perkembangannya dapat terpantau dengan baik. Salah satu indikator yang paling sering dipakai yaitu besar kode sumber saja tidak bisa menjadi jaminan bahwa proyek tersebut telah mencapai perkembangan yang telah direncanakan.

Dalam pembuatan sebuah sistem perangkat lunak yang baik, terdapat 4 komponen besar yang terlibat yang disebut 4 P: *Project*, *Product*, *People*, dan *Process*, seperti yang ditunjukkan pada Gambar 1.5 berikut ini.



Gambar 1.5 4P dalam Pengembangan Sistem Perangkat Lunak

1. *Project* atau Proyek: adalah aktivitas payung yang melingkupi ketiga komponen di atas yang dipelajari dari manajemen proyek perangkat lunak. Proyek akan memberikan dukungan penuh terhadap ketiga P yang lain dalam berbagai bentuk sehingga proses pengembangan perangkat lunak dapat berjalan dengan baik.
2. *Product* atau Produk: adalah produk perangkat lunak itu sendiri dari 3 bagian yaitu aplikasi/program, data, dan dokumentasi. Produk-produk ini akan diantarkan oleh tim pengembang ke klien untuk dapat dipergunakan.
3. *People* atau Orang: adalah orang-orang yang terlibat secara langsung dalam pembuatan sistem perangkat lunak tersebut, terlebih adalah tim pengembangan perangkat lunak. Orang-orang yang terlibat dan berkepentingan dalam pengembangan sistem perangkat lunak biasanya disebut para pemangku kepentingan atau *stakeholder*.
4. *Process* atau Proses: adalah berbagai macam proses yang harus dilalui untuk bisa membuat sebuah sistem perangkat lunak yang baik. Proses yang dipakai untuk pengembangan perangkat lunak disebut Siklus Hidup Pengembangan Perangkat Lunak atau *Software Development Life Cycle (SDLC)*.

Semua dari keempat komponen di atas harus dipantau untuk menjamin pengembangan perangkat lunak agar berhasil dengan baik. Berikut ini adalah cara-cara yang biasanya dilakukan untuk memantau perkembangan dari proyek perangkat lunak:

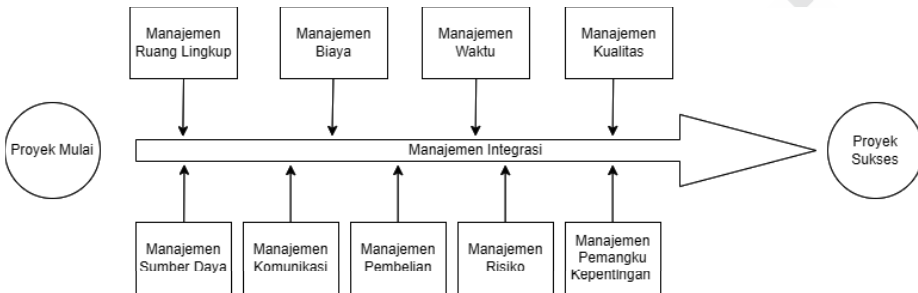
1. Melakukan teknik-teknik manajemen proyek yang telah disesuaikan untuk proyek perangkat lunak.
2. Memilih metode-metode dari rekayasa perangkat lunak yang baik: metode pengembangan perangkat lunak ini disebut dengan istilah Siklus Hidup Pengembangan Perangkat Lunak atau SDLC (*Software Development Life Cycle*).
3. Memantau dengan menggunakan berbagai macam metrik perangkat lunak: memilih berbagai metrik perangkat lunak yang dianggap relevan sesuai dengan tujuan dari perusahaan.

Manajemen proyek adalah penerapan dari pengetahuan, keterampilan, perangkat, dan teknik untuk berbagai proyek untuk memenuhi tujuan dari proyek tersebut. Pengembangan sistem perangkat lunak merupakan sebuah proyek karena sifatnya yang sementara untuk menciptakan sebuah produk (Schwalbe, 2021). Penerapan manajemen proyek biasanya meliputi sepuluh area pengetahuan, yaitu:

1. Manajemen Integrasi: pengelolaan semua area pengetahuan secara terintegrasi.
2. Manajemen Ruang Lingkup: pengelolaan ruang lingkup pengerjaan perangkat lunak.
3. Manajemen Waktu: pengelolaan waktu proyek yang terdiri dari perencanaan jadwal proyek dan pelacakan jadwal proyek.
4. Manajemen Biaya: pengelolaan semua biaya dalam proyek.
5. Manajemen Kualitas: pengelolaan aspek-aspek yang berkaitan dengan ketiga kekanan utama (ruang lingkup, biaya, dan waktu) dan aspek kualitas lainnya.
6. Manajemen Sumber Daya: pengelolaan berbagai macam sumber daya dalam menunjang penyelesaian proyek.
7. Manajemen Pembelian: pengelolaan berbagai macam pembelian yang dilakukan selama penyelesaian proyek.
8. Manajemen Komunikasi: pengelolaan komunikasi dari semua pemangku kepentingan dari proyek.

9. Manajemen Risiko: pengelolaan berbagai macam risiko yang mungkin berpengaruh terhadap keberhasilan proyek.
10. Manajemen Pemangku Kepentingan: pengelolaan semua pihak yang berkepentingan terhadap keberhasilan proyek.

Ilustrasi dari penerapan kesepuluh area pengetahuan dapat dilihat pada Gambar 1.6 berikut ini.

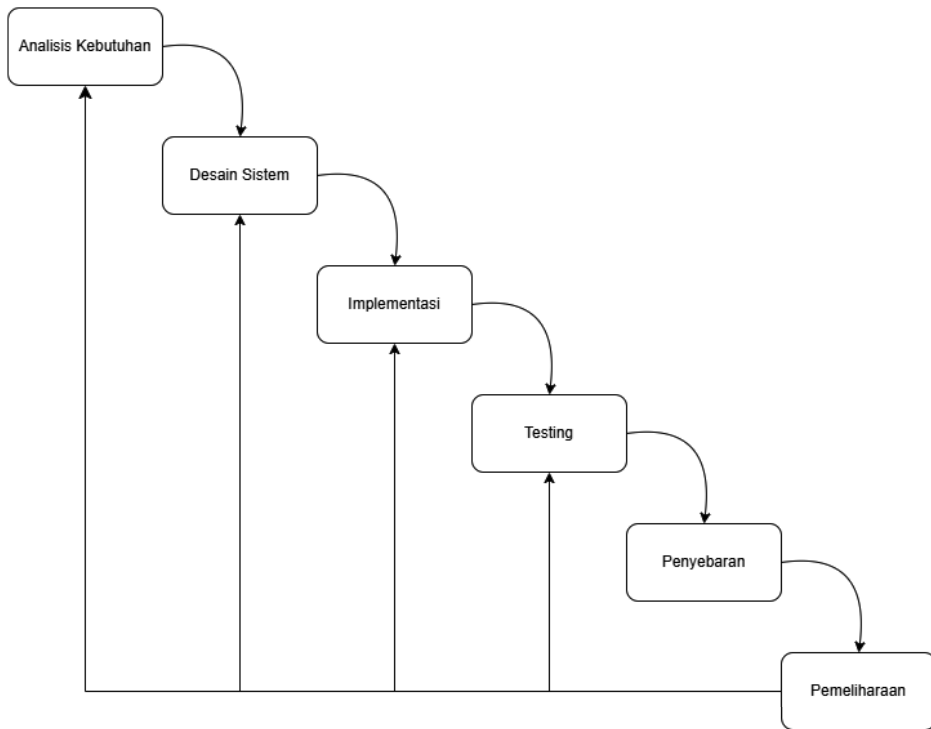


Gambar 1.6 Manajemen Proyek Perangkat Lunak

Yang menjadi panduan penerapan manajemen proyek di proyek perangkat lunak adalah dokumen PMBOK (*Project Management Body of Knowledge*) dan seseorang dapat mengambil sertifikasinya dari pengelolaan proyek yang dinamakan PMP (*Project Management Profesional*) yang dikeluarkan oleh organisasi SEI (*Software Engineering Institute*) (*Project Management Institute, 2021*).

Sementara itu pemilihan metode-metode rekayasa perangkat lunak adalah pemilihan metode yang tepat saat pembangunan sistem perangkat lunak sesuai dengan karakternya. Terdapat dua pengelompokan metode yaitu berbasis rencana dan *Agile*. Metode berbasis rencana biasanya bersifat lebih teratur dan terencana di awal seperti siklus pengembangan *Waterfall*, *Prototyping*, *Spiral*, dan lain-lainnya. Siklus pengembangan berbasis metode *Agile* biasanya bersifat lebih peka terhadap perubahan di tengah jalan, seperti contohnya adalah pengembangan berbasis *Scrum*, *Kanban*, *Extreme Programming*, dan lain-lainnya. Petunjuk dalam penggunaan dan penerapan SDLC ini dapat ditemukan dalam buku panduan yang disebut SWEBOK (*Software Engineering Body of Knowledge*) yang dibuat oleh organisasi IEEE Computer Society (*IEEE Computer Society, 2024*). Penjelasan lebih detail mengenai hal ini dapat ditemukan

di Bab 4 tentang Metrik Proses. Salah satu contoh SDLC berbasis rencana adalah *Waterfall* seperti yang digambarkan pada Gambar 1.7 berikut ini.



Gambar 1.7 Siklus Hidup Pengembangan Perangkat Lunak *Waterfall*

Fokus buku ini adalah pada bagian ketiga yaitu memantau dan menggunakan berbagai metrik perangkat lunak. Ini dikarenakan bahwa hal ini tidak kalah pentingnya dibandingkan kedua faktor yang telah dijelaskan sebelumnya. Mengukur suatu sistem perangkat lunak, terlebih pada saat sistem tersebut sebelum, sedang dibuat atau sudah memasuki masa pemeliharaan, akan memiliki banyak manfaat, yaitu:

1. Membantu dalam mengestimasi biaya dan usaha yang diperlukan untuk membangun perangkat lunak tersebut: estimasi yang baik dan mendekati nilai sebenarnya akan sangat membantu manajemen perusahaan untuk dapat memberikan dukungan yang maksimal bagi tim pengembang perangkat lunak. Berbagai estimasi tersebut dapat dilihat pada Bab 2 Metrik Produk.

2. Meningkatkan produktivitas dari tim pengembang: berbagai pengukuran, khususnya yang berkaitan dengan ukuran perangkat lunak dan kinerja dari para pengembang akan dapat membantu memberikan masukan bagi tim pengembang untuk senantiasa meningkatkan produktivitas mereka. Berbagai aspek produktivitas dapat dilihat pada Bab 4 Metrik Proses dan Bab 5 Metrik Orang.
3. Meningkatkan kualitas perangkat lunak dengan mengukur aspek-aspek yang berkaitan dengan kualitas perangkat lunak: berbagai aspek yang berkaitan dengan kualitas juga bisa diukur sehingga dapat ditingkatkan secara terus-menerus dan berkelanjutan. Berbagai aspek kualitas akan dibahas di Bab 3 buku ini.
4. Meningkatkan keandalan dari sistem perangkat lunak: berbagai aspek yang berkaitan dengan keandalan perangkat lunak juga bisa diukur sehingga sistem perangkat lunak tersebut dapat berkinerja sangat baik pada saat diterapkan. Keandalan juga merupakan salah satu aspek kualitas yang dibahas pada buku ini.
5. Meningkatkan metode dan perangkat: berbagai metode dan perangkat juga bisa diterapkan dengan baik dan efisien, sehingga dapat mendukung tim pengembang dan tim pemelihara agar dapat bekerja dengan baik. Berbagai perangkat yang menunjang pengumpulan berbagai metrik perangkat lunak dapat dilihat pada Bab 6 Perangkat Lunak Pengukur Metrik.

C. Pentingnya Mengukur Kualitas dan Kuantitas Perangkat Lunak

Tom Demarco mengatakan “Anda tidak bisa mengendalikan yang tidak bisa Anda lihat” (DeMarco, 1982), jadi kemampuan mengukur produk merupakan hal dasar yang harus dikuasai sebelum kita bisa mengendalikan proses pembuatan sebuah produk tersebut. Dalam hal sistem perangkat lunak, kemampuan mengukur perangkat lunak menjadi sangat penting dalam proses pengembangan perangkat lunak sehingga berbagai macam intervensi bisa dilakukan untuk meningkatkan kemungkinan keberhasilan dalam pengembangan perangkat lunak tersebut.

Berbagai macam pengukuran harus ditetapkan dan kemudian dikumpulkan agar bisa mendapatkan gambaran yang baik tentang bagaimana proses pengembangan perangkat lunak tersebut apakah sudah sesuai atau belum sesuai dengan yang direncanakan. Adapun metode-metode pengumpulan berbagai macam pengukuran tersebut dapat dilakukan dengan tahapan sebagai berikut (Micallef, 2022):

1. Menentukan tujuan dari pengumpulan data: tujuan dari pengumpulan data harus ditetapkan dahulu agar pemilihan metrik perangkat lunak yang dipakai menjadi lebih terarah. Tujuan yang bisa ditetapkan misalnya untuk mengurangi biaya, meningkatkan kinerja, meningkatkan efisiensi, meningkatkan kualitas, dan lain-lainnya.
2. Menentukan daftar pertanyaan yang ingin dicapai dari proses pengumpulan data: dari tujuan yang telah ditetapkan tersebut kemudian diturunkan menjadi beberapa daftar pertanyaan yang menjadi fokus dalam pengukuran.
3. Mengembangkan kategori data: dari daftar pertanyaan tersebut kemudian ditetapkan beberapa kategori data yang perlu dikumpulkan dan dicari metrik apa saja yang relevan.
4. Mendesain dan menguji formulir atau program pengumpulan data: mendesain teknik untuk bisa mendapatkan data-data yang diperlukan baik secara manual (mengumpulkan formulir dari responden) ataupun secara otomatis dengan menggunakan perangkat yang dapat mengumpulkan metrik-metrik tersebut secara otomatis.
5. Mengumpulkan dan memvalidasi data: data yang diperlukan kemudian dikumpulkan dan kemudian dilakukan berbagai macam teknik validasi untuk menjamin keabsahan data.
6. Menganalisis data: data yang terkumpul dan yang sudah valid kemudian dianalisis untuk bisa menjawab berbagai macam pertanyaan yang dibuat di nomor 2 dan pada akhirnya bisa menjawab tujuan yang ditetapkan di nomor 1.

Setelah semua metrik pengukuran perangkat lunak dijelaskan pada Bab 2 sampai dengan Bab 5, pada Bab 6 di buku ini akan dibahas mengenai perangkat apa saja yang dapat dipergunakan agar pengukuran perangkat lunak tersebut dapat berjalan dengan efektif dan efisien.

D. Definisi Metrik Perangkat Lunak

Metrik didefinisikan sebagai sebuah nilai yang diekspresikan dalam unit yang diasosiasikan dengan sifat dari sebuah objek, orang, proses, dan lain-lain. Jadi, metrik perangkat lunak adalah sebuah nilai yang diekspresikan dalam unit-unit yang diasosiasikan dengan properti dari sebuah sistem perangkat lunak. Beberapa contoh metrik perangkat lunak yang biasanya dipergunakan antara lain:

1. Jumlah baris kode dari program
2. Jumlah orang-hari yang diperlukan
3. Jumlah cacat yang dihasilkan
4. Berbagai macam indeks perangkat lunak lainnya yang relevan
5. Dan lain-lain

Kondisi dari proses pengembangan perangkat lunak tentu saja tidak dapat digambarkan hanya dari satu atau dua metrik saja, melainkan diperlukan beberapa metrik sekaligus yang nilainya digabungkan untuk membentuk nilai komposit. Tujuan dari penggunaan beberapa metrik ini adalah:

1. Berguna untuk membandingkan dan memprediksi pencapaian terakhir dari aplikasi perangkat lunak: peningkatan nilai-nilai dari metrik-metrik tersebut menandakan terjadi perkembangan dalam proses pengembangan dari perangkat lunak.
2. Berguna untuk mengindikasikan status terakhir dari perangkat lunak: kompilasi dari berbagai metrik tersebut menandakan fase yang telah dilalui dan kondisi terakhir dari sistem perangkat lunak.

Pemilihan metrik-metrik yang akan dipakai tentu saja harus melalui beberapa pemikiran dan pemahaman. Beberapa properti yang diinginkan dari metrik perangkat lunak agar dapat mencapai tujuan yang diinginkan adalah sebagai berikut.

1. Valid dan dapat diandalkan: hasil pengukuran perangkat lunak diharapkan benar, sah, dan dapat diterima semua pihak yang terlibat dalam pengembangan atau pemakaian perangkat lunak.
2. Objektif dan presisi: hasil pengukuran perangkat lunak seharusnya objektif atau bernilai yang sama dan tidak tergantung siapa yang mengukurnya. Tingkat presisi yang mencukupi juga dibutuhkan agar hasil analisis dari pengukuran tersebut dapat tepat sasaran.
3. Intuitif: nilai-nilai dan satuan dari pengukuran perangkat lunak haruslah dengan mudah dipahami oleh semua pihak yang terlibat pada pengembangan, pemeliharaan, dan pemakaian perangkat lunak tersebut.
4. Tangguh (*robust*): metrik perangkat lunak tersebut seharusnya dapat menggambarkan kondisi sesungguhnya dari sistem perangkat lunak dan masih dapat menggambarkan kondisi tersebut meskipun data yang dikumpulkan tidak lengkap.
5. Dapat diotomatisasi dan praktis secara ekonomi: metrik-metrik tersebut seharusnya dapat dikumpulkan secara otomatis dan juga biaya pengumpulannya juga terjangkau.

Metrik Perangkat Lunak atau *Software Metrics* juga didefinisikan sebagai sebuah nilai yang diekspresikan dalam unit yang berhubungan dengan perangkat lunak (Meyer dkk., 2009). Metrik Perangkat Lunak berguna untuk mengindikasikan bagaimana keadaan dari sebuah sistem perangkat lunak dan juga memungkinkan perbandingan dan perkiraan tentang apa yang telah dicapai oleh sebuah sistem perangkat lunak. Beberapa kategori *software metrics* yaitu:

1. Metrik perangkat lunak yang berhubungan dengan produk: pengukuran perangkat lunak yang didasarkan pada besarnya komponen utama dari perangkat lunak yaitu aplikasi atau programnya. Berbagai metrik perangkat lunak ini dibahas dengan lebih terperinci di Bab 2.

2. Metrik perangkat lunak yang berhubungan dengan kualitas: pengukuran perangkat lunak yang berkaitan dengan berbagai aspek kualitas dari perangkat lunak tersebut. Berbagai metrik perangkat lunak ini dibahas dengan lebih terperinci di Bab 3.
3. Metrik perangkat lunak yang berhubungan dengan proses: pengukuran perangkat lunak yang berdasarkan proses yang dilalui selama siklus hidupnya mulai dari pengembangan, pengujian, pemakaian dan pemeliharaan. Berbagai metrik perangkat lunak ini dibahas dengan lebih terperinci di Bab 4.
4. Metrik perangkat lunak yang berhubungan dengan orang: pengukuran perangkat lunak yang berdasarkan pada orang-orang yang terlibat pada proses pengembangan, pengujian, pemakaian, dan pemeliharaan perangkat lunak tersebut. Berbagai metrik perangkat lunak ini dibahas dengan lebih terperinci di Bab 5.

Penelitian di bidang pengukuran perangkat lunak terus berkembang sampai dengan sekarang. Jumlah metrik perangkat lunak terus bertambah untuk menjawab tantangan perkembangan perangkat lunak itu sendiri yang semakin besar dan kompleks dengan segala dinamikanya. Secara keseluruhan sekarang ini terdapat lebih dari 200 Metrik Perangkat Lunak dengan tujuan pengukuran yang berbeda beda (Meyer dkk., 2009).

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

BAB 2

Metrik Produk



A. Pendahuluan

Metrik Produk merupakan ukuran dari suatu produk yang dalam hal ini adalah sistem perangkat lunak itu sendiri. Meskipun produk perangkat lunak sebenarnya ada tiga komponen, yaitu: aplikasi/program, data, dan dokumentasi, fokus utama biasanya pada komponen aplikasi/program. Mengukur produk perangkat lunak dapat dilakukan dalam berbagai cara, antara lain:

1. **Metrik Ukuran:** mengukur perkembangan suatu sistem perangkat lunak berdasarkan perkembangan jumlah baris kode, fungsionalitas, atribut, metode, dan lain-lainnya yang menyusun komponen aplikasi dari perangkat lunak.
2. **Metrik Biaya dan Waktu:** mengukur kebutuhan biaya dan waktu yang diperlukan untuk menjalankan proyek pengembangan perangkat

lunak. Metrik biaya dan waktu ini sangat bermanfaat pada saat awal-awal proyek yang berfungsi memberikan estimasi biaya dan waktu yang diperlukan untuk menyelesaikan proyek perangkat lunak.

3. Metrik Bahasa Pemrograman Berorientasi Objek: untuk aplikasi/program yang dibuat dengan bahasa pemrograman berorientasi objek seperti C++, C#, Java, dan lain-lain terdapat beberapa metrik yang dikembangkan spesifik untuk bahasa pemrograman ini yang tidak dimiliki oleh bahasa pemrograman lainnya.
4. Metrik Kualitas: berbagai metrik yang dapat menggambarkan kualitas dari sistem perangkat lunak. Karena metrik kualitas ini cukup banyak dan sangat penting, maka metrik jenis ini akan dibahas secara tersendiri di Bab 3.

Berikut ini merupakan penjelasan yang lebih detail dan terperinci untuk ketiga jenis metrik yang disebutkan di atas.

B. Metrik Ukuran

Metrik ukuran merupakan metrik yang paling sederhana yang dapat menunjukkan tingkat perkembangan dari penyelesaian suatu perangkat lunak. Sebuah sistem perangkat lunak yang sederhana biasanya akan memiliki ukuran berupa jumlah baris kode yang kecil, sebaliknya sistem perangkat lunak yang besar biasanya akan memiliki ukuran berupa baris kode yang besar. Terdapat beberapa metrik ukuran perangkat lunak yang akan dibahas pada bab ini, yaitu:

1. Baris kode (*Lines of Code*): pengukuran berdasarkan jumlah baris kode yang telah dibuat oleh pemrogram.
2. Poin Fungsi (*Function Point*): pengukuran berdasarkan fungsionalitas yang terdapat pada setiap modul atau komponen program.
3. Jejak Memori (*Memory Footprint*): pengukuran berdasarkan jumlah memori yang dipakai saat sistem perangkat lunak sedang berjalan di komputer.
4. Ukuran Kode yang Terkompilasi (*Compiled Codes*): pengukuran berdasarkan ukuran dari berkas terkompilasi yang dihasilkan.
5. Dan lain-lainnya.

Berikut ini adalah penjelasan yang lebih lengkap dan detail mengenai setiap metrik ukuran yang disebutkan di atas.

1. Baris Kode atau Lines of Code (LOC)

Mengetahui ukuran dari sebuah sistem perangkat lunak adalah penting untuk membandingkan berbagai jenis sistem yang berbeda. Pengukuran yang paling sederhana adalah dengan mengukur jumlah baris kode yang disebut *Lines of Code* (LOC). Pengukuran dengan LOC ini pertama kali diusulkan oleh Harold D. Mills dari IBM di tahun 1970-an yang kemudian disempurnakan oleh Alan J. Perlis di tahun 1981 (Perlis dkk., 1981). Dengan menggunakan metrik ini, maka semakin besar sebuah perangkat lunak biasanya akan memiliki jumlah baris kode yang juga semakin besar. Ketentuan besar sebuah sistem perangkat lunak dapat ditentukan berdasarkan jumlah baris kode sebagai berikut.

- a. Sistem kecil: jumlah baris kode biasanya kurang dari seratus ribu baris kode atau kurang dari 100 KLOC (*Kilo Lines of Codes*).
- b. Sistem menengah: jumlah baris kode biasanya antara 100 KLOC sampai dengan 1 juta baris (*Million Lines of Codes*).
- c. Sistem besar: jumlah baris kode biasanya di atas satu juta baris sampai dengan 100 juta baris kode.
- d. Sistem sangat besar: jumlah baris kode biasanya melebihi 100 juta baris kode.

Terdapat beberapa variasi pengukuran berdasarkan jumlah baris kode, yaitu:

- a. *Lines*: Jumlah baris kode termasuk komentar, karakter kosong (*white space*), dan baris kosong.
- b. *Statements*: Jumlah statement. Dalam bahasa pemrograman Java statement diakhiri dengan tanda titik koma (;).
- c. *LOC (Lines of Code)*: jumlah baris kode dari sebuah program yang termasuk komentar, baris kosong, deklarasi, dan lain-lainnya. Untuk nilai yang besar biasanya dapat juga dinyatakan dalam satuan KLOC (*Kilo Lines of Code*) yang artinya jumlah dalam ribuan baris kode atau MLOC (*Million Lines of Code*) yang artinya dalam jutaan baris kode. Satu Kilo dinyatakan dalam ukuran 2^{10} atau 1024 baris.

- d. SLOC (*Source Lines of Code*): jumlah baris kode dari sebuah program dengan mengecualikan komentar dan baris kosong. Jumlah SLOC ini yang biasanya diperhitungkan untuk menentukan tingkat produktivitas seorang pemrogram.
- e. NCLOC (*Non-Comment Lines of Code*): Jumlah baris kode yang bukan komentar.

Sebagai ilustrasi tentang metrik ini, berikut ini beberapa contoh metrik ukuran menggunakan LOC untuk berbagai aplikasi atau program yang populer di dunia:

- a. Sistem Operasi Windows 11 terdiri dari 50 juta LOCs
- b. Sistem Operasi MacOS terdiri dari 84 juta LOCs.
- c. Sistem Operasi Android terdiri dari 15 juta LOCs
- d. Aplikasi Microsoft Word terdiri dari 45 juta LOCs.
- e. Dan lain-lainnya.

Meskipun menghitung baris kode merupakan metrik yang sederhana dan mudah dipahami, terdapat beberapa kelemahan dari cara pengukuran ini yang harus diperhatikan dan diwaspadai saat menerapkannya, yaitu:

- a. Sebuah sistem perangkat lunak yang sama apabila diprogram dengan 2 bahasa pemrograman yang berbeda akan menghasilkan baris kode yang juga berbeda. Sebuah program yang dibuat oleh bahasa pemrograman yang tingkatannya lebih tinggi biasanya akan dibutuhkan baris kode yang lebih sedikit.
- b. Sebuah sistem yang sama apabila diprogram oleh 2 orang yang berbeda tingkat kecakapannya, misalnya seorang pengembang junior dan senior juga akan menghasilkan baris kode yang berbeda meskipun diprogram dengan bahasa pemrograman yang sama. Seorang pengembang perangkat lunak yang lebih berpengalaman biasanya bisa membuat baris kode yang lebih efisien dan lebih singkat daripada seseorang yang kurang pengalaman.
- c. Sistem perangkat lunak modern sekarang ini banyak menggunakan pustaka (*library*) yang terdapat pada IDE misalnya *Visual Studio*, *Eclipse*, dan lain-lain yang tidak bisa dihitung langsung sebagai bagian dari kode program.

- d. Metode pengukuran ini tidak menghitung perangkat lunak yang dipakai ulang, duplikasi kode sumber, dan berbagai keuntungan lainnya dari desain ulang. Untuk pengembangan sistem perangkat lunak yang banyak menggunakan komponen dari sistem lain yang dipakai ulang, pengukuran baris kode tidak bisa menggambarkan upaya pengembangan yang tengah dilakukan.
- e. Semakin bertele-tele seorang pemrogram, akan terlihat lebih produktif karena mampu menghasilkan lebih banyak baris kode dibandingkan pemrogram lain, padahal pemrogram tersebut tidaklah lebih produktif daripada seorang pemrogram yang lebih tenang yang melakukan koding secara lebih efektif dan efisien.

2. Poin Fungsi atau Function Point

Poin Fungsi (*Function Point*) adalah metrik ukuran yang bersifat lebih objektif daripada *Lines of Code* karena sifatnya yang tidak tergantung dari bahasa pemrograman yang dipakai, namun lebih memperhatikan fungsionalitas dari modul atau komponen perangkat lunak. Metrik ini akan memperhitungkan jumlah fungsionalitas dari sebuah sistem sebagai tolok ukurnya. Metrik ini pertama kali diperkenalkan oleh Allan J. Albrecht dari IBM di tahun 1979 (Albrecht, 1979). Untuk menentukan nilai dari Poin Fungsi dari sebuah komponen dari sistem perangkat lunak, diperlukan 5 buah komponen yang harus dihitung, yaitu:

- a. Jumlah masukan eksternal: jumlah masukan atau *input* yang harus dikelola oleh modul tersebut. Semakin banyak masukan eksternal akan menunjukkan semakin kompleksnya program. Masukan eksternal ini dalam Poin Fungsi diberikan bobot 4.
- b. Jumlah keluaran eksternal: jumlah keluaran atau *output* yang harus dibuat oleh modul. Semakin banyak keluaran yang dihasilkan berarti semakin kompleks modul tersebut. Keluaran eksternal dalam Poin Fungsi diberikan bobot 5.
- c. Jumlah berkas logis yang dikelola: jumlah berkas atau *file* yang harus dikelola sendiri oleh modul, baik sebagai masukan ataupun keluaran. Semakin banyak berkas yang dikelola oleh sebuah modul berarti semakin kompleks modul tersebut. Jumlah berkas logis dalam Poin Fungsi berikan bobot 10.

- d. Jumlah berkas antarmuka eksternal: berkas yang diakses oleh aplikasi tetapi tidak dikelola sendiri di modul. Jumlah berkas antarmuka eksternal dalam Poin Fungsi diberikan bobot 7.
- e. Jumlah inkuiri eksternal: jumlah inkuiri ke dalam sistem basis data yang dikelola oleh modul. Jumlah inkuiri ini dalam Poin Fungsi diberikan bobot 4.

Adapun rumusannya adalah sebagai berikut.

$$FP = (\text{masukkan eksternal} \times 4) + (\text{luaran eksternal} \times 5) + (\text{berkas file logikal} \times 10) + (\text{berkas antar muka eksternal} \times 7) + (\text{inkuiri eksternal} \times 4)$$

Contoh:

Sebuah sistem perangkat lunak harus mengizinkan konsumen untuk melaporkan keluhan pada produk. Laporan ini akan disimpan dalam sebuah berkas dan tim pendukung akan mendapatkan laporan mengenai keluhan baru yang harus diselesaikan. Konsumen juga akan menerima laporan status tentang keluhan yang dilaporkannya. Pihak manajemen dapat meminta laporan rekapitulasi keluhan untuk bulan-bulan tertentu.

Dari informasi di atas dapat diketahui:

Input eksternal = 1 (melaporkan keluhan)

Output eksternal = 2 (laporan ke pengembang dan laporan ke konsumen)

Berkas logical internal = 1 (berkas penyimpanan laporan)

Inkuiri eksternal = 1 (manajemen meminta laporan rekapitulasi)

Berkas antarmuka eksternal = 0

Sehingga penghitungan Poin Fungsinya menjadi:

$$FP = (1 \times 4) + (2 \times 5) + (1 \times 10) + (0 \times 7) + (1 \times 4) = 28$$

Apabila nilai Poin Fungsi ini akan dikonversi menjadi jumlah baris (LOC), maka dapat diubah dengan melihat tabel konversi seperti pada Tabel 2.1 berikut ini.

Tabel 2.1 Konversi FP ke LOC (QSM, 2025)

Bahasa Pemrograman	Equivalency per Function Point			
	<i>Rata-rata</i>	<i>Median</i>	<i>Rendah</i>	<i>Tinggi</i>
C++	50	53	25	80
C#	54	59	29	70
Java	53	53	14	134
C	97	99	39	333
Assembler	119	98	25	320
HTML	34	40	14	48
JavaScript	47	53	31	63
Excel	209	191	131	315

Metrik Poin Fungsi dapat dipergunakan untuk menghitung ukuran sebuah sistem perangkat lunak, meskipun sistem tersebut belum selesai dibuat. Yang diperlukan adalah dokumen SKPL (Spesifikasi Kebutuhan Perangkat Lunak) yang sudah baik sehingga ukuran perangkat lunak dapat diperkirakan dengan baik. Setiap pernyataan kebutuhan dari sistem tersebut dihitung jumlah kelima komponen di atas dan kemudian dijumlahkan secara berbobot dengan bobot dari setiap komponen poin fungsinya, sedangkan untuk pengembangan perangkat lunak dengan pendekatan *Agile*, maka Poin Fungsi juga dapat diterapkan untuk menentukan ukuran pekerjaan dari setiap siklus yang akan dikerjakan. Penjelasan tentang proses pengembangan perangkat lunak dapat dilihat pada bagian Pendahuluan di Bab 5 yang membahas Metrik Proses.

Dalam sejarah perkembangannya, metrik Poin Fungsi ini mengalami beberapa perkembangan dan penyempurnaan, beberapa contoh di antaranya adalah:

- Ekstensi *General Sistem Characteristics* (GSC): pengembangan dengan adanya penambahan 14 pertanyaan yang berkaitan dengan fungsionalitas terdistribusi dan optimalisasi kinerja, dan lain-lainnya.
- 3D Function Poin*: pengembangan dari Poin Fungsi dengan memperhatikan aspek fungsional, teknis, dan juga manajemen proyek. Pendekatan ini lebih cocok untuk proyek pengembangan perangkat lunak modern seperti *Agile*, *DevOps*, dan lain-lainnya.

- c. *Object Points*: penyederhanaan dari Poin Fungsi dengan fokus pada objek. Pendekatan ini lebih cocok untuk pengembangan sistem perangkat lunak dengan sangat cepat (*Rapid Application Development*) ataupun dengan Purwarupa (*Prototyping*).
- d. *Feature Points*: pengembangan dari Poin Fungsi dengan memperhatikan aspek algoritma. Pendekatan ini lebih cocok untuk pengembangan aplikasi dengan algoritma intensif seperti aplikasi kecerdasan buatan, IoT (*Internet of Things*), aplikasi tertanam, dan lain-lainnya.

Dibandingkan dengan metrik ukuran Baris Kode atau *Lines of Code* (LOC), terdapat beberapa keuntungan dari pengukuran menggunakan Poin Fungsi atau *Function Point*, yaitu:

- a. Poin Fungsi dapat dipergunakan untuk menghitung setelah desain perangkat lunak selesai: metrik Poin Fungsi ini dapat segera dilakukan setelah dokumen SKPL selesai dan dapat memberikan estimasi ukuran dari sistem perangkat lunak yang akan dikembangkan.
- b. Poin Fungsi dapat dipergunakan untuk memperkirakan LOC tergantung dari tabel konversi FP ke LOC untuk bahasa pemrograman tertentu. Hubungan antara LOC dan FP dapat sangat bervariasi berdasarkan ilustrasi konversi yang ditunjukkan pada Tabel 2.1.

Sesuatu yang harus diingat pada saat menerapkan metrik Poin Fungsi ini adalah sifatnya yang cenderung subjektif. Poin Fungsi bersifat sangat subjektif karena sangat tergantung dari kapabilitas orang yang melakukan estimasi. Sebagai tambahan, estimasi dengan Poin Fungsi ini juga sangat sulit untuk diotomatis dibandingkan dengan pengukuran dengan LOC.

Organisasi yang mengelola berbagai macam pengukuran berdasarkan Poin Fungsi atau *Function Point* ini adalah IFPUG (*International Function Point User Group*) (<https://ifpug.org>). Organisasi ini menyarankan tiga jenis pengukuran perangkat lunak yaitu:

- a. *Function Point Analysis (FPA)*: kegiatan melakukan analisis dengan menggunakan Poin Fungsi atau *Function Point*.

- b. *Software Non-functional Assessment Process (SNAP)*: kegiatan untuk melakukan evaluasi berbagai aspek non-fungsional dari sistem perangkat lunak. Aspek non-fungsional ini biasanya berkaitan dengan kualitas perangkat lunak seperti yang dibahas di Bab 3.
- c. *Simple Function Poin (SFP)*: kegiatan melakukan analisis dengan versi sederhana dari Poin Fungsi atau *Function Point*.

Organisasi ini juga menawarkan keanggotaan dan juga berbagai sertifikasi yang berhubungan dengan ketiganya yaitu *Certified Function Point Specialist (CFPS)*, *Certified Function Point Practitioner (CFPP)*, *Certified SNAP Specialist (CSS)*, dan *Certified SNAP Practitioner (CSP)*.



Gambar 2.1 Situs Laman Web dari IFPUG

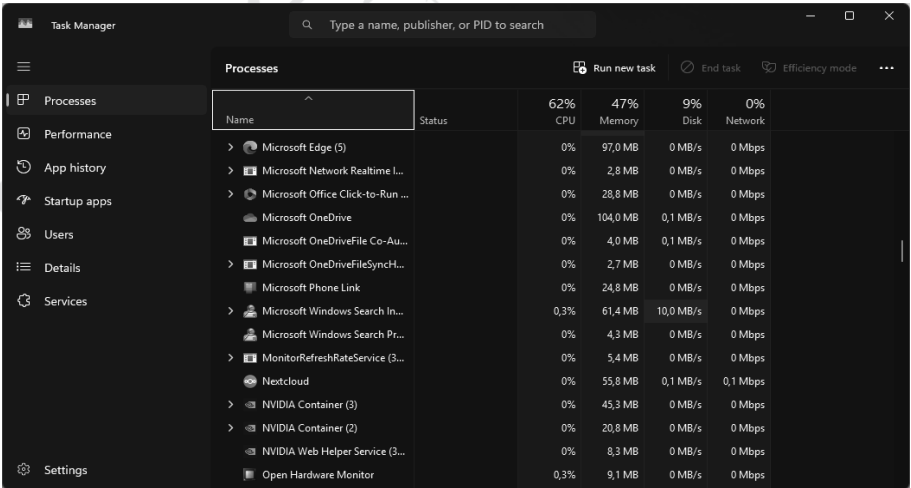
3. Jejak Memori atau Memory Footprint

Jejak Memori (*Memory Footprint*) adalah bentuk metrik ukuran yang difokuskan pada berapa besar memori pada RAM (*Random Access Memory*) yang terpakai saat aplikasi tersebut sedang berjalan di sebuah komputer. Apabila sebuah aplikasi memiliki jejak memori

yang kecil, artinya aplikasi tersebut dianggap ringkas dan dapat berjalan pada komputer yang memiliki RAM yang terbatas. Metrik ini dipakai untuk menentukan seberapa besar kode program dimuat dalam memori atau seberapa cepat kode program berjalan di komputer. Cara mengukurnya adalah dengan menghitung rata-rata jejak memori saat benar-benar dijalankan pada komputer.

Dengan semakin murahannya harga RAM dan semakin besarnya RAM yang terpasang pada sebuah komputer atau laptop, maka relevansi dari metrik ini menjadi semakin berkurang. Dengan besar RAM pada sebuah komputer atau laptop yang modern sekarang ini sebesar 8 GB atau lebih, maka besar jejak memori suatu aplikasi atau program menjadi kurang penting. Pengukuran ini sekarang lebih banyak untuk mengindikasikan kemampuan program tersebut untuk dijalankan pada komputer atau laptop yang lebih lawas. Semakin sebuah program dapat dijalankan pada komputer atau laptop yang lebih lawas, semakin besar tingkat kompatibilitas ke belakang yang dimiliki (*backward compatibility*) yang menandakan bahwa aplikasi tersebut dapat dipasang pada lebih banyak komputer atau laptop.

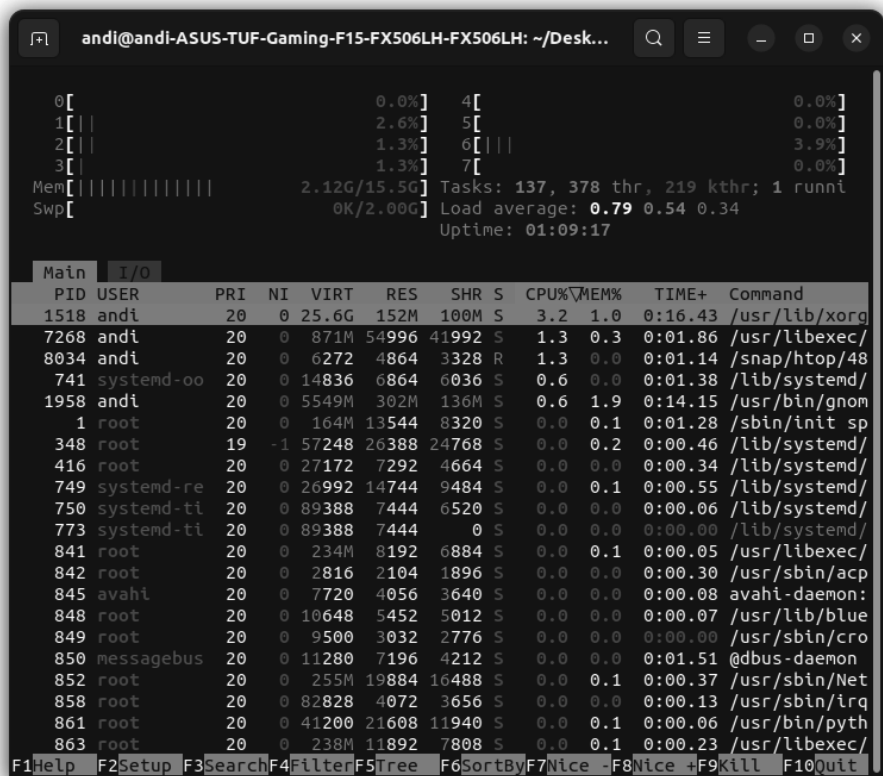
Untuk melihat besarnya jejak memori aplikasi-aplikasi yang sedang berjalan di Sistem Operasi Windows 11 dapat diketahui dengan menjalankan aplikasi *Task Manager* seperti yang ditunjukkan pada Gambar 2.2 berikut ini.



Processes					
Name		Status	62% CPU	47% Memory	9% Disk 0% Network
Microsoft Edge (5)			0%	97,0 MB	0 MB/s 0 Mbps
Microsoft Network Realtime I...			0%	2,8 MB	0 MB/s 0 Mbps
Microsoft Office Click-to-Run ...			0%	28,8 MB	0 MB/s 0 Mbps
Microsoft OneDrive			0%	104,0 MB	0,1 MB/s 0 Mbps
Microsoft OneDriveFile Co-Au...			0%	4,0 MB	0,1 MB/s 0 Mbps
Microsoft OneDriveFileSyncH...			0%	2,7 MB	0 MB/s 0 Mbps
Microsoft Phone Link			0%	24,8 MB	0 MB/s 0 Mbps
Microsoft Windows Search In...			0,3%	61,4 MB	10,0 MB/s 0 Mbps
Microsoft Windows Search Pr...			0%	4,3 MB	0 MB/s 0 Mbps
MonitorRefreshRateService (3...			0%	5,4 MB	0 MB/s 0 Mbps
Nextcloud			0%	55,8 MB	0,1 MB/s 0,1 Mbps
NVIDIA Container (3)			0%	45,3 MB	0 MB/s 0 Mbps
NVIDIA Container (2)			0%	20,8 MB	0 MB/s 0 Mbps
NVIDIA Web Helper Service (3...			0%	8,3 MB	0 MB/s 0 Mbps
Open Hardware Monitor			0,3%	9,1 MB	0 MB/s 0 Mbps

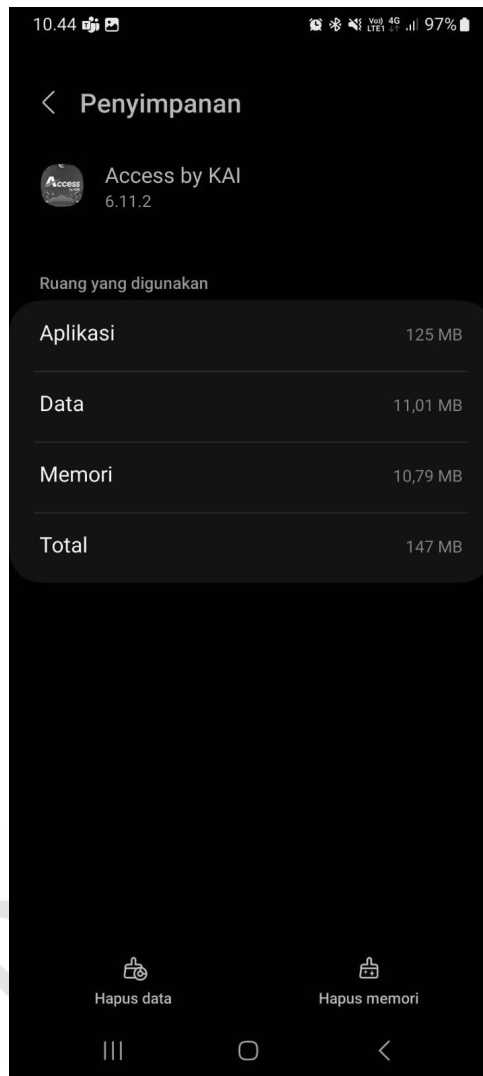
Gambar 2.2 Informasi Jejak Memori di Sistem Operasi Windows 11

Untuk Sistem Operasi Linux seperti Ubuntu, jejak memori dari berbagai aplikasi yang sedang berjalan dapat dilihat dengan mengetikkan perintah *top* atau *htop* di Terminal. Tampilan hasil perintah *htop* dapat ditunjukkan pada Gambar 2.3 berikut ini.



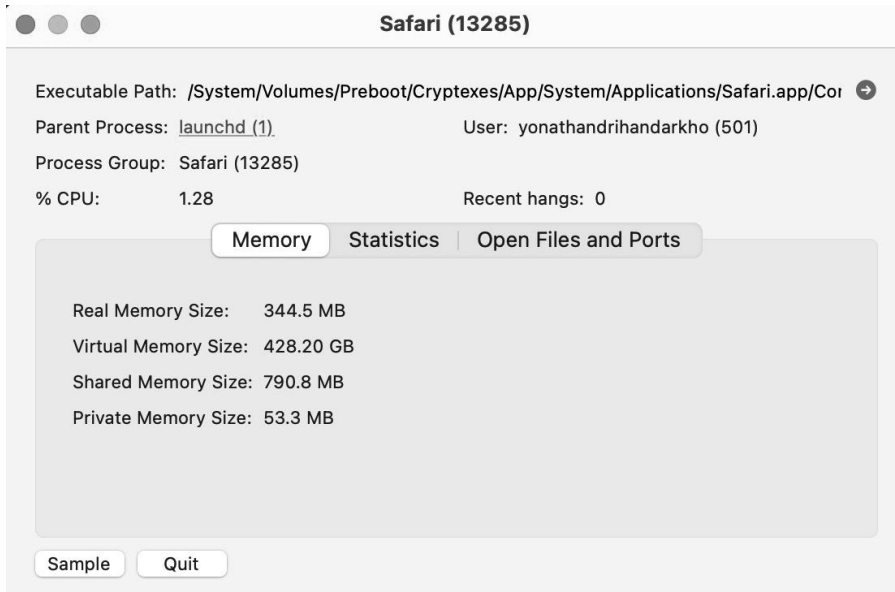
Gambar 2.3 Tampilan Jejak Memori Aplikasi di Ubuntu

Di handphone berbasis Android, jejak memori dari suatu aplikasi dapat dilihat dengan membuka Setting > Aplikasi > (Nama Aplikasi) > Penggunaan (klik Data Seluler) seperti yang terlihat pada gambar berikut ini misalnya untuk Aplikasi Access by KAI.



Gambar 2.4 Jejak Memori Aplikasi di Android

Untuk Sistem Operasi MacOS, jejak memori aplikasi dapat dilihat dari *Activity Monitor* dan memilih salah satu aplikasi seperti yang ditunjukkan pada Gambar 2.5 berikut ini untuk aplikasi browser *Safari*.



Gambar 2.5 Jejak Memori Aplikasi Browser Safari di MacOS

Acuan untuk jejak memori atau *memory footprint* adalah sebagai berikut: untuk aplikasi sederhana biasanya menggunakan memori sebesar 1–10 MB, sedangkan aplikasi Desktop akan menggunakan jejak memori sebesar 50–500 MB. Aplikasi *mobile* di Sistem Operasi Android ataupun iOS biasanya memakai jejak memori sebesar 10–200 MB, sedangkan aplikasi gim atau permainan akan memakai memori sebesar 500 MB sampai lebih dari 4 GB. Berbagai aplikasi yang berada di server biasanya menggunakan memori sebesar 100 MB sampai lebih dari 2 GB.

4. Metrik Ukuran Kode yang Terkompilasi atau Compiled Codes

Metrik ini dipakai untuk menentukan besar program saat nanti dimuat di disk komputer setelah terkompilasi. Sebuah program yang ditulis dengan bahasa pemrograman seperti C# atau Java perlu dikompilasi terlebih dahulu sebelum bisa dipergunakan dalam sebuah komputer. Hasil dari proses kompilasi ini berupa berkas tereksekusi dalam format .MSI, .EXE atau .COM di Sistem Operasi Windows 11, .APK di Sistem Operasi Android, maupun .APP atau .PKG di Sistem Operasi MacOS. Besarnya berkas terkompilasi inilah yang menjadi tolok ukur

dari sebuah program apakah merupakan aplikasi yang tergolong kecil, sedang, maupun besar.

Besar Kode yang Terkompilasi ini dapat menunjukkan kemudahan dalam distribusi dari sebuah sistem perangkat lunak. Sebuah aplikasi atau program yang memiliki besar kode yang terkompilasi kecil menandakan bahwa program tersebut ringan dan mudah diunduh meskipun dalam koneksi Internet yang kurang kencang. Kebalikannya apabila besar kode yang terkompilasi besar akan menandakan bahwa program atau aplikasi tersebut berat dan sulit untuk diunduh. Metrik ini tentu saja tidak berlaku untuk aplikasi atau program yang ditulis dalam bahasa skrip seperti PHP ataupun *Python* karena saat eksekusinya tidak melalui proses kompilasi terlebih dahulu.

Acuan untuk kode terkompilasi atau *compiled codes* adalah mirip dengan yang terdapat pada bagian jejak memori atau *memory footprint*. Aplikasi sederhana biasanya menghasilkan kode terkompilasi sebesar 1–10 MB, sedangkan aplikasi Desktop akan menghasilkan kode terkompilasi sebesar 50–500 MB. Aplikasi *mobile* di Sistem Operasi Android ataupun iOS biasanya memiliki kode terkompilasi sebesar 10–200 MB, sedangkan aplikasi gim atau permainan akan menghasilkan kode terkompilasi sebesar 500 MB sampai lebih dari 4 GB. Berbagai aplikasi yang berada di Server biasanya menghasilkan kode terkompilasi sebesar 100 MB sampai lebih dari 2 GB.

C. Metrik Usaha, Biaya dan Waktu

Dalam manajemen proyek perangkat lunak, salah satu area pengetahuan yang penting adalah manajemen biaya, waktu dan juga usaha yang perlu disediakan. Aspek biaya dan waktu merupakan dua dari 4 area pengetahuan yang paling penting selain manajemen ruang lingkup, manajemen waktu dan manajemen kualitas.

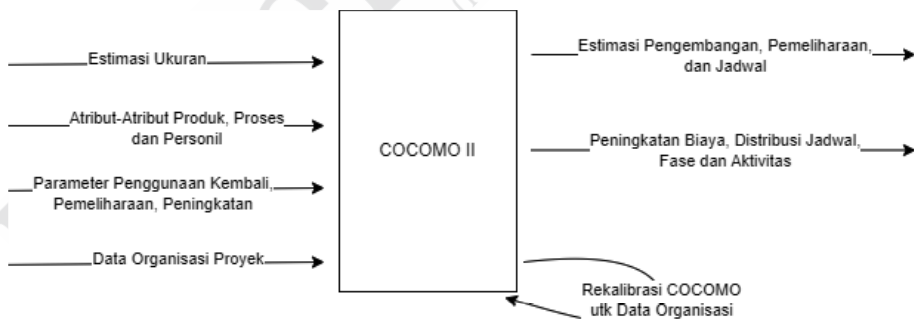
Salah satu tantangan terbesar dalam pengerjaan suatu sistem perangkat lunak adalah mengestimasi biaya yang diperlukan. Mengestimasi suatu biaya pengembangan sistem perangkat lunak biasanya diperlukan di tiga kesempatan waktu, yaitu:

1. Pada saat melakukan tender dari proyek perangkat lunak: estimasi biaya pada saat melakukan tender atau *bid* dari proyek perangkat lunak biasanya masih bersifat kasar namun harus cukup akurat

sehingga dapat memastikan bahwa tetap ada margin keuntungan yang didapat oleh perusahaan yang akan membuat perangkat lunak tersebut. Dilema yang dihadapi adalah apabila estimasi terlalu besar nantinya akan kalah dibandingkan dengan perusahaan kompetitor, sedangkan kalau estimasinya terlalu rendah akan menimbulkan risiko margin keuntungan yang sangat rendah dan bahkan malah menderita kerugian.

2. Pada saat awal proyek perangkat lunak tersebut dimulai: pada saat awal proyek pengembangan perangkat lunak, tingkat estimasi biaya harusnya sudah semakin akurat dan hampir mendekati biaya yang seharusnya dihabiskan.
3. Pada saat proyek perangkat lunak sedang berlangsung: pada saat proyek perangkat lunak berlangsung, proses pemantauan biaya dan juga penyempurnaan dari estimasi harus senantiasa dilakukan. Hasil ini bisa jadi tolok ukur di proyek-proyek perangkat lunak berikutnya.

Tingkat akurasi yang diharapkan untuk estimasi biaya perangkat lunak diharapkan semakin meningkat saat mendekati penyelesaiannya. Salah satu metode yang bisa dilakukan adalah dengan pendekatan COCOMO (*Constructive Cost Model*) II yang dipopulerkan oleh *Barry Boehm* dengan versi yang pertama di tahun 1981 dan kemudian berkembang menjadi versi kedua di tahun 1995 (Boehm dkk., 2000).



Gambar 2.6 Model Estimasi COCOMO II

Gambar 2.6 di atas menunjukkan estimasi dengan menggunakan pendekatan COCOMO II membutuhkan 4 buah masukan, yaitu:

- Estimasi ukuran yang bisa dilakukan dengan menggunakan Poin Fungsi.
- Nilai-nilai dari berbagai atribut yang berkaitan dengan produk, proses, dan personel dalam proyek perangkat lunak.
- Nilai-nilai dari berbagai parameter tentang tingkat penggunaan kembali perangkat lunak (*software reuse*), pemeliharaan, dan upaya peningkatan dari proyek perangkat lunak.
- Data-data yang berkaitan dengan organisasi yang mengerjakan proyek perangkat lunak.

Masukkan-masukkan tersebut kemudian dikalibrasi ulang untuk data-data organisasi berdasarkan informasi-informasi dari berbagai proyek yang telah terkumpul, kemudian menghasilkan 2 buah luaran penting, yaitu:

- Nilai-nilai estimasi dari pengembangan, pemeliharaan, dan jadwal dari proyek perangkat lunak.
- Nilai-nilai peningkatan biaya, distribusi jadwal, fase dan aktivitas dalam proyek perangkat lunak.

Adapun rumus dari COCOMO II dalam bentuk yang disederhanakan adalah sebagai berikut.

$$Usaha = A \times LOC^B \times rerata(EM)$$

Di mana:

Usaha = usaha yang diperlukan dalam satuan *Person-Month*

A = konstanta yang ditetapkan 2,94 untuk COCOMO II

LOC = jumlah baris kode

B = Faktor skala (misalnya untuk bahasa pemrograman Java adalah 1,1)

rerata(EM) = rata-rata pengali usaha (misalnya untuk bahasa pemrograman Java adalah 1,2)

Untuk menentukan waktu tinggal kita menentukan produktivitas dari tim pengembangan dalam satuan *Person-Month*. Nilai konstanta untuk B dan rerata (EM) kemudian dapat terus diperbaiki untuk mendapatkan nilai estimasi yang lebih akurat. Nilai konstanta yang ditemukan dapat

dipergunakan untuk melakukan estimasi usaha, waktu, dan biaya di proyek-proyek perangkat lunak selanjutnya.

Situs web untuk simulasi penghitungan dengan COCOMO II secara detail dapat diakses di <https://softwarecost.org/tools/COCOMO/>. Pada situs tersebut kita dapat memasukkan berbagai macam parameter yang diperlukan untuk penghitungan dengan COCOMO II dan akan dilakukan perhitungan berbagai macam luaran berdasarkan masukan yang diberikan, seperti yang ditunjukkan pada Gambar 2.7 berikut ini.

COCOMO II - Constructive Cost Model

Monte Carlo Risk: Auto Calculate:

Software Size Sizing Method:

SLOC	% Design Modified	% Code Modified	% Integration Required	Assessment and Assimilation (0% - 8%)	Software Understanding (0% - 50%)	Unfamiliarity (0-1)
New						
Reused	0	0				
Modified						

Software Scale Drivers

Precedentedness	Nominal	Architecture / Risk Resolution	Nominal	Process Maturity	Nominal
Development Flexibility	Nominal	Team Cohesion	Nominal		

Software Cost Drivers

Product	Personnel	Platform	Project
Required Software Reliability	Nominal	Analyst Capability	Nominal
Data Base Size	Nominal	Programmer Capability	Nominal
Product Complexity	Nominal	Personnel Continuity	Nominal
Developed for Reusability	Nominal	Application Experience	Nominal
Documentation Match to Lifecycle Needs	Nominal	Platform Experience	Nominal
		Language and Toolset Experience	Nominal
		Time Constraint	Nominal
		Storage Constraint	Nominal
		Platform Volatility	Nominal
		Use of Software Tools	Nominal
		Multisite Development	Nominal
		Required Development Schedule	Nominal

Gambar 2.7 Situs Simulasi COCOMO II

Perangkat yang bisa membantu mengestimasi nilai usaha dan biaya dengan COCOMO II adalah dengan menggunakan perangkat *SLOCCount*. Pembahasan mengenai aplikasi *SLOCCount* dapat dilihat pada Bab 6 di buku ini.

D. Metrik Bahasa Pemrograman Berorientasi Objek

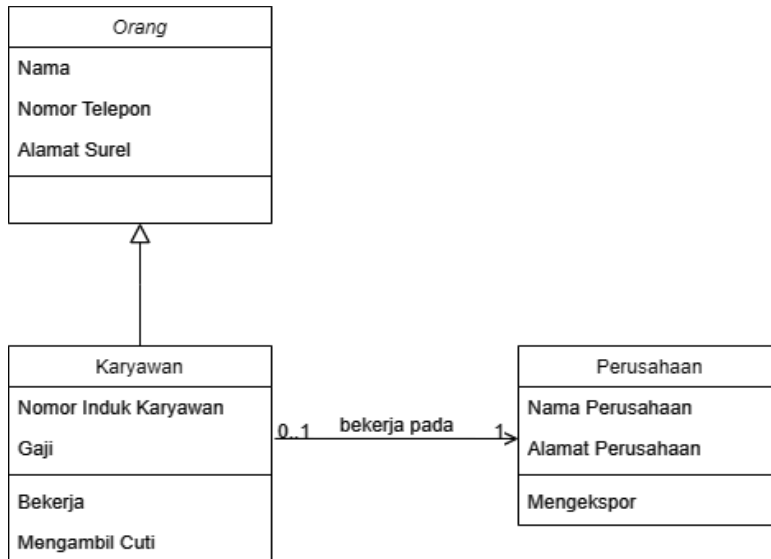
Bahasa pemrograman berorientasi objek sekarang ini masih menjadi bahasa pemrograman pilihan untuk mengembangkan berbagai sistem perangkat lunak. Bahasa pemrograman berorientasi objek pertama kali adalah bahasa C++ yang merupakan pengembangan bahasa pemrograman C. Dalam perkembangannya, muncul berbagai bahasa pemrograman berorientasi objek yang lebih modern untuk mempermudah bagi pemrogram dalam bekerja seperti bahasa C# yang dibuat oleh *Microsoft*, Java yang dibuat oleh *Sun Microsystems* (sekarang diakuisisi oleh *Oracle*), dan lain-lainnya. Metrik untuk perangkat lunak yang berorientasi objek pertama kali diperkenalkan oleh Chidamber dan Kemerer dari MIT (Chidamber & Kemerer, 1994).

Bahasa pemrograman berorientasi objek berdasarkan konsep bahwa semua entitas dalam sistem perangkat lunak harus diwakili dalam berbagai objek dan objek-objek tersebut dibuat dengan menggunakan kelas atau *class*. Kelas tersebut memiliki tiga komponen utama, yaitu:

1. Nama kelas: kelas akan memiliki nama yang membedakan kelas tersebut dengan kelas-kelas lainnya dalam sebuah sistem perangkat lunak. Objek akan dibuat dari kelas tersebut dengan sebuah proses yang disebut *instantiasi*.
2. Atribut-atribut: atribut merupakan berbagai macam variabel yang memberikan antara satu objek dengan objek yang lain dalam sebuah kelas.
3. Metode-metode: metode merupakan kegiatan atau aktivitas yang bisa dilakukan oleh sebuah objek terhadap objek lain, atau perlakuan objek lain terhadap sebuah objek, atau objek tersebut terhadap diri sendiri.

Sebuah kelas biasanya mampu melakukan tugas-tugas yang sangat sederhana dikarenakan kemampuan yang dimiliki sangat terbatas, tetapi kelas tersebut dapat berinteraksi dengan kelas-kelas yang lain untuk membuat fungsi yang lebih kompleks. Gambar 2.8 adalah ilustrasi sebuah diagram kelas yang melibatkan interaksi dari tiga buah kelas yaitu: *Orang*, *Karyawan*, dan *Perusahaan*. Kelas *Orang* memiliki tiga atribut yaitu *Nama*, *Nomor Telepon*, dan *Alamat Surel*. Untuk kelas *Karyawan* merupakan turunan dari kelas *Orang* dan memiliki atribut tambahan *Nomor Induk Karyawan* dan *Gaji* dan metode baru yaitu *Bekerja* dan *Mengambil Cuti*.

Kelas *Karyawan* juga berasosiasi dengan kelas *Perusahaan* dengan relasi *bekerja pada*, dan kelas *Perusahaan* tersebut memiliki atribut *Nama Perusahaan* dan *Alamat Perusahaan*, juga memiliki metode *Mengekspor*.



Gambar 2.8 Ilustrasi sebuah diagram kelas

Dalam bahasa pemrograman Java kelompok dari kelas-kelas yang membentuk tugas yang lebih kompleks tersebut dinamakan *Package*. *Package* dan kelas juga bisa membentuk *Package* yang lebih besar lagi sesuai dengan kebutuhan dari sistem perangkat lunak tersebut. Jadi sebuah sistem perangkat lunak adalah kumpulan dari serangkaian *Package* yang memiliki tugas masing-masing dan di dalam *Package* tersebut bisa berupa kelas ataupun *Package* lainnya. Interaksi antar-*Package* dalam sebuah aplikasi dapat digambarkan dalam sebuah *Package Diagram* seperti yang dapat dilihat pada Gambar 2.13.

Pemrograman berorientasi objek, seperti Java, C++, C#, dan lain-lain, memiliki beberapa karakteristik unik yang tidak dimiliki oleh bahasa pemrograman sebelumnya yang cenderung bersifat prosedural, yaitu:

1. Abstraksi (*Abstraction*): penyederhanaan suatu konsep dengan menyembunyikan detail dari pihak lain yang tidak berkepentingan.
2. Enkapsulasi (*Encapsulation*): pembungkusan atau penggabungan data dan metode menjadi satu objek.

3. Pewarisan (*Inheritance*): kemampuan untuk mewariskan atribut dan metode dari satu kelas ke kelas turunannya.
4. Penyembunyian Informasi (*Information Hiding*): menyembunyikan detail dari suatu informasi ke pihak atau objek lainnya.
5. Ketergantungan (*Coupling*): situasi di mana sebuah komponen perlu terhubung dengan komponen lainnya agar tetap bisa berfungsi dengan baik, hubungan ini dinamakan asosiasi.
6. Kohesi (*Cohesion*): kohesi dalam sebuah perangkat lunak menandakan seberapa solid isi dari sebuah kelas, yang ditunjukkan dengan seberapa besar metode-metode dalam sebuah kelas terhubung satu dengan yang lain.
7. Dan lain-lain

Untuk dapat merancang sistem perangkat lunak dengan bahasa pemrograman berorientasi objek, prinsip yang paling diutamakan adalah prinsip modularitas atau *modularity*. Tim pemrogram harus berupaya agar sistem perangkat lunak yang dibangunnya memiliki tingkat modularitas yang tinggi dengan cara memanfaatkan penyembunyian informasi (*information hiding*), meminimalkan ketergantungan (*coupling*), dan memaksimalkan kohesi (*cohesion*).

Untuk mendesain perangkat lunak berorientasi objek biasanya menggunakan pemodelan yang disebut UML atau *Unified Modelling Language*. UML ini terdiri dari berbagai diagram yang menjelaskan sistem yang sedang dibangun dengan berbagai sudut pandang, yaitu:

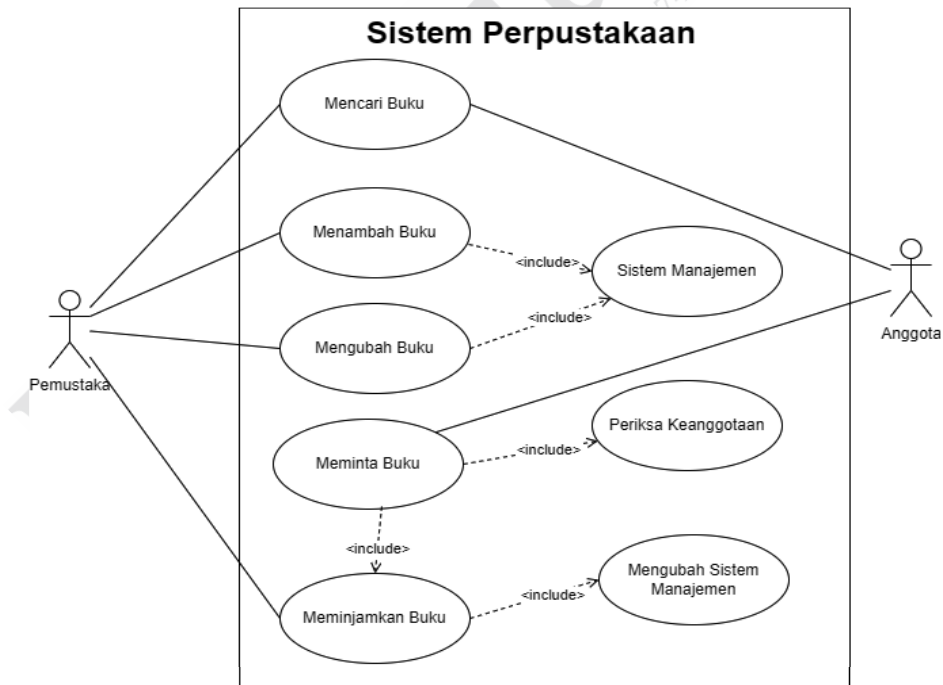
1. Diagram Struktural:

- a. *Class Diagram*: sebuah diagram yang menunjukkan asosiasi antar kelas yang ada di dalam sistem perangkat lunak yang akan dikembangkan. Objek-objek dalam aplikasi merupakan hasil perwujudan (disebut *instantiasi*) dari setiap kelas yang ada pada *Class Diagram*. Contoh *Class Diagram* ditunjukkan pada Gambar 2.8.
- b. *Object Diagram*: sebuah diagram yang menunjukkan situasi objek dalam waktu tertentu. Objek-objek tersebut merupakan hasil *instantiasi* dari kelas.
- c. *Component Diagram*: diagram yang memodelkan komponen-komponen perangkat lunak dan ketergantungannya satu dengan yang lain.

- d. *Deployment Diagram*: sebuah diagram yang menggambarkan arsitektur fisik (yang berupa perangkat keras dan perangkat lunak) pada saat sistem perangkat lunak tersebut diterapkan.
- e. *Package Diagram*: sebuah diagram yang menunjukkan asosiasi antar-*Package* dalam sistem perangkat lunak. *Package* itu sendiri merupakan kumpulan dari kelas-kelas yang memiliki tugas bersama menjalankan suatu fungsi di dalam aplikasi perangkat lunak. Contoh *Package Diagram* ditunjukkan pada Gambar 2.13.
- f. *Composite Structure Diagram*: sebuah diagram yang menunjukkan struktur internal dari sebuah kelas atau komponen.

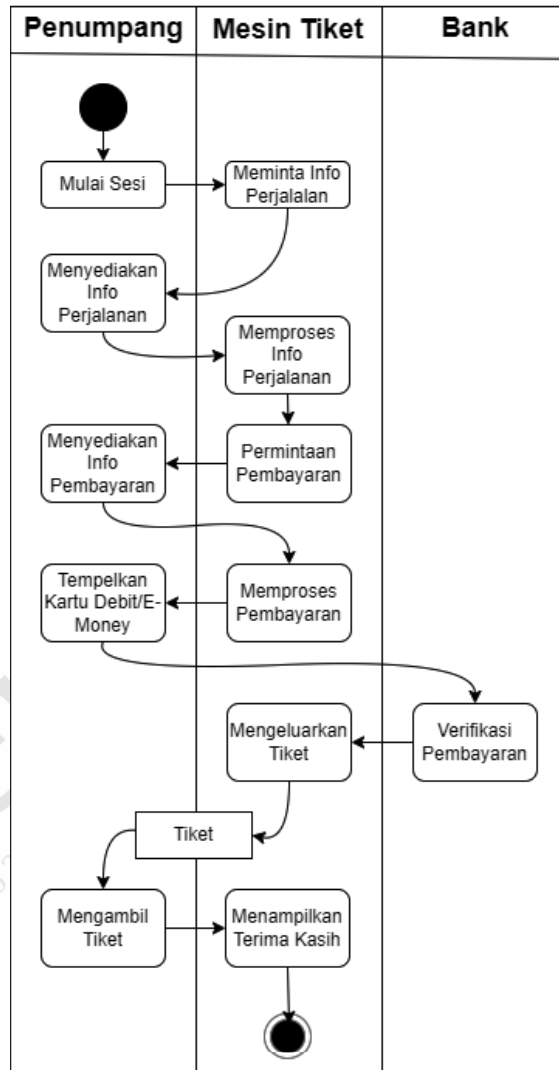
2. Diagram Perilaku:

- a. *Use Case Diagram*: sebuah diagram yang menunjukkan perilaku aplikasi dari sudut pandang pengguna atau sistem lain yang berinteraksi dengan sistem yang akan dibangun. Gambar 2.9 di bawah ini mencontohkan salah satu bentuk *Use Case Diagram* untuk sebuah sistem perpustakaan.



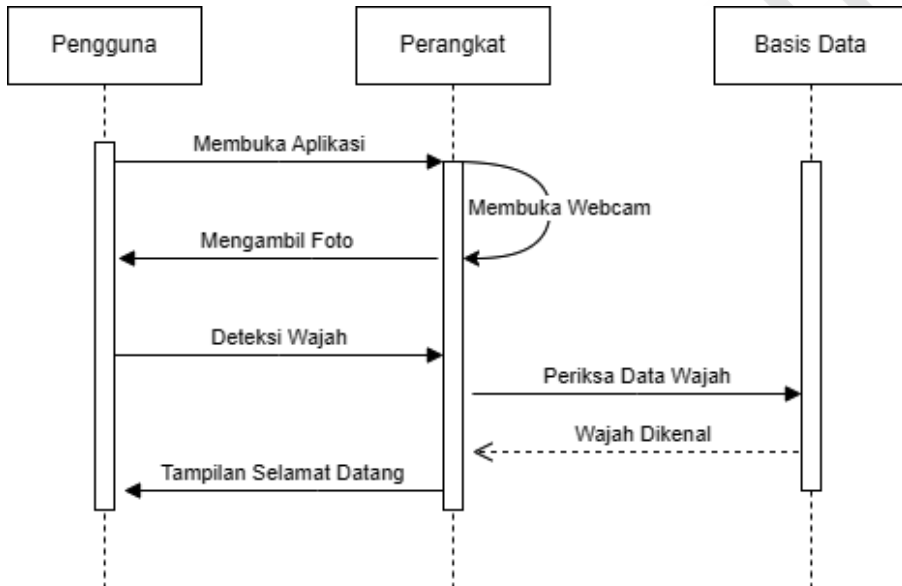
Gambar 2.9 Contoh *Use Case Diagram*

- b. *Activity Diagram*: sebuah diagram yang menunjukkan proses transisi antar aktivitas yang terjadi di setiap kasus penggunaan yang digambarkan di *Use Case Diagram*. Gambar 2.10 di bawah ini menunjukkan salah satu contoh *Activity Diagram* dari Kasus Pembelian Tiket pada suatu Sistem Dispenser Tiket Perjalanan Kereta Komuter.



Gambar 2.10 Contoh *Activity Diagram* Pembelian Tiket oleh Penumpang

- c. *State Machine Diagram*: sebuah diagram yang menunjukkan perubahan keadaan (*state*) dari suatu objek sebagai reaksi terhadap suatu kejadian atau *event*.
- d. *Sequence Diagram*: sebuah diagram yang menunjukkan transisi menurut urutan waktu dari objek-objek yang berinteraksi satu sama lain melalui metode-metodenya. Gambar 2.11 di bawah ini menunjukkan salah satu contoh *Sequence Diagram* untuk kasus penggunaan Pengenalan Wajah.



Gambar 2.11 Contoh Sequence Diagram Pengenalan Wajah

- e. *Collaboration Diagram*: sebuah diagram yang menggambarkan hubungan struktural antar-objek.
- f. *Timing Diagram*: sebuah diagram yang menggambarkan keadaan sebuah objek dan interaksinya sepanjang waktu.
- g. *Interaction Overview Diagram*: sebuah diagram yang merupakan kombinasi antara *Activity Diagram* dan *Sequence Diagram* untuk menjelaskan alur interaksi yang kompleks.

Dari sekian banyak diagram yang menjadikan *Unified Modelling Language*, terdapat 4 diagram yang dianggap paling penting, yaitu: *Use Case Diagram*, *Activity Diagram*, *Sequence Diagram*, dan *Class Diagram*. Keempat diagram tersebut dapat dianggap sebagai tipe diagram minimal yang harus dibuat untuk mendesain suatu sistem perangkat lunak yang berskala kecil sampai dengan medium. Untuk desain sistem perangkat lunak yang besar dan bahkan yang sangat kompleks, tentu saja jumlah diagramnya harus ditambah agar desainnya menjadi lebih detail dan sudut pandang yang lebih lengkap.

Untuk mendesain aplikasi dengan bahasa pemrograman berorientasi objek juga akan menggunakan pendekatan modularitas yaitu: memaksimalkan kohesi (*cohesion*) dan meminimalkan keterkaitan (*coupling*). Sistem yang tingkat modularitas tinggi akan mudah untuk dikembangkan dan dipelihara, sekaligus juga memiliki tingkat keberhasilan yang tinggi. Sebaliknya sistem yang tingkat modularitas rendah akan sangat sulit dipelihara dan memiliki peluang kegagalan yang tinggi.

Beberapa metrik dalam bahasa pemrograman berorientasi objek antara lain:

1. Metrik Jumlah Kelas atau Jumlah Berkas Header
2. Metrik Jumlah Atribut dalam Kelas
3. Metrik Jumlah Metode dalam Kelas
4. Metrik Kekurangan Kohesi dalam Metode
5. Metrik Tanggapan untuk Sebuah Kelas
6. Metrik Ketergantungan
7. Metrik Pewarisan
8. Dan lain-lain

Berikut ini adalah penjelasan yang lebih lengkap dan lebih detail mengenai setiap metrik yang disebutkan di atas.

1. Metrik Jumlah Kelas atau Jumlah Berkas Header

Salah satu metrik yang populer yang cara menghitungnya adalah dengan menghitung jumlah kelas-kelas (tidak termasuk *inner class*) ataupun jumlah berkas *header* (biasanya di akhir dengan akhir *.h* atau *.hpp*). Berkas kelas (biasanya berakhir dengan akhiran *.java* di bahasa pemrograman Java atau *.cpp* di bahasa pemrograman C++)

menandakan banyaknya kelas yang akan terlibat dalam sebuah sistem perangkat lunak, sedangkan berkas header biasanya berisi berbagai deklarasi fungsi, kelas, *macro*, dan struktur data.

*JumlahKelas = JumlahTotal Berkas Kelas
dalam Proyek Perangkat Lunak*

*JumlahHeader = JumlahTotal Berkas Header
dalam Proyek Perangkat Lunak*

Beberapa kelemahan dari metrik ini adalah sangat tergantung pada gaya pemrograman dari seseorang. Seorang pemrogram mungkin membuat sebuah kelas yang sangat besar sedangkan pemrogram lainnya bisa membaginya menjadi beberapa kelas yang lebih kecil. Untuk metrik jumlah kelas atau jumlah header, acuan yang dapat dipergunakan adalah sebagai berikut: sebuah proyek perangkat lunak yang kecil biasanya membutuhkan 5–20 berkas kelas dan header, sedangkan proyek perangkat lunak yang menengah membutuhkan 20–100 berkas kelas dan header. Untuk proyek perangkat lunak yang besar akan membutuhkan lebih dari 100 buah berkas kelas dan header untuk dapat bekerja dengan baik.

Sebagai ilustrasi jumlah kelas dan jumlah header di berbagai aplikasi populer adalah sebagai berikut. Aplikasi streaming lagu *Spotify* memiliki sekitar 300 kelas dan sekitar 200 berkas header, sedangkan aplikasi obrolan *Telegram* versi Desktop memiliki sekitar 800 kelas dan sekitar 500 berkas header. Aplikasi perkantoran seperti *LibreOffice* merupakan salah satu aplikasi yang sangat kompleks karena memiliki sekitar 10.000 kelas dan sekitar 5.000 berkas header.

2. Metrik Jumlah Atribut Dalam Kelas

Metrik yang nilainya ditentukan oleh jumlah atribut yang dimiliki dalam sebuah kelas, dengan mengecualikan konstanta dan metode. Metrik ini dipakai untuk menentukan tingkat kompleksitas sebuah data. Jika sebuah kelas memiliki jumlah atribut yang sedikit, maka

dapat dipastikan kompleksitas dari kelas tersebut adalah rendah. Sebaliknya, apabila jumlah atribut yang banyak, maka tingkat kompleksitas dari kelas tersebut juga tinggi.

$$\text{Jumlah Atribut dalam Kelas} = \text{Jumlah Variabel Lokal dalam Kelas}$$

Sebagai acuan untuk metrik ini adalah kalau sebuah kelas memiliki atribut 1–4 berarti termasuk kelas yang sederhana, sedangkan kelas yang memiliki atribut 5–8 termasuk kelas yang cukup kompleks yang kemungkinan memiliki nilai kohesi yang kurang baik. Apabila sebuah kelas memiliki lebih dari 9 atribut artinya kelas tersebut kemungkinan memiliki tugas yang terlalu banyak dan perlu untuk dipecah menjadi kelas-kelas yang lebih kecil.

3. Metrik Jumlah Metode Dalam Kelas

Metrik ini dipakai untuk menghitung seberapa besar sebuah kelas. Cara menghitungnya adalah dengan menghitung jumlah metode dari setiap kelas. Sebuah kelas dengan metode yang terlalu banyak cenderung akan mengerjakan tugas-tugas yang terlalu spesifik untuk aplikasi tertentu dan menghilangkan kemampuan kelas tersebut untuk dipergunakan ulang pada aplikasi-aplikasi lainnya. Adapun rumusnya adalah sebagai berikut.

$$\text{Jumlah Metode dalam Kelas} = \text{Jumlah Semua Metode di Kelas}$$

Jumlah baris kode di setiap metode juga bisa dijadikan tolak ukur tingkat efisiensi penulisan oleh tim pemrogram. Nilai rata-rata baris kode yang ideal di setiap metode adalah 20–30 LOC (Alsmadi, 2006). Jumlah baris kode yang terlalu banyak dalam sebuah metode menandakan metode tersebut mengerjakan terlalu banyak hal yang membuatnya tidak kohesif. Acuan yang biasanya dijadikan patokan adalah sebagai berikut.

- a. Jumlah metode 1–5 dianggap kelas yang sederhana dan dianggap baik.

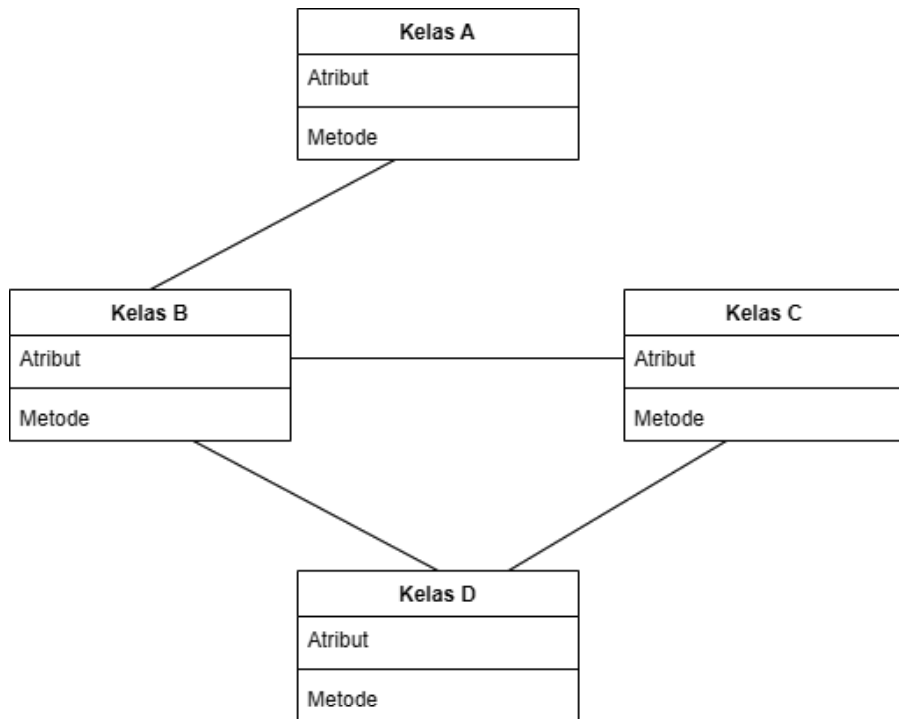
- b. Jumlah metode 6–10 dianggap kelas yang cukup kompleks yang nilai kohesi kelasnya juga kemungkinan tidak baik.
- c. Jumlah metode lebih 11 atau lebih menandakan sebuah kelas yang mengerjakan terlalu banyak tugas, disarankan untuk dipecah menjadi kelas-kelas yang lebih kecil.

4. Metrik Ketergantungan (Coupling)

Ketergantungan (*coupling*) adalah situasi di mana sebuah komponen perlu terhubung dengan komponen lainnya agar tetap bisa berfungsi dengan baik, hubungan ini dinamakan asosiasi. Dalam sebuah sistem perangkat lunak, komponen ini bisa di tingkat kelas, *Package*, ataupun sub-sistem. Metrik perangkat lunak yang mengukur ketergantungan ini lebih difokuskan pada asosiasi antar kelas dan *Package* saja. Ukuran metrik ini menunjukkan kekuatan asosiasi antar-entitas dalam sebuah sistem perangkat lunak. Asosiasi ini bisa berbentuk komunikasi antarkelas (melalui metode-metodenya) ataupun asosiasi dengan pewarisan.

Di tingkat kelas, ketergantungan sebuah kelas dengan kelas yang lain didefinisikan menjadi 3 tipe:

- a. *Fan-in*: jumlah kelas lain yang memberikan masukan ke sebuah kelas. *Fan-in* yang tinggi mengindikasikan kelas tersebut dibutuhkan oleh banyak kelas-kelas lain untuk menjalankan tugasnya masing-masing.
- b. *Fan-out*: jumlah luaran dari kelas yang diberikan kepada kelas-kelas yang lainnya. Semakin tinggi *fan-out* berarti kelas ini tergantung pada banyak kelas-kelas yang lain untuk menjalankan tugasnya.
- c. *Coupling Between Object*: metrik Kopling Antar-Objek ini mengukur seberapa banyak kelas atau antarmuka saling terhubung dengan kelas atau antarmuka yang lainnya, tetapi mengecualikan asosiasi pewarisan. Semakin tinggi nilai CBO mengindikasikan sebuah desain perangkat lunak yang kurang baik.

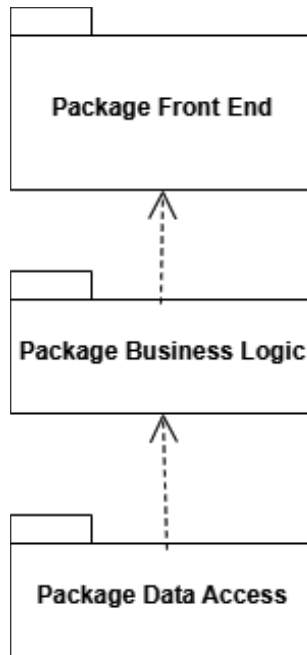


Gambar 2.12 *Coupling* Antar Kelas

Pada saat melakukan kegiatan koding sistem perangkat lunak, prinsip desain ideal yang dipakai adalah memaksimalkan *fan-in* dan meminimalkan *fan-out*. Sebuah modul atau kelas harus bermanfaat bagi atau sering dipakai oleh banyak kelas-kelas yang lain. Namun, kelas tersebut juga diharapkan tidak terlalu tergantung dengan kelas-kelas yang lainnya untuk bekerja.

Di tingkat *Package* atau modul, ketergantungan didefinisikan menjadi 2 tipe:

- Afferent Coupling* (C_A): jumlah *package* yang tergantung dari kelas-kelas di dalam *package* ini. Metrik ini mengindikasikan besarnya tanggung jawab *package*.
- Efferent Coupling* (C_E): jumlah *package* lain di mana *package* ini bergantung. Metrik ini mengindikasikan tingkat independensi dari *package*.



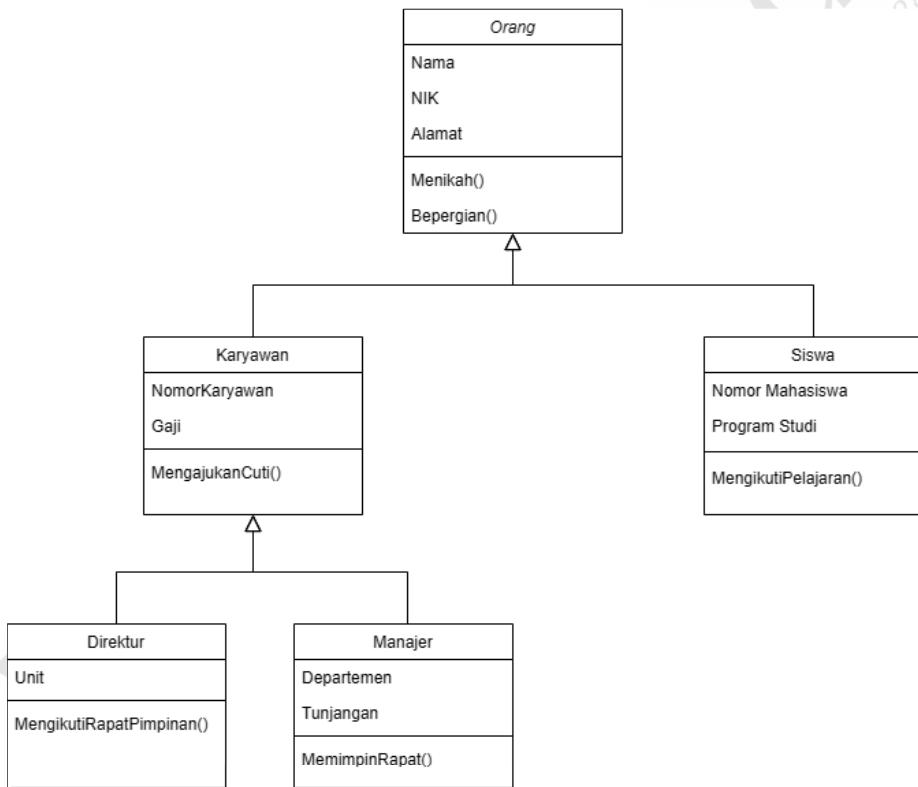
Gambar 2.13 Contoh Package Diagram

Pemahaman tentang *Afferent Coupling* dan *Efferent Coupling* adalah mirip dengan konsep *fan-in* dan *fan-out*, tetapi berada di tingkatan *Package* atau modul dan dalam tataran arsitektur perangkat lunak. Sebuah *Package* harus memiliki C_A yang tinggi yang artinya *Package* tersebut dipergunakan oleh *Package-Package* yang lainnya, sedangkan sebuah *Package* harus memiliki C_E yang rendah yang menunjukkan bahwa *Package* tersebut tidak terlalu tergantung pada *Package-Package* yang lain untuk dapat menjalankan tugasnya.

5. Metrik Pewarisan (Inheritance)

Salah satu ciri paling penting dari sebuah bahasa pemrograman berorientasi objek adalah kemampuan sebuah kelas untuk diwariskan menjadi kelas lain atau disebut juga pewarisan. Apabila sebuah kelas mewarisi dari kelas lain, maka semua atribut dan metodenya juga akan mewarisi dari kelas atasnya (disebut kelas *super*) dan kemudian bisa ditambahkan atribut-atribut dan metode-metode tambahan oleh pemrogram.

Keuntungan dari pewarisan ini adalah memberikan kemudahan bagi seorang pemrogram untuk membuat sebuah kelas yang memiliki kemiripan dengan kelas lain tanpa harus menulis ulang semuanya di kelas yang baru. Adapun konsekuensi dari sifat ini adalah memengaruhi tingkat kompleksitas dari sebuah perangkat lunak di mana semakin banyak pewarisan akan mengurangi tingkat kompleksitas dari perangkat lunak. Namun, tingkat pewarisan yang terlalu banyak juga akan menyulitkan tim pemeliharaan dalam memelihara sistem di masa pemakaian sistem perangkat lunak tersebut.



Gambar 2.14 Pewarisan dalam Kelas Diagram

Terdapat 2 metrik perangkat lunak yang berkaitan dengan pewarisan, yaitu:

- a. *Number of Children (NOC)*: metrik yang menghitung jumlah anak dari sebuah kelas super.

$$NOC = \text{Jumlah Anak dari Sebuah Kelas}$$

Semakin tinggi nilai NOC ini menandakan tingkat kepentingan dari kelas tersebut yang memengaruhi kelas-kelas anaknya. Perubahan pada kelas yang memiliki nilai NOC tinggi akan menjadi sangat berisiko karena akan langsung juga berpengaruh pada kelas-kelas turunannya. Pada Gambar 2.14, NOC untuk kelas orang adalah 2, sedangkan NOC untuk kelas karyawan adalah 1, dan NOC untuk kelas direktur, manajer, dan siswa adalah 0.

- b. Metrik Kedalaman Pewarisan (*Depth of Inheritance* atau *Depth of Inheritance Tree* atau *DIT*): metrik Kedalaman Pewarisan mengukur jenjang pewarisan dari setiap kelas dalam bahasa pemrograman berorientasi objek.

$$DIT = \text{Jumlah Tingkatan Pewarisan dari Sebuah Kelas}$$

Apabila sebuah kelas diturunkan menjadi kelas lain, maka memiliki nilai Kedalaman Pewarisan 1. Apabila turunannya lebih dari satu maka nilai metrik ini akan semakin meningkat. Tingginya nilai Kedalaman Pewarisan akan meningkatkan usaha dalam pemeliharaan dari sebuah sistem. Tingginya nilai ini juga mengindikasikan tingkat kompleksitas yang semakin tinggi yang peningkatan risiko kesalahan program. Pada Gambar 2.14, DIT untuk kelas Orang adalah 0, sedangkan DIT untuk kelas karyawan dan siswa adalah 1, dan DIT untuk kelas direktur dan manajer adalah 2.

6. Metrik Bobot Metode di Setiap Kelas (Weighted Method per Class atau WMC)

Metrik *Weighted Method per Class* (WMC) merupakan variasi dari metrik jumlah metode di atas dengan penambahan bobot di setiap metodenya tergantung tingkat kompleksitasnya. Setiap metode dari sebuah kelas dihitung tingkat kompleksitasnya (misalnya dengan Kompleksitas Siklomatik yang dibahas di Bab 3) dan kemudian dijumlahkan. Sebagai ilustrasi ada sebuah kelas yang memiliki n jumlah metode dengan masing-masing metode memiliki kompleksitas $c_1, c_2, \dots, c_n$. Maka nilai WMC dari kelas tersebut adalah sebagai berikut.

$$WMC = \sum_{i=1}^n C_i$$

Di mana:

n = jumlah metode dalam kelas

C_i = nilai kompleksitas dari metode i (misalnya dihitung dengan kompleksitas siklomatik).

Metrik ini bermanfaat untuk menentukan usaha pengembangan dan tingkat pemeliharaan dari sebuah kelas di mana semakin tinggi nilai WMC menandakan kelas yang kompleks yang mungkin perlu untuk dipecah menjadi kelas-kelas yang lebih sederhana. Sebagai contoh nilai yang baik untuk WMC adalah antara 20 – 40.

7. Metrik Tanggapan untuk Sebuah Kelas (Response for a Class atau RFC)

Set dari metode-metode yang mungkin dieksekusi sebagai tanggapan atau respons dari pesan yang diterima oleh objek di kelas tersebut. Metrik ini diusulkan oleh Chidamber dan Kemerer (Chidamber & Kemerer, 1994). Cara menghitung metrik ini adalah sebagai berikut.

$$R_s = \{M_i\} \cup \{R_i\}$$

Di mana:

M_i adalah semua metode di dalam kelas

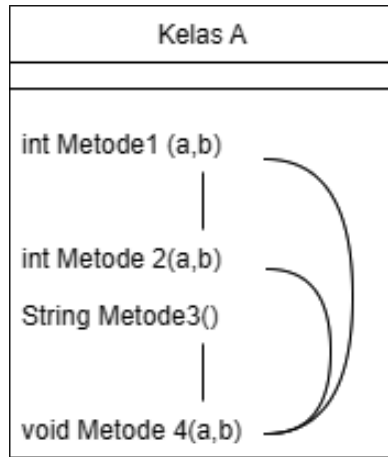
R_i adalah semua metode yang dipanggil oleh kelas tersebut.

Jika sejumlah besar metode dapat dipanggil sebagai tanggapan dari sebuah pesan, pengujian, atau kegiatan debug artinya kelas tersebut menjadi semakin kompleks. Apabila semua metode dipanggil untuk menanggapi, artinya objek dari kelas tersebut adalah kompleks yang akan memengaruhi pemahaman, pemeliharaan, dan tingkat pengujiannya.

8. Metrik Kekurangan Kohesi Dalam Metode versi 4 (Lack of Cohesion in Methods 4 atau LCOM4)

Kohesi dalam sebuah perangkat lunak menandakan seberapa solid isi dari sebuah kelas, yang ditunjukkan dengan seberapa besar metode-metode dalam sebuah kelas terhubung satu dengan yang lain. Sebuah kelas dikatakan solid apabila dibuat khusus untuk mencapai satu tugas sederhana saja dan tidak lebih. Apabila sebuah kelas memiliki atribut yang banyak dan metode yang banyak juga, maka biasanya kelas tersebut dibuat untuk mengerjakan lebih dari satu tugas, jadi sebenarnya kelas tersebut bisa dipisah menjadi 2 atau lebih kelas yang lebih sederhana. Metrik yang dipakai untuk mengukur tingkat kohesi disebut LCOM (*Lack of Cohesion in Methods*) dan yang paling mutakhir adalah metrik LCOM4.

Metrik *Lack of Cohesion in Methods* versi 4 merupakan versi peningkatan dibandingkan dengan metrik sebelumnya yaitu LCOM, LCOM2, dan LCOM3. Metrik ini diusulkan oleh Hitz dan Montazeri (Hitz & Montazeri, 1995). Versi orisinal dari *LCOM* diusulkan oleh Chidamber dan Kemerer (Chidamber & Kemerer, 1994). Nilai ideal dari LCOM4 adalah 1 yang berarti kelas tersebut hanya memiliki satu tugas sederhana saja, sedangkan nilai yang lebih dari satu menandakan bahwa kelas tersebut menjalankan lebih dari satu tugas dan sebaiknya dipecah menjadi kelas-kelas yang lebih sederhana.



Gambar 2.15 Ilustrasi Kohesi dalam Kelas

Gambar 2.15 di atas menunjukkan sebuah kelas yang kohesif dengan nilai LCOM4 adalah 1. Kelas tersebut memiliki 4 buah metode yang saling berkontribusi satu sama lain sehingga kelas tersebut memiliki nilai kohesif yang tinggi dan tidak dapat dipecah lagi menjadi kelas yang lebih kecil. Apabila metode-metode dalam sebuah kelas ternyata saling tidak berkomunikasi satu sama lain, maka kelas tersebut tidak kohesif dan kemungkinan besar dapat dipecah menjadi 2 atau lebih kelas yang lebih kecil.

BAB 3

Metrik Kualitas



A. Pendahuluan

Kualitas didefinisikan sebagai kemampuan dari suatu perangkat lunak untuk memenuhi atau bahkan melebihi ekspektasi dari penggunaanya. Kualitas adalah suatu nilai yang sulit untuk ditetapkan secara pasti dan bersifat multi-faset, karena kualitas suatu sistem perangkat lunak tergantung dari beberapa faktor:

1. Pemangku kepentingan (*stakeholder*): setiap pemangku kepentingan dari sistem perangkat lunak pasti memiliki harapan atau ekspektasi yang berbeda-beda tergantung peran dari pemangku kepentingan tersebut.
2. Tipe sistem perangkat lunak: sistem perangkat lunak yang berbeda memerlukan aspek kualitas yang berbeda untuk menentukan apakah sistem tersebut bagus atau tidak. Misalnya suatu aplikasi perbankan atau *mobile-banking* akan lebih menonjolkan aspek keamanan

sistem dibandingkan dengan yang lainnya, sedangkan untuk sebuah gim atau permainan akan lebih menekankan aspek kinerja, lebih lagi untuk gim edukasi untuk anak-anak akan lebih menekankan aspek kemudahan pemakaian.

3. Tipe pengguna: pengguna sistem yang berbeda juga memiliki ekspektasi kualitas yang berbeda. Pengguna yang tingkat lanjut (*advanced users*) biasanya menginginkan sistem yang bisa banyak dimodifikasi konfigurasinya dan berkinerja cepat, sedangkan pengguna awam cenderung lebih menekankan ke kemudahan penggunaan dan tidak terlalu banyak konfigurasi yang bisa diubah.

Sebuah sistem perangkat lunak tentu saja memiliki pemangku kepentingan (*stakeholder*) yang berbeda-beda dengan ekspektasi yang berbeda pula. Ekspektasi tersebut tentu saja berkaitan dengan peran dari setiap pemangku kepentingan tersebut mulai dari awal pengembangan sebuah perangkat lunak sampai dengan saat perangkat lunak tersebut sudah dipakai dan dijalankan. Beberapa contoh ekspektasi perangkat lunak berdasarkan pemangku kepentingan antara lain sebagai berikut.

1. Sponsor proyek perangkat lunak: sponsor proyek perangkat lunak tentu saja menginginkan sebuah sistem yang kompleks yang sesuai kebutuhan bisnisnya namun dengan biaya dan anggaran yang sehemat-hematnya. Karena sponsor ini merupakan pihak yang mengeluarkan biaya pembuatan sistem perangkat lunak, faktor waktu dan ruang lingkup diharapkan maksimal dengan setiap rupiah yang dikeluarkan.
2. Tim pengembang perangkat lunak: tim pengembang perangkat lunak pasti menginginkan sebuah sistem perangkat lunak yang mudah dibuat sehingga mereka dapat menyelesaikannya dalam waktu singkat tanpa menguras energi dan pikiran yang berlebihan.
3. Tim penguji perangkat lunak: tim penguji perangkat lunak pasti menginginkan sebuah perangkat lunak yang bebas dari cacat dan mudah untuk diuji sehingga pekerjaan mereka menjadi mudah.
4. Pengguna perangkat lunak: para pengguna perangkat lunak tentu saja menginginkan sebuah perangkat lunak yang mudah dipergunakan, hampir tanpa cacat, dan meningkatkan produktivitas.

5. Tim pemelihara perangkat lunak: tim pemelihara perangkat lunak akan menginginkan sebuah sistem perangkat lunak yang bebas cacat dan minim pemeliharaan.

Semua keinginan di atas tentu saja tidak mungkin terpenuhi semua sehingga kualitas perangkat lunak biasanya merupakan hasil kompromi dari berbagai ekspektasi di atas. Seiring dengan pesatnya perkembangan ilmu pengetahuan dan teknologi, maka mengembangkan perangkat lunak berkualitas tinggi sekarang ini menghadapi berbagai macam tantangan karena tuntutan yang semakin tinggi dan ketat. Berikut ini beberapa faktor yang menyebabkan sebuah mengembangkan perangkat lunak menjadi sebuah proses yang rawan kesalahan.

1. Tekanan jadwal yang berlebihan pada tim pengembang: sponsor proyek perangkat lunak menginginkan sebuah sistem perangkat lunak yang semakin canggih dan multifungsi dengan waktu pengembangan yang singkat agar bisa lebih unggul dibandingkan pesaing. Ini akan memberikan tekanan jadwal yang mungkin berlebihan kepada tim pengembangan untuk segera mengeluarkan fitur-fitur baru secepat mungkin.
2. Pelatihan yang buruk atau kurangnya pengalaman pemrogram: tingkat pengalaman dan juga pelatihan akan berpengaruh pada kemampuan dari tim pengembang untuk membuat sistem perangkat lunak berkualitas tinggi.
3. Kebutuhan perangkat lunak yang semakin aneh dan bertambah dengan cepat: sekarang ini hampir semua perangkat lunak, khususnya perangkat lunak yang dikembangkan untuk handphone menjadi semakin besar dan mampu memiliki sangat banyak fitur dan kemampuan (*superapp*). Perkembangan ini tentu saja disebabkan karena keinginan perusahaan yang memakai perangkat lunak tersebut untuk terus mengembangkan bisnis yang dimiliki.
4. Tingkat kompleksitas dari perangkat lunak yang tinggi: beberapa sistem perangkat lunak yang akan dibangun memang memiliki tingkat kompleksitas yang tinggi sehingga rawan untuk memiliki banyak cacat. Tingkat kompleksitas biasanya diukur dengan metrik kompleksitas siklomatik dan dianjurkan tingkatnya tidak lebih dari angka 15.

Pengukuran kualitas dari suatu sistem perangkat lunak juga merupakan pekerjaan yang rumit dan kompleks. Tidak semua aspek kualitas dari perangkat lunak dapat diukur dengan mudah. Berikut ini adalah beberapa pengukuran yang berkaitan dengan kualitas sebuah sistem perangkat lunak.

1. Metrik Kinerja (*Performance*)
2. Metrik Keamanan (*Security*)
3. Metrik Kemudahan Penggunaan (*Usability*)
4. Metrik Keandalan (*Reliability*)
5. Metrik Pemeliharaan (*Maintainability*)

Berikut ini adalah penjelasan yang lebih detail dan mendalam untuk masing-masing metrik tersebut.

B. Metrik Kinerja (*Performance*)

Metrik Kinerja: metrik kinerja menggambarkan keseluruhan kinerja dari sistem perangkat lunak dalam menghasilkan luaran yang diinginkan oleh para penggunanya. Metrik ini wujudnya bisa bermacam-macam, tergantung dari sistem perangkat lunak yang akan diujikan, antara lain:

1. Keterlambatan atau *Latency*
2. Laju Aliran atau *Throughput*
3. Tingkat penggunaan atau utilisasi
4. Dan lain-lainnya

Berikut ini adalah penjelasan dari masing-masing metrik tersebut.

1. Keterlambatan atau *Latency*

Keterlambatan adalah perbedaan waktu antara suatu perintah untuk melakukan sesuatu fungsi atau kemampuan pada aplikasi dengan permulaan perintah tersebut ditanggapi oleh sebuah sistem perangkat lunak. Semakin rendah nilai keterlambatan dari suatu perangkat lunak, maka dapat dianggap perangkat lunak tersebut berkinerja baik.

$$\text{Latency} = \text{Waktu Respons Diberikan} - \text{Waktu Respons Diminta}$$

$$Keterlambatan = Waktu Respons - Waktu Perintah$$

Nilai keterlambatan ini juga bisa bervariasi tergantung dari jenis perangkat lunak dan juga jenis pengguna yang memakainya. Nilai-nilai yang biasanya dianggap masuk akal adalah:

- < 100 mili detik untuk aplikasi yang membutuhkan interaksi langsung dengan pengguna
- 1 – 2 detik merupakan batas psikologis dari hampir semua aplikasi lainnya.
- > 3 detik biasanya dianggap terlalu lama dan memberikan risiko kehilangan pengguna.

2. Laju Aliran atau Throughput

Salah satu contoh bentuk *throughput* misalnya pada sebuah sistem untuk mengelola transaksi biasanya metrik kinerja yang ditentukan dengan satuan transaksi/detik. Ini biasanya untuk menandakan kemampuan maksimum yang dapat dimiliki oleh sistem tersebut dalam melakukan pekerjaannya mendukung pengolahan transaksi. Semakin besar nilai transaksi/detik berarti sistemnya sangat bagus dan dapat melayani lebih banyak pelanggan.

$$Laju Aliran = \frac{Jumlah Transaksi atau Data yang Diproses}{Total Waktu}$$

$$Throughput = \frac{Jumlah Tugas yang Selesai}{Waktu Total}$$

Atau:

$$Throughput = \frac{Jumlah Data yang Terkirim}{Waktu Transmisi dalam Detik}$$

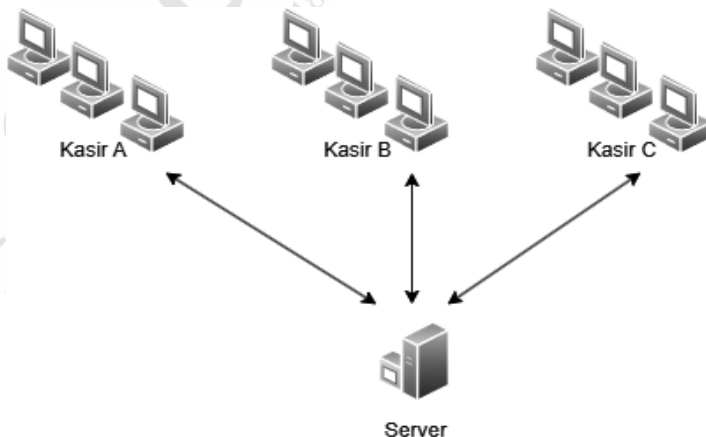
Atau:

$$\text{Throughput} = \frac{\text{Jumlah Permintaan yang Diproses}}{\text{Total Waktu Pengukuran}}$$

Contoh penerapan metrik ini misalnya pada suatu Sistem Pengolah Transaksi (*Transaction Processing System* atau TPS) yang biasanya dipakai pada suatu minimarket atau supermarket. Sistem ini biasanya berupa sistem dengan arsitektur *Client-Server* dengan ketentuan sebagai berikut.

- Transaksi dibangkitkan oleh salah satu komputer di kasir.
- Transaksi akan diproses di Server secepat mungkin.
- Jumlah pemrosesan per transaksi tidaklah besar.

Kinerja sistem ini biasanya diukur dengan kemampuan dalam mengolah transaksi selama satuan waktu tertentu. Pengolahan dari transaksi oleh Server biasanya dinyatakan dalam satuan *Operations (Ops)* atau *Kilo Operations (KOps)* per detik. Misalnya sebuah transaksi membutuhkan usaha 15 *KOps*, dan Server memiliki kemampuan 15.000 *KOps* setiap detik, maka kemampuan TPS tersebut adalah $(15.000/15) = 1.000$ Transaksi per detik. Gambar 3.1 di bawah ini menggambarkan contoh konfigurasi sebuah Sistem Pengolah Transaksi sederhana.



Gambar 3.1 Ilustrasi Sistem Pengolah Transaksi (TPS)

Terdapat pula bentuk pengukuran *Throughput* yang lain misalnya untuk aplikasi permainan atau gim, besarnya *throughput* diukur dari jumlah FPS (*Frame per Second*) yang dihasilkan oleh aplikasi tersebut pada layar monitor. Semakin tinggi nilai FPS untuk tingkat resolusi tertentu (biasanya minimal resolusi 1.080p), maka permainan tersebut akan terasa lebih mulus dalam berjalan dan lebih menyenangkan untuk dimainkan.

3. Utilisasi

Utilisasi mengukur tingkat kebergunaan suatu sistem perangkat lunak menggunakan berbagai sumber daya yang dimiliki untuk mencapai kemampuan maksimalnya. Semakin tinggi tingkat utilisasi dari suatu sistem perangkat lunak, maka perangkat lunak tersebut dianggap semakin berguna untuk penggunaannya. Sebaliknya, jika tingkat utilisasi rendah menandakan bahwa sistem tersebut banyak menganggur dan menghabiskan sumber daya misalnya listrik dengan sia-sia.

Unit atau satuan dari metrik utilisasi bisa beragam tergantung dari konteksnya. Sebagai contoh misalnya Sistem Pemrosesan Transaksi yang dijelaskan sesuai Gambar 3.1. Apabila sistem tersebut memiliki kemampuan 1.000 Transaksi/detik, dan setelah observasi ternyata rata-rata kenyataannya hanya melakukan 400 Transaksi/detik, maka utilisasi dari sistem TPS tersebut adalah:

$$\text{Utilisasi} = 400 / 1.000 = 40\%$$

Acuan untuk nilai utilisasi ini bisa beragam tergantung kasus yang dianalisis. Misalnya sebuah server yang sekarang memiliki tingkat utilisasi 40% alias sebagian besar kemampuannya tidak dipergunakan atau menganggur. Kemampuan yang tidak dipergunakan ini bisa dipergunakan untuk memperkirakan waktu di mana server sudah tidak mampu lagi. Misalnya terdapat peningkatan utilisasi server sebanyak 15% per tahun, maka dapat diperkirakan umur server adalah sekitar 60% / 15% atau sekitar 4 tahun lagi.

C. Metrik Keamanan (*Security*)

Keamanan dari sebuah sistem perangkat lunak merupakan hal yang esensial untuk diperhatikan, khususnya untuk sistem yang mengelola data pribadi penggunanya. Sistem perangkat lunak seperti sistem perbankan, sistem rumah sakit, sistem kependudukan, dan lain-lain biasanya menyimpan data-data pribadi dari penggunanya yang berpotensi untuk dicuri oleh pihak-pihak lain yang tidak bertanggung jawab. Untuk mencegah hal-hal yang tidak diinginkan tersebut, maka data-data pribadi tersebut harus dilindungi dengan aman. Beberapa metrik yang berhubungan dengan aspek keamanan sistem perangkat lunak antara lain: jumlah kerentanan yang terdeteksi, waktu rerata untuk mendeteksi, waktu rerata untuk menanggapi, laju insiden keamanan, dan waktu penyebaran dari tambalan.

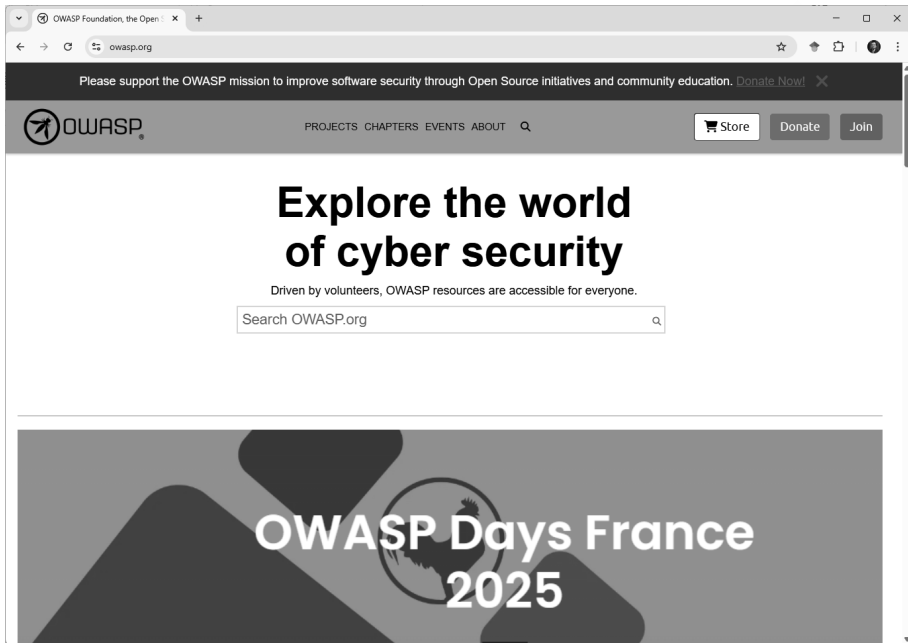
1. Jumlah Kerentanan (*Vulnerability*) yang Terdeteksi

Metrik perangkat lunak yang mengakumulasi jumlah kerentanan (*vulnerability*) yang berkaitan dengan masalah keamanan sistem perangkat lunak. Kerentanan dalam perangkat lunak artinya adalah kondisi di mana sistem memiliki kemungkinan untuk diserang oleh pihak-pihak yang tidak bertanggung jawab. Semakin banyak akumulasi jumlah kerentanan yang terdeteksi menandakan tingkat keamanan dari sistem perangkat lunak yang rendah.

Salah satu cara untuk mengetahui tingkat kerentanan adalah dengan menggunakan OWASP Top 10 yang dikeluarkan oleh yayasan OWASP (<https://owasp.org>). Setiap beberapa periode yayasan ini mengeluarkan 10 peringkat tertinggi (Top 10) yang berkaitan dengan risiko keamanan sistem misalnya di tahun 2017, 2021 dan juga di tahun 2025 ini. Berikut ini tingkat kerentanan sistem berdasarkan OWASP Top 10 di tahun 2021 (OWASP, 2025).

- a. *A01-2021 – Broken Access Control*
- b. *A02-2021 – Cryptographic Failures*
- c. *A03-2021 – Injection*
- d. *A04-2021 – Insecure Design*
- e. *A05-2021 – Security Misconfiguration*

- f. *A06-2021 – Vulnerable and Outdated Components*
- g. *A07-2021 – Identification and Authentication Failures*
- h. *A08-2021 – Software dan Data Integrity Failures*
- i. *A09-2021 – Security Logging and Monitoring Failures*
- j. *A10-2021 – Server-Side Request Forgery (SSRF)*



Gambar 3.2 Situs Laman Web Yayasan OWASP

Salah satu contoh studi kasus penerapan OWASP untuk perangkat lunak sistem akademik pada perguruan tinggi (Riberu dan Emanuel, 2024).

2. Waktu Rerata untuk Mendeteksi (Mean Time to Detect)

Metrik perangkat lunak yang mengukur rerata waktu dalam mendeteksi celah keamanan pada suatu sistem perangkat lunak. Sejak sistem perangkat lunak disebarkan untuk dipakai oleh pengguna, pencatatan waktu untuk mendeteksi berbagai masalah keamanan sistem perlu didata dan kemudian dihitung waktu reratanya.

$$MTTD = \frac{\text{Total Waktu Deteksi Semua Insiden}}{\text{Jumlah Insiden}}$$

Semakin rendah nilai MTTD, menandakan cepatnya tim keamanan untuk menanggapi insiden sehingga dampak kerusakan dari insiden tersebut dapat diminimalkan dan kerugian dapat ditekan.

3. Waktu Rerata untuk Menanggapi (Mean Time to Response)

Metrik perangkat lunak yang mengukur rerata waktu untuk menanggapi laporan dari celah keamanan pada suatu sistem perangkat lunak. Tingkat tanggapan yang dapat diterima oleh pengguna tergantung dari beberapa hal, khususnya pada tingkat potensi keparahan cacat yang ditimbulkan. Semakin tinggi potensi tingkat keparahan dari suatu cacat, maka tanggapan yang diharapkan akan semakin cepat.

$$MTTR = \frac{\text{Total Waktu Tanggapan untuk Semua Insiden}}{\text{Jumlah Insiden}}$$

4. Laju Insiden Keamanan (Security Incident Rate)

Metrik perangkat lunak yang mengukur laju dari laporan insiden keamanan dalam satuan waktu tertentu. Semakin tinggi nilai laju dari insiden keamanan berarti sistemnya masih belum stabil secara keamanan dan harus tetap dikawal sampai mencapai laju kestabilan tertentu.

$$SIR = \frac{\text{Jumlah Insiden Keamanan}}{\text{Total Waktu}}$$

Total waktu di sini dalam hitungan hari atau bulan atau tahun. Berikut ini contoh Laju Insiden Keamanan di Industri seperti yang ditunjukkan pada Tabel 3.1 berikut ini.

Tabel 3.1 Laju Insiden Keamanan di Berbagai Industri

Jenis Industri	Laju Insiden Per Tahun
Perbankan	Tinggi
Perdagangan Elektronik	Sedang
Kesehatan	Tren peningkatan
Pemerintahan	Variatif

Sumber: (IBM Security, 2023)

5. Waktu Penerapan Tambalan (Patch Deployment Time)

Metrik perangkat lunak yang mengukur waktu yang diperlukan untuk menambal celah keamanan. Waktu penambalan bisa bervariasi tergantung dari ukuran tambalan, jadi penempatan waktu di mana penambalan perlu dilakukan harus dipertimbangkan dengan matang.

$$PDT = Waktu\ Deploy\ Selesai - Waktu\ Tambalan\ Dirilis$$

Berikut ini standar industri untuk waktu penerapan tambalan yang ditunjukkan pada Tabel 3.2 berikut ini.

Tabel 3.2 Standar Industri untuk Waktu Penerapan Tambalan

Tingkat Keparahan	Target Waktu Penerapan
Kritikal	Kurang dari 24 – 72 jam
Tinggi	Kurang dari 1 – 2 minggu
Sedang	Kurang dari 1 bulan
Rendah	Jadwal rutin

Sumber: NIST, CIST Control, Microsoft Security Response Center)

D. Metrik Kebergunaan (Usability)

Beberapa metrik yang dapat dipergunakan untuk mengukur kebergunaan. Kebergunaan di sini diartikan sebagai sejauh mana sistem perangkat lunak tersebut dapat digunakan dengan mudah, efisien, dan memuaskan para penggunanya. Beberapa metrik yang dapat dipakai untuk mengukur kebergunaan antara lain:

1. Skala Kebergunaan Sistem (System Usability Scale atau SUS)

Sebuah metrik yang dipergunakan untuk mengukur tingkat kebergunaan sebuah sistem perangkat lunak dengan praktis dan cepat. Metrik ini pertama kali diperkenalkan oleh John Brooke di tahun 1986 (Brooke, 1986). Metrik ini menggunakan sepuluh buah pertanyaan yang dijawab oleh pengguna dengan memilih jawaban dalam skala *Likert* (1: Sangat Tidak Setuju, 2: Tidak Setuju, 3: Netral, 4: Setuju, dan 5: Sangat Setuju). Sepuluh pertanyaan tersebut adalah:

1. Saya pikir saya akan sering menggunakan sistem ini.
2. Saya menemukan bahwa sistem terlihat rumit.
3. Saya pikir sistem ini mudah dipergunakan.
4. Saya pikir saya akan membutuhkan dukungan dari personel teknis untuk menggunakan sistem ini.
5. Saya menemukan berbagai fungsi dalam sistem telah terintegrasi dengan baik.
6. Saya menemukan terlalu banyak ketidakkonsistenan di dalam sistem.
7. Saya membayangkan bahwa hampir semua orang akan belajar menggunakan sistem ini dengan cepat.
8. Saya menemukan bahwa sistem ini terlalu merepotkan.
9. Saya merasa percaya diri menggunakan sistem ini.
10. Saya perlu belajar banyak hal sebelum dapat menggunakan sistem ini.

Hasil penilaian dari kesepuluh pertanyaan tersebut adalah untuk pertanyaan yang bersifat positif yaitu nomor 1,3,5,7, dan 9 nilainya dikurangi satu, sedangkan untuk pertanyaan yang bersifat negatif yaitu nomor 2,4,6,8, dan 10 nilainya 5 dikurangi nilainya. Nilai total kemudian dikalikan dengan 2,5 untuk menghasilkan nilai total 100.

Adapun interpretasi dari nilai total adalah sebagai berikut:

- a. Skor di atas 80,3 menandakan kebergunaan sangat baik
- b. Skor antar 68 sampai 80,3 menandakan kebergunaan di atas rata-rata

- c. Skor 68 menandakan kebergunaan rata-rata
- d. Skor di bawah 68 menandakan perlunya perbaikan di sistem untuk aspek kebergunaan.

2. Kuesioner Pengalaman Pengguna (User Experience Questionnaire atau UEQ)

Sebuah metrik yang mengukur berbagai persepsi pengguna terhadap sebuah aplikasi. Metrik ini pertama kali diperkenalkan oleh Martin Schrepp, Andreas Hinderks, dan Jorg Thomaschewski di tahun 2014 (Schrepp dkk., 2015). Terdapat 6 skala utama dari pengalaman pengguna yang diukur, yaitu:

- a. Daya Tarik (*Attractiveness*)
- b. Kejelasan (*Perspicuity*)
- c. Efisiensi (*Efficiency*)
- d. Keandalan (*Dependability*)
- e. Stimulasi (*Stimulation*)
- f. Kebaruan (*Novelty*)

Terdapat 26 pertanyaan untuk menilai keenam skala di atas dengan penilaian dengan menggunakan 7 skala Likert untuk penilaian terburuk sampai dengan terbaik, yang terdiri dari (Schrepp dkk., 2015):

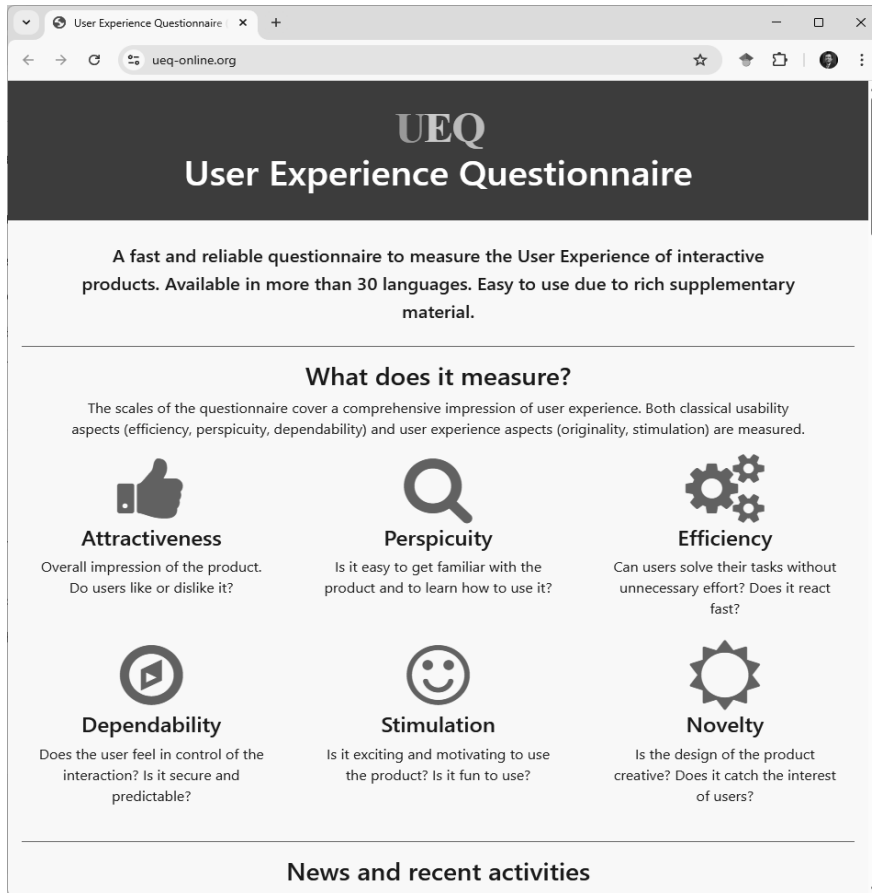
- a. Menyusahkan – Menyenangkan
- b. Tidak Dapat Dipahami – Dapat Dipahami
- c. Kreatif – Monoton
- d. Mudah Dipelajari – Sulit Dipelajari
- e. Bermanfaat – Kurang Bermanfaat
- f. Tidak Menarik – Menarik
- g. Tak Dapat Diprediksi – Dapat Diprediksi
- h. Cepat – Lambat
- i. Berdaya Cipta – Konvensional
- j. Menghalangi – Mendukung
- k. Baik – Buruk

- l. Rumit – Sederhana
- m. Tidak Disukai – Menggembirakan
- n. Lazim – Terdepan
- o. Tidak Nyaman – Nyaman
- p. Aman – Tidak Aman
- q. Memotivasi – Tidak Memotivasi
- r. Memenuhi Ekspektasi – Tidak Memenuhi Ekspektasi
- s. Tidak Efisien – Efisien
- t. Terorganisasi – Berantakan
- u. Atraktif – Tidak Atraktif
- v. Ramah Pengguna – Tidak Ramah Pengguna
- w. Konservatif - Inovatif

Terdapat pula versi pendek dari UEQ ini dengan hanya 8 pernyataan saja untuk keenam skala (Schrepp dkk., 2017), yaitu:

- a. Menghalangi – Mendukung
- b. Rumit – Sederhana
- c. Tidak Efisien – Efisien
- d. Membingungkan – Jelas
- e. Membosankan- Mengasyikkan
- f. Tidak Menarik – Menarik
- g. Konvensional – Berdaya Cipta
- h. Lazim – Terdepan

Situs untuk *User Experience Questionnaires* ini adalah di situs <https://ueq-online.org> seperti yang dapat dilihat pada Gambar 3.3 berikut ini.



Gambar 3.3 Laman Awal Situs Web UEQ-Online

Hasil masukkan pengguna dengan menggunakan UEQ ini dapat dijadikan acuan perbaikan dari suatu aplikasi, situs web, atau sistem perangkat lunak lainnya.

3. Laju Retensi (Retention Rate)

Sebuah metrik yang mengukur seberapa banyak pengguna aktif yang menggunakan suatu aplikasi dalam periode waktu tertentu, misalnya satu tahun. Metrik ini sangat penting untuk mengetahui bagaimana suatu aplikasi mampu mempertahankan penggunaannya dan mencegah pengguna untuk berhenti memakai aplikasi tersebut. Rumus dari metrik ini adalah:

$$\text{Laju Retensi} = \frac{(\text{Jumlah Pengguna Aktif di Akhir Periode})}{(\text{Jumlah Pengguna Aktif di Awal Periode})} \times 100\%$$

Sebagai contoh misalnya sebuah aplikasi perdagangan elektronik (e-commerce) mencatat pengguna aktif di awal tahun adalah 100 ribu orang, dan di akhir tahun pengguna aktifnya turun menjadi 90 ribu orang, maka laju retensinya adalah:

$$\text{Laju Retensi} = 90.000 / 100.000 = 90\%$$

Laju Retensi yang kurang dari 100% artinya tingkat pengguna aktif mengalami penurunan, sedangkan Laju Retensi yang di atas 100% menandakan pengguna aktif dari aplikasi tersebut mengalami peningkatan.

4. Laju Keberhasilan Tugas (Task Success Rate)

Metrik Laju Keberhasilan Tugas adalah metrik yang berfungsi mengukur persentase pengguna yang berhasil menjalankan tugas tertentu yang terdapat pada suatu aplikasi atau sistem perangkat lunak. Adapun rumus yang dapat dipergunakan adalah:

$$\begin{aligned} &\text{Laju Keberhasilan Tugas} \\ &= \frac{\text{Jumlah Pengguna yang Berhasil Menyelesaikan Tugas}}{\text{Total Pengguna yang Mencoba Tugas}} \times 100\% \end{aligned}$$

Nilai laju keberhasilan tugas dapat dipengaruhi beberapa faktor antara lain:

- Tingkat kemudahan atau kompleksitas yang dirancang untuk penyelesaian tugas pada aplikasi.
- Tingkat kemampuan pengguna sendiri dalam menyelesaikan tugas.
- Tingkat efektivitas desain yang dirancang pada aplikasi.

E. Metrik Keandalan (*Reliability*)

Keandalan dari sebuah sistem perangkat lunak mengacu pada kemampuan perangkat lunak tersebut untuk dapat berfungsi secara konsisten dan bebas dari kegagalan. Keandalan dari sebuah sistem perangkat lunak didukung oleh beberapa aspek penting, yaitu:

1. Kebenaran (*Correctness*)
2. Toleransi Terhadap Kegagalan (*Fault Tolerance*)
3. Kemampuan untuk Pulih (*Recoverability*)
4. Ketersediaan (*Availability*)
5. Ketangguhan (*Robustness*)

Beberapa metrik yang dapat dipergunakan untuk mengukur keandalan antara lain: laju kegagalan, waktu rerata untuk gagal, dan waktu rerata antara kegagalan, dan ketersediaan.

1. Laju Kegagalan (*Failure Rates*)

Metrik laju kegagalan akan mengukur jumlah kegagalan yang terjadi pada suatu rentang waktu tertentu. Rumus untuk menghitung laju kegagalan adalah sebagai berikut.

$$\lambda = \frac{R(t_1) - R(t_2)}{(t_2 - t_1) \times R(t_1)}$$

Di mana:

t_1 dan t_2 adalah awal dan akhir dari interval waktu tertentu

$R(t)$ adalah fungsi keandalan, yaitu probabilitas dari tidak adanya kegagalan sebelum waktu t

2. Waktu Rerata untuk Gagal (*Mean Time to Failure atau MTTF*)

Metrik ini akan mengukur rata-rata waktu suatu sistem perangkat lunak mengalami kegagalan untuk pertama kali. Adapun rumusnya adalah sebagai berikut.

$$MTTF = \frac{\text{Total Waktu Operasi}}{\text{Waktu Kegagalan}}$$

Semakin besar nilai MTTF menandakan sebuah sistem yang semakin bagus karena bisa bekerja lama sebelum gagal, sebaliknya kalau nilai MTTF-nya rendah artinya sistem tersebut akan gagal dalam waktu dekat. Satuan waktu yang dipakai di sini adalah jam.

3. Rerata Waktu Antar Kegagalan (Mean Time Between Failure atau MTBF)

Metrik ini akan mengukur waktu rata-rata sistem perangkat lunak bekerja dengan baik sampai menemui suatu kegagalan. Semakin besar nilai MTBF ini menandakan bahwa sistem perangkat lunak dapat bekerja dengan stabil dengan waktu yang cukup lama sebelum menemui kegagalan berikutnya, sebaliknya semakin rendah nilai MTBF ini menandakan bahwa sistem akan sering mengalami kegagalan dan kualitasnya rendah. MTBF berguna untuk aplikasi yang kritis secara keamanan. Adapun rumusnya adalah sebagai berikut.

$$MTBF = \frac{\text{Total Waktu Operasi}}{\text{Jumlah Kegagalan}}$$

Sebagai contoh misalnya dinyatakan sebuah sistem perangkat lunak memiliki MTBF sebesar 10.000 jam. Karena sistem tersebut dinyalakan selama 10 jam per hari, maka dapat diperkirakan kerusakan pertama dari sistem tersebut akan terjadi sekitar $10.000 / 10 = 1.000$ hari lagi.

4. Rerata Waktu untuk Perbaikan (Mean Time to Repair atau MTTR)

Metrik ini akan mengukur waktu rata-rata yang diperlukan oleh tim pendukung untuk dapat memperbaiki suatu kegagalan sampai sistem perangkat lunak tersebut dapat dipergunakan kembali.

$$MTTR = \frac{\text{Total Waktu Downtime}}{\text{Jumlah Kegagalan}}$$

Nilai yang diharapkan dari MTTR ini adalah sekecil mungkin, yang artinya tim pemeliharaan bekerja dengan sangat cepat untuk melakukan perbaikan saat sistem mengalami kegagalan.

5. Ketersediaan (Availability)

Ketersediaan merupakan suatu metrik yang mengukur seberapa sering sebuah sistem dapat dipakai oleh penggunaanya secara normal ketika memang benar-benar dibutuhkan. Adapun rumus dari ketersediaan adalah sebagai berikut.

$$\text{Ketersediaan} = \frac{MTBF}{MTBF + MTTR} \times 100\%$$

Nilai dari metrik ini biasanya berupa persentase dan dapat dinyatakan dalam dokumen SLA (*Service Level Agreement*) sebagai dokumen pengikat layanan yang resmi. Semakin tinggi nilai ketersediaan (semakin mendekati 100%), maka sistem dianggap memiliki ketersediaan tinggi. Sebagai contoh berbagai layanan komputasi awan misalnya AWS (*Amazon Web Services*) memiliki tingkat ketersediaan sebagai berikut.

- Layanan mesin virtual EC2 = 99,99%
- Layanan basis data relasional RDS = 99,95%
- Layanan penyimpanan S3 = 99,99%

F. Metrik Pemeliharaan (Maintainability)

Aspek kualitas pemeliharaan adalah aspek yang paling berperan selama penggunaan sebuah sistem perangkat lunak. Fase pemeliharaan akan berlangsung setelah sistem perangkat lunak diserahkan dari tim pengembang kepada pihak sponsor untuk dipergunakan. Pemeliharaan

biasanya meliputi kegiatan perbaikan apabila terdapat kesalahan yang ditemukan, dan juga penambahan atau penyempurnaan fitur apabila dibutuhkan.

Pengukuran aspek pemeliharaan dari suatu sistem perangkat lunak terdiri dari:

1. Metrik Kompleksitas Siklomatik
2. Halstead Complexity Measures
3. Metrik Jumlah Kondisi
4. Metrik Kepadatan Cacat
5. Metrik Indeks Pemeliharaan

Berikut penjelasannya.

1. Metrik Kompleksitas Siklomatik

Metrik perangkat lunak ini mengukur jumlah keputusan yang disebabkan oleh pernyataan kondisional di kode sumber (McCabe, 1976). Mengukur kompleksitas akan membantu kita dalam:

- a. Memprediksi upaya pengujian yang perlu dilakukan: semakin kompleks suatu modul atau fungsi perangkat lunak, semakin banyak kasus pengujian yang harus dibuat untuk dapat memastikan seluruh kemungkinan jalur dari modul atau fungsi tersebut bekerja dengan baik.
- b. Memprediksi jumlah kecacatan (*defect*): semakin kompleks suatu modul atau fungsi perangkat lunak, maka semakin besar juga peluang untuk memiliki cacat atau *defect*.
- c. Memprediksi biaya pemeliharaan sistem: semakin kompleks suatu modul atau fungsi perangkat lunak, maka antisipasi biaya untuk pemeliharaannya juga semakin tinggi.
- d. Dan lain-lain.

Tujuan utama dari metrik ini adalah untuk mengindikasikan kemampuan pengujian (*testability*) dan pemahaman (*understandability*). Metrik ini dikembangkan berdasarkan teori *graph*, di mana pengukuran jumlah jalur independen linier yang menyusun sebuah program.

Rumus dari kompleksitas siklomatik adalah:

$$M = V(G) = e - n - 2p$$

Di mana:

$V(G)$ = angka siklomatik dari Graph G

e = jumlah edge

n = jumlah node

p = jumlah dari bagian yang tidak tersambung dari graph

Persamaan ini sebetulnya sama dengan jumlah jalur yang berbeda pada sebuah program. Semakin banyak jalur yang berbeda, maka program tersebut dianggap semakin kompleks. Sebuah program atau modul dianggap memiliki kemampuan pengujian dan pemeliharaan yang baik apabila setiap modulnya memiliki nilai M tidak lebih dari 10. Nilai kompleksitas siklomatik yang terlalu tinggi misalnya di atas 15 menunjukkan tingkat kompleksitas yang tinggi dan seharusnya metode tersebut dapat dipecah menjadi metode-metode yang lebih sederhana.

Persamaan kedua dari kompleksitas siklomatik adalah:

$$V(G) = R$$

Di mana:

R = jumlah area tertutup dalam graph G

Persamaan ketiga adalah:

$$V(G) = P + 1$$

Di mana:

P = jumlah predikat atau kondisi

Ketiga rumus di atas akan memberikan hasil yang sama untuk kode sumber yang sama. Untuk acuan justifikasi nilai kompleksitas untuk sebuah modul adalah sebagai berikut: nilai kompleksitas 1 – 10 berarti kode yang sederhana dan mudah diuji, sedangkan nilai

11 – 20 berarti kode sumber memiliki kompleksitas sedang yang mungkin perlu disederhanakan. Untuk nilai kompleksitas siklomatik 21 – 50 berarti kode sumbernya kompleks dan memiliki risiko tinggi, terlebih nilai lebih dari 50 mengindikasikan kode sumber yang sangat kompleks dan sangat mungkin akan sangat sulit untuk dipelihara.

2. Halstead Complexity Measures

Metrik ini adalah metrik untuk mengukur tingkat kompleksitas suatu sistem perangkat lunak yang diusulkan oleh Maurice H. Halstead di tahun 1977 (Halstead, 1977). Terdapat dua komponen dasar dari penghitungan ini, yaitu:

- Operator*: simbol-simbol yang melakukan operasi seperti +, -, *, /, for, =, dan lain-lainnya.
- Operand*: variabel, konstanta, atau nilai literal lainnya.

Cara penghitungan dari *Halstead Complexity Measures* ini adalah sebagai berikut.

- Menghitung jumlah operator yang berbeda (duplikat operator hanya dihitung sekali saja) dalam variabel η_1
- Menghitung jumlah operand yang berbeda (duplikat operand hanya dihitung sekali saja) dalam variabel η_2
- Menghitung jumlah total operator (termasuk yang duplikat) dalam variabel N_1
- Menghitung jumlah total operand (termasuk yang duplikat) dalam variabel N_2
- Menghitung *vocabulary* dari program dengan rumus $\eta = \eta_1 + \eta_2$
- Menghitung panjang program dengan rumus $N = N_1 + N_2$

Dari informasi di atas, akhirnya dapat dihitung

$$\text{Estimasi Panjang Program} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

$$\text{Volume} = N \times \log_2$$

$$Tingkat\ Kesulitan = \frac{\eta_h}{2} \times \frac{N_2}{\eta_b}$$

$$Usaha = D \times V$$

3. Metrik Jumlah Kondisi (State)

Metrik ini dipakai untuk menghitung jumlah kondisi (*state*) yang mungkin dari sebuah sistem komputer, baik di tingkat objek ataupun modul yang lebih besar. Cara menghitungnya adalah dengan membuat model dari sistem perangkat lunak dan menghitung jumlah kondisi yang mungkin terjadi.

$$Jumlah\ Kondisi = \frac{Jumlah\ Total\ Kondisi\ yang\ Mungkin}{Jumlah\ Objek\ atau\ Modul}$$

Yang dimaksudkan dengan kondisi di sini adalah situasi yang mungkin terjadi dalam suatu sistem perangkat lunak. Sebagai contoh misalnya Sistem Operasi Windows 11 yang ada di laptop yang memiliki 4 kemungkinan keadaan: mati, hidup, tidur (*sleep*), dan tidur dalam (*hibernate*).

4. Metrik Kepadatan Cacat (Defect Density)

Sebuah metrik laju yang menggambarkan bagaimana cacat terjadi di setiap ukuran/fungsionalitas dari sebuah sistem perangkat lunak. Pengukuran bisa berdasarkan LOC atau FP. Adapun rumusnya adalah sebagai berikut.

$$Kepadatan\ Cacat = \frac{Jumlah\ Cacat}{Ukuran\ Sistem}$$

Jumlah kepadatan cacat yang tinggi biasanya menandakan bahwa sistem perangkat lunak tersebut masih belum siap untuk diserahkan kepada pihak klien. Apabila suatu perangkat lunak memiliki tingkat kepadatan cacat yang tinggi dipaksakan untuk diserahkan ke klien, maka dapat dipastikan tim pemeliharaan akan bekerja sangat keras saat sistem tersebut memasuki masa pemeliharaan karena harus cepat dan tanggap mengatasi berbagai cacat yang muncul.

5. Metrik Indeks Pemeliharaan (Maintainability Index)

Metrik Indeks Pemeliharaan atau Maintainability Indeks merupakan metrik yang mengindikasikan seberapa mudah suatu kode program dapat dipahami, dimodifikasi dan diperbaiki oleh tim pemelihara saat sistem perangkat lunak tersebut telah diserahkan kepada klien untuk dipergunakan. Metrik ini merupakan kombinasi dari berbagai metrik sebelumnya yaitu *Halstead Complexity Measures*, Kompleksitas Siklomatik, dan juga *Lines of Code*. Metrik ini diperkenalkan oleh Paul Oman dan Jack Hagemeister di tahun 1992 (Oman & Hagemeister, 1992).

$$MI = 171 - 5.2 \times \ln(V_{Halstead}) - 0.23(Kompleksitas Siklomatik) - 16.2 \times \ln(LOC)$$

Nilai yang diberikan pada kisaran 0 – 100 dengan semakin tingginya nilai MI menandakan sistem tersebut mudah untuk dipelihara. Nilai MI yang diinginkan adalah paling rendah 85, sedangkan nilai MI yang tidak diinginkan adalah di bawah nilai 65 yang menunjukkan bahwa sistem perangkat lunak sulit dipelihara dan perlu disederhanakan dengan kegiatan refactoring.

G. Metrik Portabilitas (Portability)

Metrik portabilitas atau *portability* adalah satuan pengukuran yang menandakan tingkat kemudahan penggantian platform atau Sistem Operasi. Sebuah sistem perangkat lunak dikatakan memiliki tingkat portabilitas tinggi apabila sistem tersebut dapat dijalankan di banyak

Sistem Operasi tanpa membutuhkan usaha yang ekstra dari tim pengembangnya. Untuk membuat sebuah aplikasi memiliki tingkat portabilitas tinggi, maka biasanya harus dikembangkan dengan bahasa pemrograman khusus misalnya menggunakan *Flutter* ataupun *Kotlin*.

Mengembangkan sebuah sistem perangkat lunak yang memiliki tingkat portabilitas tinggi memiliki beberapa kompromi yang harus diketahui. Sebuah sistem yang memiliki tingkat portabilitas tinggi biasanya tidak bisa memanfaatkan kemampuan maksimal dari Sistem Operasi di mana perangkat lunak tersebut dijalankan karena saat mengembangkannya harus juga mempertimbangkan bahwa kemampuan tersebut tidak akan dapat dijalankan di Sistem Operasi yang lainnya.

Cara mengukur tingkat portabilitas dari sebuah sistem perangkat lunak adalah dengan mengukur besarnya kode program yang dapat dipakai berulang di banyak platform. Semakin banyak potongan kode program yang dapat dipakai di banyak platform, maka dapat dianggap tingkat portabilitasnya tinggi. Adapun rumusnya adalah sebagai berikut.

$$\text{Portabilitas} = 1 - \frac{\text{Jumlah LOC yang Independen Terhadap Platform}}{\text{Total LOC}}$$

H. Metrik Waktu-Nyata (*Real-Time*)

Sebuah sistem perangkat lunak dapat dikatakan sebagai sebuah sistem waktu-nyata atau *real-time* apabila waktu luarannya dapat dijamin. Anggapan bahwa sistem ini harus berkinerja cepat sebenarnya tidak sepenuhnya benar. Sebuah sistem apabila waktu yang diperlukan untuk menghasilkan luaran dapat dijamin, maka sistem perangkat lunak tersebut dapat dikategorikan sistem perangkat lunak *real-time*.

Beberapa metrik yang dapat dipakai untuk mengukur sistem waktu nyata adalah:

1. Waktu Tanggapan (*Response Time*): waktu antara sebuah perintah diberikan sampai awal adanya tanggapan dari sistem. Untuk waktu tanggapan ini semakin kecil waktu tanggapannya dianggap semakin baik. Untuk sistem waktu nyata waktu tanggapan dihitung dalam satuan mili detik.

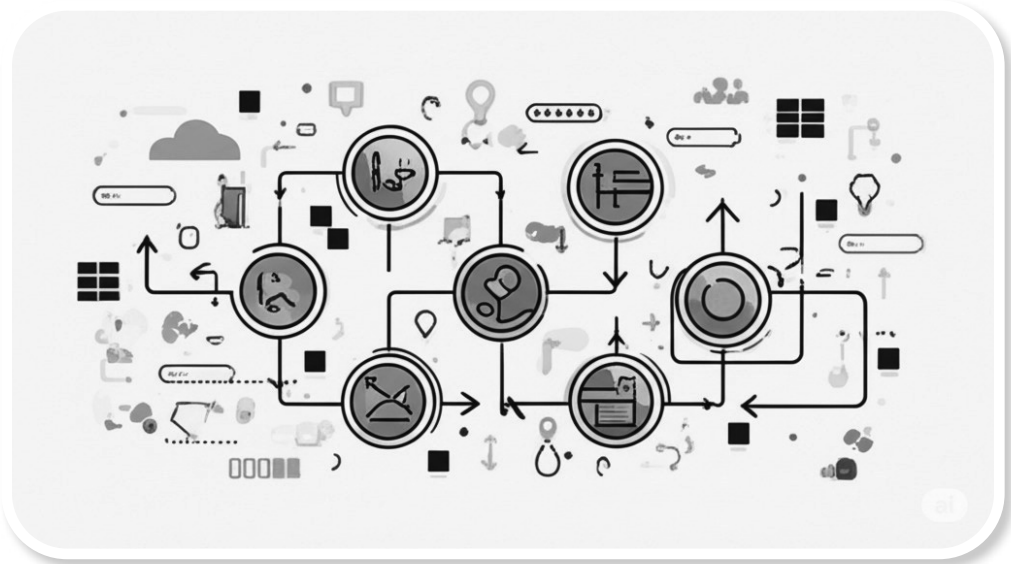
$$\text{WaktuTanggapan} = \text{WaktuTanggapanDiselesaikan} - \text{WaktuPerintahDiberikan}$$

2. Keterlambatan atau *Latency*: sudah dibahas sebagai bagian dari Metrik Kinerja.
3. Laju Kegagalan Tanggapan: nilai persentase yang menunjukkan tugas yang gagal melewati ambang waktu maksimum yang ditentukan.

$$\begin{aligned} & \text{LajuKegagalanTanggapan} \\ &= \frac{\text{Tugas yang Gagal Dijalankan Sesuai Target Waktu}}{\text{Total Semua Tugas yang Diberikan}} \times 100\% \end{aligned}$$

BAB 4

Metrik Proses



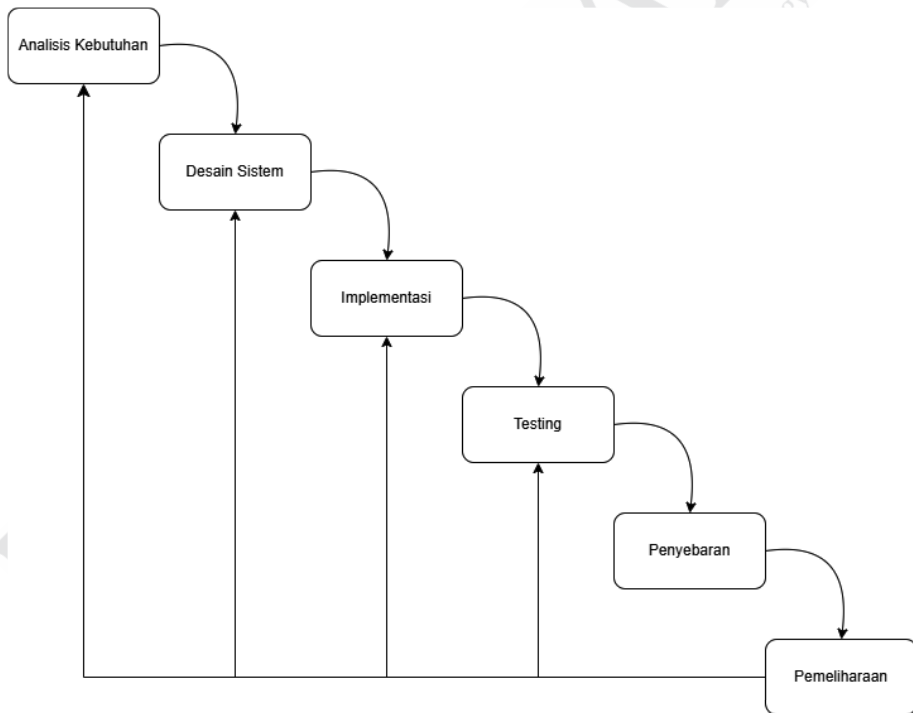
A. Pendahuluan

Proses merupakan rangkaian kegiatan yang menghasilkan produk, dan produk dalam hal ini adalah sebuah sistem perangkat lunak. Proses pengembangan perangkat lunak ini juga dikenal dengan istilah Siklus Hidup Pengembangan Perangkat Lunak atau *Software Development Life Cycle* (SDLC). Siklus Hidup Pengembangan Perangkat Lunak ini secara garis besar dibagi menjadi 2 bagian kategori besar, yaitu:

1. SDLC berbasis rencana: siklus pengembangan perangkat lunak yang mensyaratkan kelengkapan perencanaan sebelum mulai tahapan desain dan implementasi sebuah sistem perangkat lunak.
2. SDLC yang bersifat *Agile*: siklus pengembangan perangkat lunak yang bersifat pengembangan secara cepat dengan pengeluaran rilis secepat mungkin.

Siklus pengembangan perangkat lunak berbasis rencana membutuhkan perencanaan yang matang sebelum melangkah ke tahapan-tahapan berikutnya yaitu tahap desain, pengembangan dan pengujian. Beberapa contoh siklus pengembangan perangkat lunak ini antara lain:

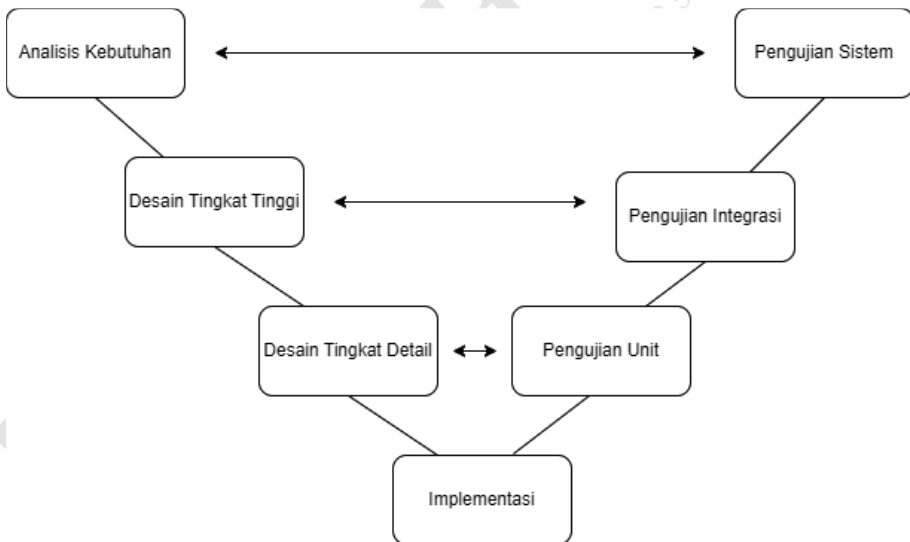
1. *Waterfall*: model pengembangan perangkat lunak *Waterfall* atau Air Terjun itu bersifat bertahap dengan akhir dari setiap tahapan didefinisikan secara pasti. Proses pengembangan tidak akan melangkah ke tahap berikutnya sebelum disepakati tahap yang sekarang telah dianggap selesai dengan indikator-indikator yang jelas. Adapun tahap pengembangan perangkat lunak dengan model *Waterfall* adalah: analisis kebutuhan, desain sistem, implementasi, testing, penyebaran, dan pemeliharaan. Gambar 4.1 menunjukkan model pengembangan *Waterfall* ini.



Gambar 4.1 Model Pengembangan Perangkat Lunak *Waterfall*

Model pengembangan perangkat lunak *Waterfall* biasanya berjangka waktu sekitar 3 – 4 bulan dan akan sangat cocok untuk pengembangan perangkat lunak berukuran kecil sampai menengah. Setiap tahapan harus dilalui secara bertahap satu per satu sampai dengan tahapan terakhir dan tidak boleh kembali ke tahapan sebelumnya sebelum semua tahapan yang harus dilalui diselesaikan. Model pengembangan perangkat lunak ini sangat cocok untuk mengembangkan perangkat lunak yang kebutuhannya sudah final dan tidak akan berubah dalam waktu dekat. Semua kebutuhan harus dituliskan dan disepakati secara keseluruhan sebelum melangkah ke tahapan desain dan seterusnya.

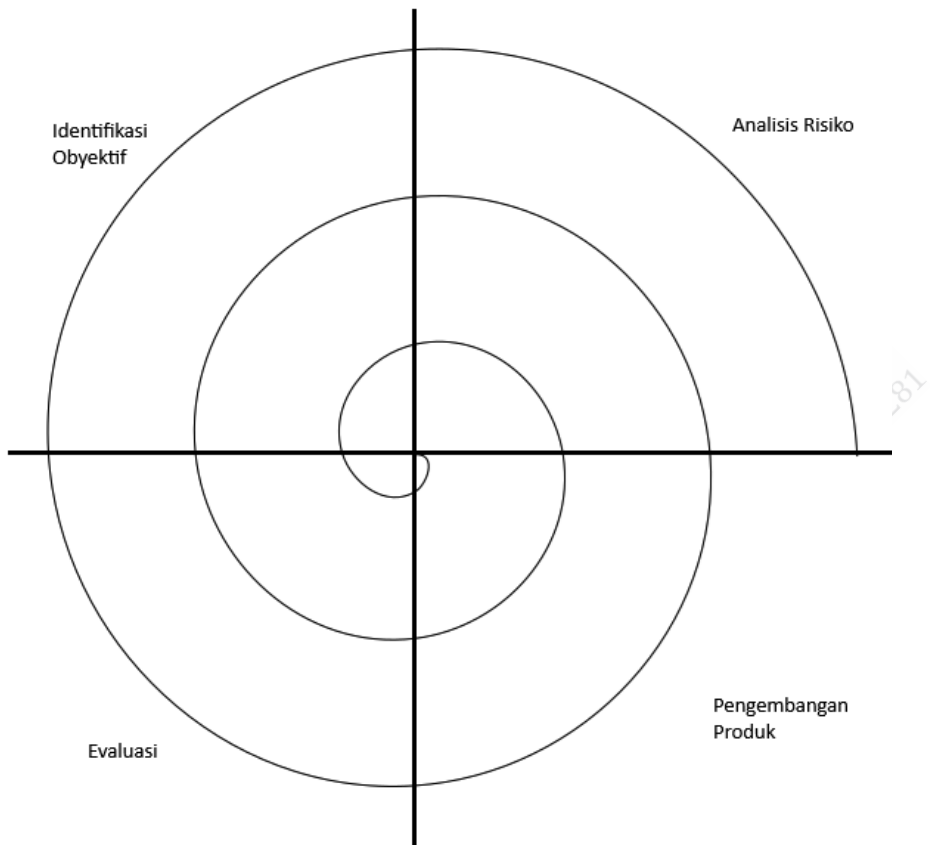
2. *Model-V*: Model-V merupakan proses pengembangan perangkat lunak yang mirip dengan *Waterfall*, tetapi dengan tahapan pendamping yaitu proses verifikasi dan validasi yang ketat di setiap tahapannya. Tahapan dalam siklus pengembangan perangkat lunak Model-V ini ditunjukkan pada Gambar 4.2 berikut ini.



Gambar 4.2 Pengembangan Perangkat Lunak Model – V

Pada siklus pengembangan Model-V, terdapat proses pengujian yang mendampingi setiap tahapan proses pengembangan perangkat lunak, yaitu:

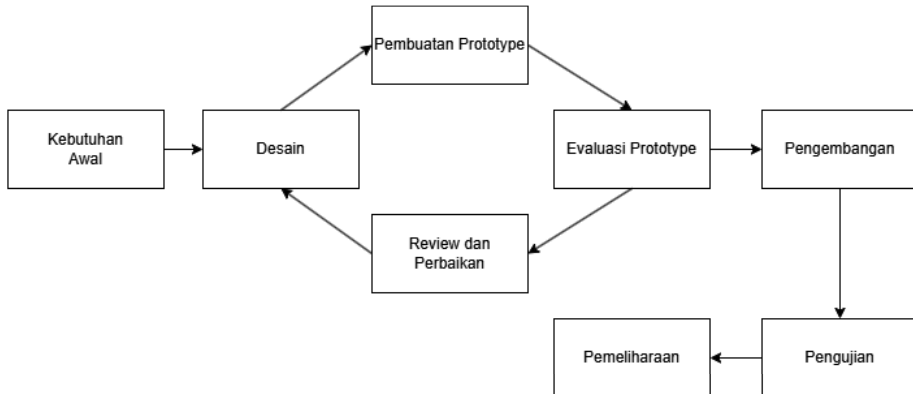
- a. Pengujian Unit (*Unit Testing*): pengujian di setiap modul yang dibuat oleh tim pengembang. Proses pengujian bisa secara manual ataupun dengan perangkat khusus seperti JUnit untuk bahasa pemrograman Java.
 - b. Pengujian Integrasi (*Integration Testing*): pengujian integrasi dari setiap modul menjadi fungsi yang lebih besar. Biasanya integrasi ini dalam tingkatan sub-sistem.
 - c. Pengujian Sistem (*System Testing*): pengujian sistem secara keseluruhan untuk memastikan semua fungsi bisa bekerja sebagaimana mestinya. Tahapan ini dilakukan sebelum Pengujian Penerimaan oleh Pengguna (*User Acceptance Testing*).
3. *Spiral*: pengembangan perangkat lunak spiral sebenarnya mirip dengan pengembangan secara *Waterfall*, tetapi diulang-ulang sampai beberapa kali sampai semua fitur yang diinginkan telah dikerjakan. Salah satu ciri dari sistem pengembangan ini adalah sebelum melakukan satu putaran pengembangan, selalu ada tahapan analisis risiko untuk mengetahui dan mengantisipasi risiko-risiko apa saja yang mungkin mengganggu proses pengembangan perangkat lunak sampai berakhirnya putaran tersebut. Siklus pengembangan ini cocok untuk pengembangan sistem perangkat lunak yang dalam skala medium sampai dengan besar. Ilustrasi pengembangan perangkat lunak dengan model Spiral ditunjukkan pada Gambar 4.3 berikut ini.



Gambar 4.3 Model Pengembangan Perangkat Lunak *Spiral*

4. Purwarupa atau *Prototyping*: model pengembangan perangkat lunak ini mengutamakan pembangunan purwarupa untuk dapat menggali berbagai macam kebutuhan dari klien maupun calon pengguna sistem perangkat lunak tersebut. Berlawanan dengan model pengembangan *Waterfall* yang harus mengetahui ke semua kebutuhan di awal, model pengembangan dengan purwarupa sangat cocok untuk pengembangan sistem di mana keseluruhan kebutuhan belum dapat ditentukan di awal. Kebutuhan-kebutuhan yang telah diketahui di awal akan dibuatkan purwarupanya dahulu untuk kemudian dikonsultasikan ke klien untuk sebagai sarana mengulik kebutuhan-kebutuhan lainnya yang sebelumnya tidak diketahui. Setelah melalui beberapa tahapan pengembangan purwarupa

sampai dapat disimpulkan bahwa semua kebutuhan dapat diketahui, maka purwarupa tersebut dijadikan acuan untuk membangun sistem perangkat lunak sesungguhnya yang dapat dikembangkan dengan model pengembangan lainnya seperti *Waterfall* ataupun *Spiral*.



Gambar 4.4 Model Pengembangan Perangkat Lunak *Prototyping*

Siklus hidup pengembangan perangkat lunak *Agile* dibuat berdasarkan suatu manifesto yang disebut Agile Manifesto (<https://agilemanifesto.org>). Manifesto ini dibuat sebagai jawaban dari keprihatinan bahwa banyak proyek perangkat lunak berakhir gagal ataupun berhasil dengan tidak sempurna. *Agile Manifesto* ini dideklarasikan oleh 17 orang praktisi pengembangan perangkat lunak, yang isinya:

Kami menemukan cara-cara yang lebih baik untuk mengembangkan perangkat lunak dengan melakukannya dan membantu orang lain untuk melakukannya. Melalui pekerjaan ini kami lebih menghargai:

- *Individu-individu dan interaksinya dibandingkan proses dan perangkat*
- *Perangkat lunak yang bekerja dibandingkan dengan dokumentasi yang lengkap*
- *Kolaborasi konsumen dibandingkan dengan negosiasi kontrak*
- *Menanggapi perubahan dibandingkan dengan mengikuti sebuah rencana.*

Bahwasanya, meskipun terdapat nilai pada item-item di sebelah kanan, kami lebih menghargai item-item di sebelah kiri.

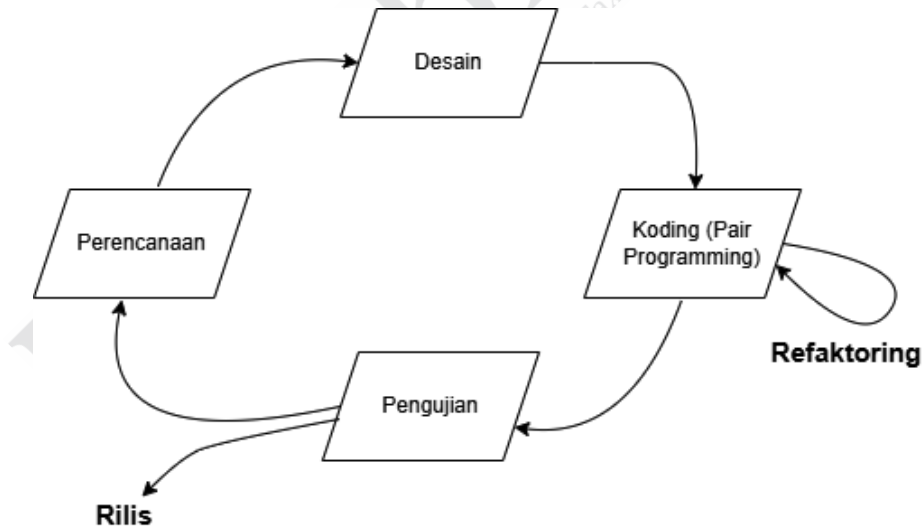
Agile Manifesto ini kemudian didefinisikan dalam 12 buah prinsip, yaitu:

- a. Prioritas utama kami adalah untuk memuaskan konsumen melalui pengantaran yang sedini dan sesering mungkin dari perangkat lunak yang berharga.
- b. Menyambut baik kebutuhan yang berubah, bahkan yang terjadi di akhir pengembangan. Proses *Agile* mengamankan perubahan untuk keuntungan kompetitif dari konsumen.
- c. Pengantaran perangkat lunak yang bekerja sesering mungkin, mulai dari dua minggu sampai dua bulan, dengan preferensi skala waktu yang lebih pendek.
- d. Orang bisnis dan pengembang perangkat lunak harus bekerja sama setiap hari sepanjang pengerjaan proyek.
- e. Membangun proyek yang dikelilingi individu-individu yang termotivasi. Berikan kepada mereka lingkungan dan dukungan yang dibutuhkan, dan percayakan bahwa mereka akan melakukan pekerjaan dengan baik.
- f. Metode yang paling efisien dan efektif untuk menyampaikan informasi kepada dan di dalam tim pengembang adalah melalui pembicaraan tatap muka.
- g. Perangkat lunak yang bekerja adalah pengukuran utama dari perkembangan.
- h. Proses *Agile* mempromosikan pengembangan yang berkelanjutan. Sponsor, pengembang, dan pengguna harus bisa memelihara laju yang konstan setiap saat.
- i. Perhatian yang terus-menerus untuk keunggulan teknis dan desain yang baik meningkatkan kelincahan.
- j. Kesederhanaan—seni untuk memaksimalkan besar pekerjaan dan bukan yang terselesaikan—adalah esensial.
- k. Arsitektur, kebutuhan, dan desain terbaik akan muncul dari tim yang bisa mengatur diri-sendiri.

- l. Pada interval yang teratur, tim merefleksikan tentang bagaimana cara menjadi lebih efektif, kemudian mengubah dan berubah kelakuannya berdasarkan hal tersebut.

Berdasarkan *Agile Manifesto* dan kedua belas prinsip tersebut, maka bermunculanlah berbagai macam model pengembangan perangkat lunak. Terdapat 3 model yang paling populer yang dibahas di buku ini yaitu Pemrograman Ekstrem, Kanban, dan Scrum.

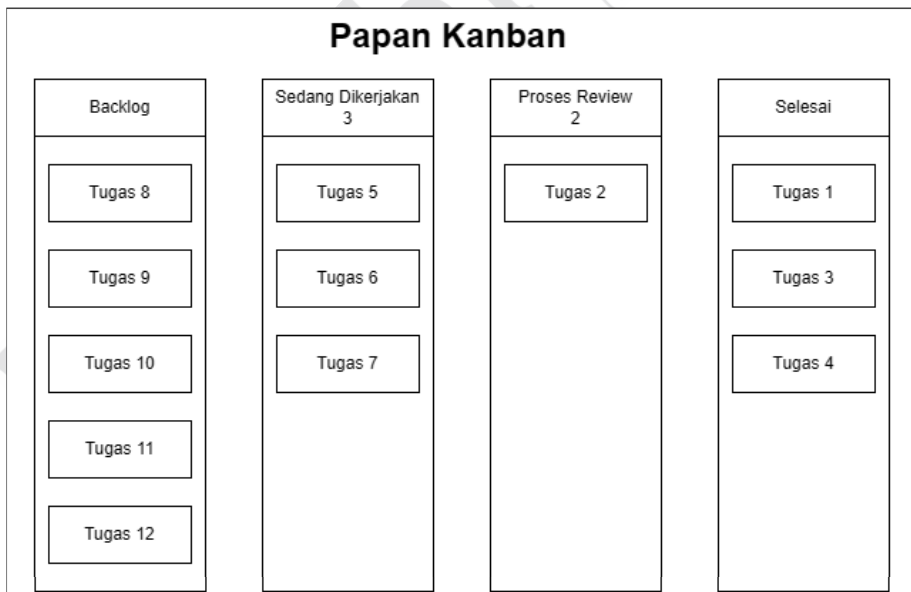
- a. Pemrograman Ekstrem (*Extreme Programming*): model pengembangan perangkat lunak dengan iterasi dan kolaborasi yang singkat di mana salah satu ciri khasnya adalah adanya pemrograman berpasangan (*pair programming*). Setiap fungsi yang dikerjakan pada suatu iterasi akan dikerjakan oleh 2 orang pemrogram secara bersama-sama. Pada iterasi berikutnya penugasan akan berganti dan juga pasangan pemrogram juga akan berganti untuk mengerjakan fungsi yang berbeda. Diharapkan dengan pendekatan ini maka aplikasi yang dikembangkan akan dapat dikerjakan dengan cepat, tetapi tetap dapat menghasilkan perangkat lunak berkualitas tinggi.



Gambar 4.5 Model Pengembangan Perangkat Lunak *Extreme Programming*

Kegiatan lain yang dilakukan pada Pemrograman Ekstrem ini selain *Pair Programming* adalah Refactoring. Kegiatan refactoring merupakan kegiatan untuk melakukan penyederhanaan dari kode sumber yang telah ditulis agar dapat mencapai efisiensi kode yang lebih baik. Apabila kode sumber telah ditulis dan terkumpul, maka ada masa di mana kode sumber yang terkumpul tersebut terlihat kurang efisien dan terlihat bertele-tele, kegiatan refactoring ini akan berusaha menyederhanakan kode sumber tersebut agar bisa lebih ramping untuk fungsionalitas yang sama.

- b. *Kanban*: Kanban merupakan salah satu metode Agile yang mengikuti sebuah alur kerja yang divisualisasikan dalam sebuah papan yang disebut Papan Kanban. Kanban ini pertama kali ditemukan oleh Taiichi Ohno di perusahaan Toyota di tahun 1940-an dan kemudian diadaptasikan untuk pengembangan perangkat lunak oleh David J. Anderson di tahun 2007 (Anderson, 2010). Contoh dari Papan Kanban dapat ditunjukkan pada Gambar 4.6 berikut ini.

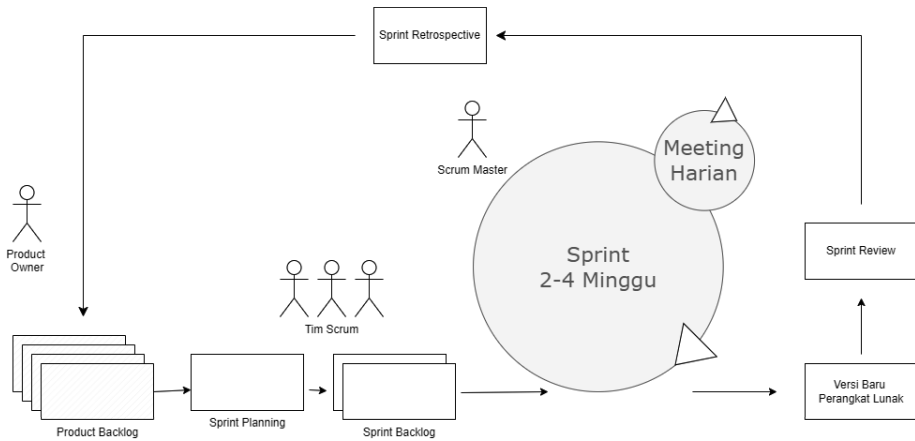


Gambar 4.6 Contoh Papan Kanban

Pada proses pengembangan dengan Kanban ini, fungsi perangkat lunak dipecah-pecah menjadi tugas-tugas kecil yang diletakkan di kolom “Backlog”. Setiap kali sebuah tugas dikerjakan, maka kartunya akan dipindahkan ke bagian “Sedang Dikerjakan”, kemudian apabila selesai dikerjakan akan masuk ke bagian “Proses Review” dan kemudian kalau selesai akan masuk ke kolom “Selesai”.

Semua tugas akan melewati tahapan yang sama, namun yang menjadi unik adalah adanya pembatas kartu yang dituliskan di kolom “Sedang Dikerjakan” dan “Proses Review” yang tidak boleh dilanggar oleh siapa pun yang terlibat pada proses pengembangan (disebut *Work In Progress Limit* atau *WIP Limit*). Jadi misalnya kalau di Gambar 4.6 di atas ternyata di kolom “Sedang Dikerjakan” terdapat 3 kartu tugas dan sudah mencapai limit maksimumnya, maka semua bagian dari tim akan melakukan berbagai upaya agar ada kartu yang bisa pindah ke kolom berikutnya yaitu “Proses Review”. Di kolom “Proses Review” juga terdapat limit berupa 2 kartu, sehingga hanya masih terdapat 1 slot kartu kosong lagi yang dapat diisi. Pendekatan Kanban ini dilakukan agar pengerjaan proyek bisa berjalan dengan lancar dan berkelanjutan.

- c. *Scrum*: model pengembangan perangkat lunak dengan iterasi yang singkat yang disebut dengan *Sprint*. Sebuah sprint lamanya bisa 2 – 4 minggu sesuai kesepakatan dan setiap sprint akan menyelesaikan beberapa kebutuhan yang dituliskan dalam sebuah cerita pengguna atau *user stories*. Tim pengembangan Scrum ini merupakan tim yang kecil tetapi harus memiliki kemampuan di atas rata-rata. Setiap hari akan diadakan rapat singkat untuk membahas apa yang telah dicapai dan rencana di hari ini. Selama proses pengembangan perangkat lunak yang bisa mencapai 3 – 5 *Sprints*, hubungan antara tim pengembang dengan dunia luar hanya melalui pimpinan tim yang disebut *Scrum Master* (Scrum, 2025).



Gambar 4.7 Model Pengembangan Perangkat Lunak SCRUM

Apabila proses pengembangan perangkat lunak tersebut, baik yang berbasis rencana ataupun *Agile*, dapat diukur secara kuantitatif dalam sebuah metrik, maka ukuran dari metrik tersebut memungkinkan kita untuk memahami, mengendalikan, dan meningkatkan proses tersebut. Mengukur proses dari pengembangan sistem perangkat lunak melibatkan beberapa aspek, yaitu:

- Analisis dari bagaimana sebuah produk perangkat lunak tersebut dikembangkan
- Siklus hidup pengembangan perangkat lunak yang dipilih
- Pengantaran (*deliverables*) apa saja yang akan diberikan
- Bagaimana semuanya itu akan dianalisis
- Bagaimana sebuah proses dapat membantu membuat produk perangkat lunak dengan lebih cepat
- Bagaimana sebuah proses dapat membantu membuat produk perangkat lunak dengan lebih baik

B. Metrik Pemantauan Pembentukan Cacat (*Defect* atau *Bug*)

Cacat perangkat lunak (*software defect*) adalah galat, kekurangan, kesalahan, kegagalan, dan bentuk kecacatan lainnya yang ditemukan pada sebuah sistem perangkat lunak yang mencegah sistem tersebut berperilaku sebagaimana mestinya. Cacat perangkat lunak dapat juga dikatakan sebagai galat perangkat lunak (*software error*) atau kutu perangkat lunak (*software bug*).

Cacat perangkat lunak memiliki dampak yang beragam pada sistem perangkat lunak tersebut saat dijalankan pada sebuah komputer atau laptop. Dampak tersebut akan menimbulkan berbagai ketidaknyamanan yang akan dirasakan oleh pengguna. Dampak yang ringan mungkin tidak terdeteksi oleh pengguna secara langsung karena seolah semua fungsinya berjalan semestinya, sedangkan cacat yang serius dapat menyebabkan perangkat lunak mengalami kegagalan berjalan (*crash*) atau pembekuan (*freeze*) yang mengarah pada penolakan layanan. Cacat bentuk lain juga bisa berupa cacat keamanan yang memungkinkan pengguna lain yang memiliki niat jahat untuk mendapatkan hak akses secara tidak sah yang bisa merugikan pengguna atau organisasi di mana pengguna tersebut bekerja.

Terdapat 4 kategori cacat perangkat lunak:

1. Kesalahan karena komisi: kesalahan karena komisi artinya bahwa suatu cacat perangkat lunak terjadi karena fungsi atau kemampuan yang diminta ternyata tidak sama dengan yang sebenarnya diminta.
2. Kesalahan karena kelalaian: kesalahan karena kelalaian artinya bahwa suatu cacat perangkat lunak terjadi karena suatu fungsi atau kemampuan ternyata tidak atau lupa dibuat oleh tim pengembang perangkat lunak.
3. Kesalahan karena kejelasan dan ambiguitas: kesalahan karena kejelasan dan ambiguitas terjadi karena pernyataan-pernyataan kebutuhan yang dituangkan dalam dokumen kebutuhan ternyata kurang jelas atau ambigu sehingga tim pengembang mengalami kesulitan untuk mewujudkannya dalam sistem perangkat lunak.

4. Kesalahan karena kecepatan dan kapasitas: kesalahan karena kecepatan dan kapasitas ditemukan saat pengujian yang menunjukkan bahwa kinerja dari sistem perangkat lunak ternyata di bawah dari target kinerja yang disepakati sebelumnya.

Sumber-sumber dari kecacatan perangkat lunak berdasarkan tahapan pengembangan perangkat lunak, yaitu:

1. Galat pada kebutuhan perangkat lunak: galat yang muncul saat menentukan kebutuhan dari perangkat lunak yang akan dikembangkan.
2. Galat pada perancangan atau desain: galat yang muncul karena salah menafsirkan kebutuhan perangkat lunak saat melakukan perancangan ataupun desain.
3. Galat pada kode sumber: galat pada saat menuliskan kode sumber hasil terjemahan dari desain oleh pemrogram.
4. Galat pada dokumentasi pengguna: galat pada dokumentasi yang diberikan kepada para pengguna sistem perangkat lunak.
5. Galat karena perbaikan yang salah: galat yang terjadi karena perbaikan yang dilakukan ternyata malah menimbulkan galat yang baru.
6. Galat pada data dan tabel basis data: galat pada saat perancangan sampai pemrograman basis data.
7. Galat pada kasus-kasus pengujian: galat pada saat merancang koleksi kasus pengujian.

1. Metrik Cacat Per Jam (Defect Per Hour)

Metrik ini mengukur jumlah cacat di setiap jam selama waktu pengujian dan pemeliharaan dari sebuah sistem perangkat lunak. Pada awal pelepasan suatu versi perangkat lunak, biasanya laporan cacat akan juga melonjak dan juga harus segera ditanggapi oleh tim pemelihara perangkat lunak. Lonjakan ini akan semakin lama semakin melandai sampai terdapat rilis versi berikutnya. Adapun rumusnya adalah sebagai berikut.

$$DPH = \frac{\text{Total Cacat yang Ditemukan}}{\text{Total Waktu Pengujian dalam Jam}}$$

Sebagai tolok ukur untuk nilai DPH adalah kalau nilainya lebih dari 5 Cacat per jam berarti sistem mungkin masih labil dan masih banyak cacat yang belum ditemukan. Untuk DPH antar 1 sampai 5 dianggap nilai yang wajar; sedangkan sebuah sistem perangkat lunak yang stabil akan memiliki nilai DPH kurang dari 1 Cacat per jam.

2. Metrik Cacat Per Function Point

Metrik ini juga menunjukkan tingkat produktivitas cacat dalam satuan per Poin Fungsi (*Function Point*). Sama seperti Metrik Cacat yang Diproduksi Per Jam, metrik ini seharusnya mencapai nilai serendah mungkin meskipun perlu diketahui bahwa membuat program yang bebas cacat adalah hal yang tidak memungkinkan. Adapun rumusnya adalah sebagai berikut.

$$\text{Cacat per Function Point} = \frac{\text{Total Cacat yang Ditemukan}}{\text{Total Function Points}}$$

3. Metrik Jumlah Tambalan Setelah Rilis (*Patches After Release*)

Metrik ini mengukur jumlah tambalan (*patches*) yang didistribusikan setelah rilis dari sistem perangkat lunak. Tingkatan tambalan ini sebenarnya memiliki dua arti, arti pertama bahwa jumlah tambalan yang banyak berarti menandakan biaya pemeliharaan perangkat lunak yang tinggi, arti yang kedua adalah jumlah tambalan yang banyak menandakan sebuah perangkat lunak yang masih banyak cacatnya setelah rilis.

Sebagai acuan nilai metrik ini adalah apabila terdapat 0 – 1 tambalan per bulan artinya sistem sudah dianggap stabil. Nilai 2 – 5 tambalan per bulan menandakan masih ada cacat yang terlewat, tetapi masih dianggap masih dalam wajar; sedangkan nilai tambalan lebih dari 5 per bulan menandakan kualitas sistem perangkat lunak yang belum baik dan masih terdapat cacat-cacat yang tersembunyi.

4. Metrik Kegagalan Membangun (Failed Builds)

Metrik yang mengukur jumlah kegagalan pembangunan (*build*) dari sistem perangkat lunak saat dibangun di setiap malamnya. Kelemahan dari metrik ini adalah tingkat kegagalan bisa sebenarnya tergantung dari tingkat kompleksitas dari sistem perangkat lunak itu sendiri. Adapun rumusnya adalah sebagai berikut.

$$\text{Kegagalan Membangun} = \frac{\text{Jumlah Build yang Gagal}}{\text{Jumlah Total Build dalam Sehari}} \times 100\%$$

C. Metrik Perubahan Kebutuhan (Requirement Changes)

Metrik yang menghitung perubahan kebutuhan yang terjadi selama pembangunan sistem perangkat lunak. Meskipun perubahan harus dianggap oleh tim pengembang sebagai sesuatu yang tidak terelakkan, namun perubahan yang terlalu sering akan juga menimbulkan kekacauan dalam pengembangan suatu sistem perangkat lunak. Adapun rumusnya adalah sebagai berikut.

$$\text{Perubahan Kebutuhan} = \frac{\text{Jumlah Kebutuhan yang Berubah}}{\text{Jumlah Total Kebutuhan}} \times 100\%$$

Anggapan yang ada selama ini adalah perubahan seharusnya tidak boleh lebih dari 25% dari semua kebutuhan yang ada agar tidak menimbulkan kekacauan dalam proses pengembangan.

D. Metrik Waktu Pemrograman (Programming Time)

Metrik yang mengukur waktu yang dibutuhkan oleh pemrogram untuk membangun suatu modul dari perangkat lunak. Cara mengukurnya adalah dengan meminta setiap pemrogram untuk mengisi formulir untuk melacak waktu yang diperlukan setiap kali pemrogram tersebut selesai mengembangkan suatu modul. Modul dalam hal ini bisa berupa suatu Package atau suatu fungsi khusus dalam sebuah perangkat lunak.

Adapun rumusnya adalah sebagai berikut.

$$\text{Waktu Pemrograman} = \frac{\text{Total Waktu Pengembangan}}{\text{Jumlah Modul}}$$

E. Metrik Proses Evolusi

Metrik proses evolusi adalah metrik yang mengukur berbagai macam perubahan dari metrik ukuran perangkat lunak selama evolusi perangkat lunak tersebut dalam masa pemeliharaannya. Beberapa pengukuran yang bisa dilakukan antara lain:

1. Perubahan LOC: baris kode bisa berubah selama masa pemeliharaan. Pertambahan baris kode bisa terjadi saat adanya penambahan fitur baru, penambalan, dan lain-lainnya, sedangkan pengurangan baris kode bisa terjadi karena proses Refactoring ataupun penghilangan kode sumber yang sudah tidak terpakai. Rumus yang bisa dipakai adalah sebagai berikut.

$$\text{Perubahan LOC} = \text{LOC yang Ditambah} - \text{LOC yang Berkurang}$$

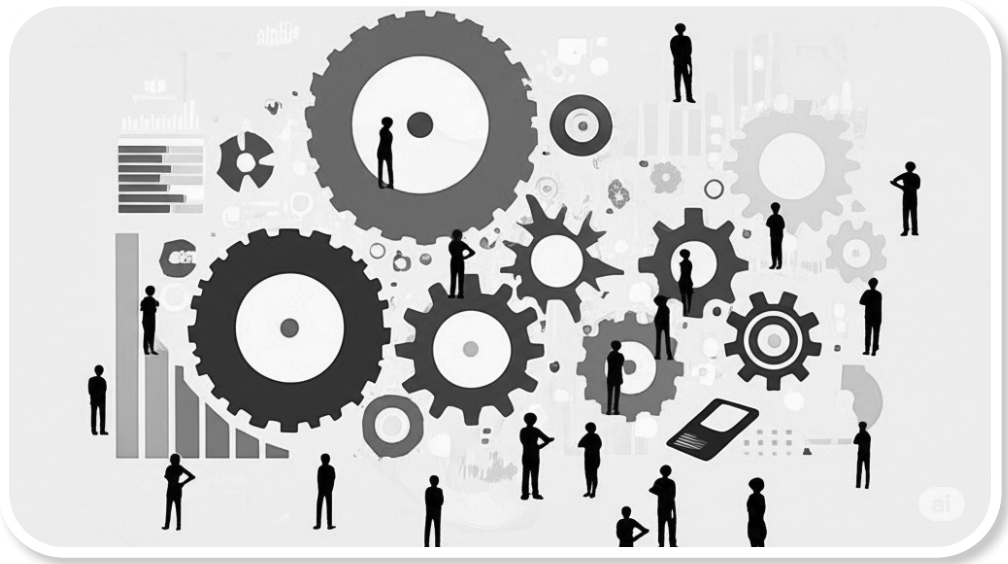
2. *Churn Rate*: persentase baris kode yang berubah dalam satu periode waktu. Nilai *Churn Rate* yang tinggi menandakan adanya potensi ketidakstabilan sistem yang harus diperhatikan oleh tim pemelihara, sedangkan nilai *Churn Rate* yang rendah menandakan bahwa sistem perangkat lunak sudah mencapai kestabilan.

$$\text{Churn Rate} = \frac{\text{LOC yang Berubah} + \text{LOC yang Dihapus}}{\text{Total LOC}}$$

3. Laju Pertumbuhan Baris Kode: metrik yang menandakan laju pertambahan baris kode selama masa pemeliharaan. Misalnya sebuah sistem perangkat lunak dengan total LOC sebesar 10K dan kemudian dalam sebulan ada penambahan LOC sebesar 1K, maka Laju Pertumbuhan Baris Kodenya adalah 10%.

BAB 5

Metrik Orang



A. Pendahuluan

Tim pengembang atau pemrogram merupakan salah satu aset paling penting dari sebuah perusahaan atau departemen Teknologi Informasi. Pengukuran kinerja dari tim pengembang tersebut ataupun setiap individu pemrogram perlu dilakukan agar perkembangan dari pengembangan perangkat lunak dapat dipantau dan dikelola dengan baik. Namun, seseorang cenderung tidak menyukai apabila hanya dianggap sebagai angka saja, sehingga metrik orang ini sifatnya cukup kontroversial. Metrik orang merupakan nilai ukuran yang perlu diketahui oleh manajemen perusahaan untuk:

1. Tujuan finansial: tim manajemen tentu saja ingin mengetahui metrik orang ini untuk dapat memprediksi biaya yang dibutuhkan untuk mengembangkan suatu sistem perangkat lunak.

2. Tujuan manajemen proyek: pengukuran metrik orang ini akan membantu dalam pengelolaan sumber daya manusia yang merupakan salah satu area pengetahuan dalam manajemen proyek.
3. Identifikasi persoalan sumber daya manusia: perkembangan metrik perangkat lunak sepanjang berjalannya proses pengembangan suatu perangkat lunak dapat mengindikasikan persoalan yang akan atau sedang terjadi saat itu sehingga intervensi dapat dilakukan oleh pihak manajemen apabila diperlukan.

Mengukur orang-orang yang terlibat dalam pengembangan sistem perangkat lunak melibatkan beberapa aspek, yaitu:

1. Melibatkan analisis dari orang-orang yang sedang mengembangkan produk perangkat lunak, khususnya tim pengembang perangkat lunak.
2. Fokus pengukuran pada seberapa cepat orang-orang tersebut bekerja saat mengembangkan perangkat lunak.
3. Fokus tambahan adalah pada seberapa banyak kesalahan (*bug*) yang dibuat oleh pengembang perangkat lunak.
4. Mengukur aspek tambahan lainnya, misalnya berapa banyak cuti sakit yang diambil oleh tim pengembang.

Beberapa metrik orang yang sering dipakai pada saat proses pengembangan sistem perangkat lunak adalah sebagai berikut.

1. Metrik *Person-Month* (atau *Person-Hour*): metrik yang menggambarkan besarnya usaha yang harus dikeluarkan untuk mengembangkan sebuah sistem perangkat lunak dengan menggabungkan jumlah orang dan satuan waktu (bulan atau jam).
2. Metrik LOC per jam: metrik yang menggambarkan tingkat produktivitas seorang atau tim pengembang dalam menuliskan kode sumber dari sistem perangkat lunak tersebut.
3. Metrik FP per jam: metrik yang menggambarkan tingkat produktivitas seorang atau tim pengembang dalam merealisasikan Poin Fungsi dari sistem perangkat lunak tersebut.
4. Metrik Kebutuhan per jam: metrik yang menggambarkan tingkat produktivitas tim pengembang dengan menghitung jumlah kebutuhan atau *requirement* yang telah dapat direalisasikan di setiap jamnya.

5. Metrik Cacat yang Diproduksi per Jam: metrik yang menggambarkan produktivitas cacat yang dihasilkan oleh tim pengembang perangkat lunak setiap jamnya dalam satuan jumlah cacat per ukuran baris kode yang dibuat.
6. Metrik Cacat yang Diproduksi per FP: metrik yang menggambarkan produktivitas cacat yang dihasilkan oleh tim pengembang perangkat lunak di setiap Poin Fungsi yang dibuat.

Berikut ini penjelasan yang lebih detail dan mendalam untuk masing-masing metrik di atas.

B. Metrik *Person-Month*

Person-Month adalah sebuah pengukuran usaha dalam pengembangan suatu produk atau jasa (tidak hanya untuk pengembangan sistem perangkat lunak saja). Satuan lain yang sering dipergunakan adalah *Person-Hour* (orang-jam). Satuan ini menggambarkan korelasi antara jumlah orang yang dibutuhkan untuk mengembangkan suatu sistem perangkat lunak dengan waktu yang akan dihabiskan dalam pengembangannya. Misalnya kalau pengembangan suatu aplikasi ditentukan usahanya adalah 100 *Person-Month* artinya kalau jumlah personelnya ada 20 orang, maka durasi pengerjaannya adalah 5 bulan. Adapun rumusnya adalah sebagai berikut.

$$PM = \frac{\text{Total Jam Kerja yang Dibutuhkan}}{\text{Jam Kerja per Orang per Bulan}}$$

Satu hal yang harus diwaspadai dalam pengukuran dengan satuan ini adalah sifatnya yang tidak sepenuhnya linier atau proporsional lurus. Misalnya kalau usaha yang diperlukan sebesar 100 *Person-Month*, bukan berarti kalau ada 100 orang personel akan menghabiskan waktu 1 bulan saja, atau kebalikannya kalau jumlah personelnya ada 1 orang maka akan menghabiskan waktu sebesar 100 bulan (8 tahun lebih). Ada aspek-aspek lain yang harus diperhatikan dalam hal ini, yaitu:

1. Penambahan personel tidak berbanding lurus sepenuhnya dengan pengurangan waktu pengerjaan. Semakin banyak personel akan membutuhkan upaya komunikasi yang koordinasi yang lebih intens

sehingga mungkin penambahan tersebut malah akan memperlambat waktu pengerjaan.

2. Berbagai aspek keahlian yang diperlukan dalam mengembangkan sistem perangkat lunak juga harus diperhitungkan. Tidak mungkin kita mengharapkan seorang pemrogram untuk menjadi pakar dalam segala bidang. Tingkat proporsi dari setiap bidang keahlian yang diperlukan juga harus diperhatikan saat memperhitungkan jumlah personel yang dibutuhkan.
3. Pengurangan personel sampai tingkat yang ekstrem misalnya satu atau dua orang saja juga akan bersifat kontra-produktif. Jumlah personel yang terlalu sedikit akan membutuhkan keahlian yang kompleks yang tidak mungkin akan dipenuhi oleh beberapa personel tersebut saja.

Penggunaan metrik *Person-Month* sebaiknya dipergunakan saat estimasi dan dianggap sebagai perkiraan yang kasar dan bukan estimasi yang sifatnya terperinci dan akurat dengan mempertimbangkan beberapa aspek yang telah disebutkan di atas.

C. Metrik LOC per Jam

Lines of Code adalah metrik perangkat lunak yang telah dibahas di bab 2 (Metrik Produk), sedangkan metrik LOC per jam lebih menggambarkan tingkat produktivitas dari seorang pengembang perangkat lunak. Semakin besar nilai LOC per jam menandakan semakin produktifnya seorang pemrogram. Sebagai acuan untuk mengukur tingkat produktivitas ini misalnya untuk bahasa pemrograman Java sekitar 15 – 30 LOC per jam, C# sekitar 20 – 40 LOC per jam, Python sekitar 20 – 50 LOC per jam, dan lain-lainnya. Besaran pastinya dari setiap pemrogram beragam tergantung dari kemampuan dari setiap individu dari pemrogram. Adapun rumusnya adalah sebagai berikut.

$$LOC \text{ per Jam} = \frac{\text{Total Baris Kode yang Dihasilkan atau Dimodifikasi}}{\text{Jumlah Jam yang Dihabiskan}}$$

Ada beberapa hal penting yang harus diperhatikan apabila menggunakan metrik pengukuran ini untuk mengukur produktivitas dari seorang pemrogram. Banyaknya baris kode yang dihasilkan belum tentu menunjukkan tingkat produktivitas yang sesungguhnya karena seorang pemrogram yang lebih berpengalaman biasanya bisa menghasilkan kode program yang lebih padat dan singkat dibandingkan pemrogram yang kurang berpengalaman. Faktor kedua adalah bahasa pemrograman juga berpengaruh karena semakin tinggi generasi bahasa pemrogramannya biasanya juga bisa mendapatkan suatu kemampuan fungsi dengan baris kode yang lebih padat dan singkat.

D. Metrik FP per Jam

Metrik Poin Fungsi (*Function Point*) merupakan salah satu metrik ukuran yang dibahas di Bab 2. Metrik FP per Jam menandakan tingkat produktivitas dari seorang pemrogram dalam menghasilkan Poin Fungsi setiap jamnya. Metrik ini lebih baik dibandingkan metrik sebelumnya yaitu LOC per jam karena Poin Fungsi sifatnya lebih obyektif terhadap bahasa pemrograman dan tingkat pengalaman dari pemrogramnya. Adapun kelemahan dari Poin Fungsi ini adalah otomasi dari metrik ini menjadi lebih sulit dibandingkan metrik LOC per jam. Nilai acuan dari metrik ini juga serupa dengan nilai LOC per jam yang disebutkan di sebelumnya dengan melihat tabel konversi yang dituliskan di Tabel 2.1 di buku ini. Adapun rumusnya adalah sebagai berikut.

$$FP \text{ per Jam} = \frac{\text{Total Poin Fungsi yang Dihasilkan atau Dimodifikasi}}{\text{Jumlah Jam yang Dihabiskan}}$$

E. Metrik Kebutuhan per Jam

Metrik kebutuhan per Jam menghitung produktivitas dari tim pengembang perangkat lunak dengan menghitung jumlah kebutuhan (*requirement*) yang terselesaikan setiap jamnya. Semakin tinggi nilai ini menandakan semakin produktifnya tim pengembang dan dapat dijadikan patokan untuk memperkirakan waktu penyelesaian proyek perangkat lunak. Adapun rumusnya adalah sebagai berikut.

$$\text{Kebutuhan per Jam} = \frac{\text{Total Kebutuhan yang Diselesaikan}}{\text{Jumlah Jam yang Dihabiskan}}$$

Hal yang perlu diperhatikan apabila menggunakan metrik ini adalah perlunya pemahaman bahwa tidak semua kebutuhan itu memiliki tingkat kesulitan yang sama. Semua kebutuhan yang tercatat perlu ada kategorisasi apakah masuk dalam kategori ringan, sedang, dan berat dengan estimasi waktu pengerjaan yang berbeda pula. Dengan adanya kategorisasi ini maka prediksi produktivitas tim pengembang dengan metrik kebutuhan per jam ini akan menjadi semakin akurat.

BAB 6

Perangkat Lunak Pengukur Metrik



A. Pendahuluan

Mengumpulkan berbagai metrik perangkat lunak bukanlah pekerjaan yang mudah dan membutuhkan usaha yang besar. Tidak semua metrik perangkat lunak yang dibahas di bab-bab sebelumnya dapat dilakukan upaya pengumpulan dengan mudah dan otomatis. Beberapa metrik, khususnya yang berkaitan dengan metrik proses dan metrik orang masih harus dilakukan pengumpulan secara manual atau masih membutuhkan intervensi manual untuk dapat diterapkan. Sedangkan untuk metrik-metrik yang berkaitan dengan produk dan metrik kualitas sebagian dapat dilakukan pengukuran secara otomatis dengan dibantu dengan beberapa perangkat lunak lainnya.

Pada bab ini akan dibahas beberapa perangkat lunak yang dapat dipergunakan untuk mengukur berbagai metrik. Beberapa tantangan yang dihadapi saat melakukan pengukuran berbagai metrik perangkat lunak adalah sebagai berikut.

1. Adalah sulit untuk menemukan satu perangkat saja yang dapat melakukan pengukuran dari semua metrik perangkat lunak yang dibutuhkan. Metrik-metrik yang dibutuhkan juga tergantung dari tujuan atau objektif dari pengukuran itu sendiri.
2. Apabila menggunakan beberapa perangkat sekaligus untuk mengumpulkan berbagai metrik perangkat lunak, terkadang terdapat pengukuran yang berbeda untuk metrik yang sama dari perangkat-perangkat yang berbeda. Ini tentu saja akan membuat frustrasi karena menimbulkan kebingungan mana sebenarnya nilai-nilai metrik yang benar.
3. Perangkat yang tersedia untuk mengumpulkan berbagai metrik perangkat lunak hampir semuanya bisa mengakomodasi bahasa pemrograman populer seperti Java, C, C++, Ruby, dan lain-lain. Bahasa Java, C, dan C++ merupakan bahasa pemrograman yang dikenal oleh banyak perangkat pengukur metrik.

Berikut ini penggambaran beberapa perangkat lunak yang dapat dipakai untuk mendapatkan beberapa metrik perangkat lunak yang dibutuhkan.

B. Perangkat Lunak Pendukung

1. XAMPP

XAMPP bukan merupakan perangkat yang dapat mengumpulkan berbagai macam metrik perangkat lunak, tetapi perangkat ini penting untuk dapat menyimpan dan menganalisis hasil-hasil dari berbagai perangkat lainnya. XAMPP itu sendiri adalah sekumpulan perangkat lunak yang diperlukan untuk mengolah data yang terdiri dari:

- a. Web Server *Apache*
- b. Bahasa Skrip PHP
- c. Basis Data MySQL (disebut juga Maria DB)
- d. Bahasa Pemrograman Perl
- e. Antarmuka pengelola basis data PHPMyAdmin

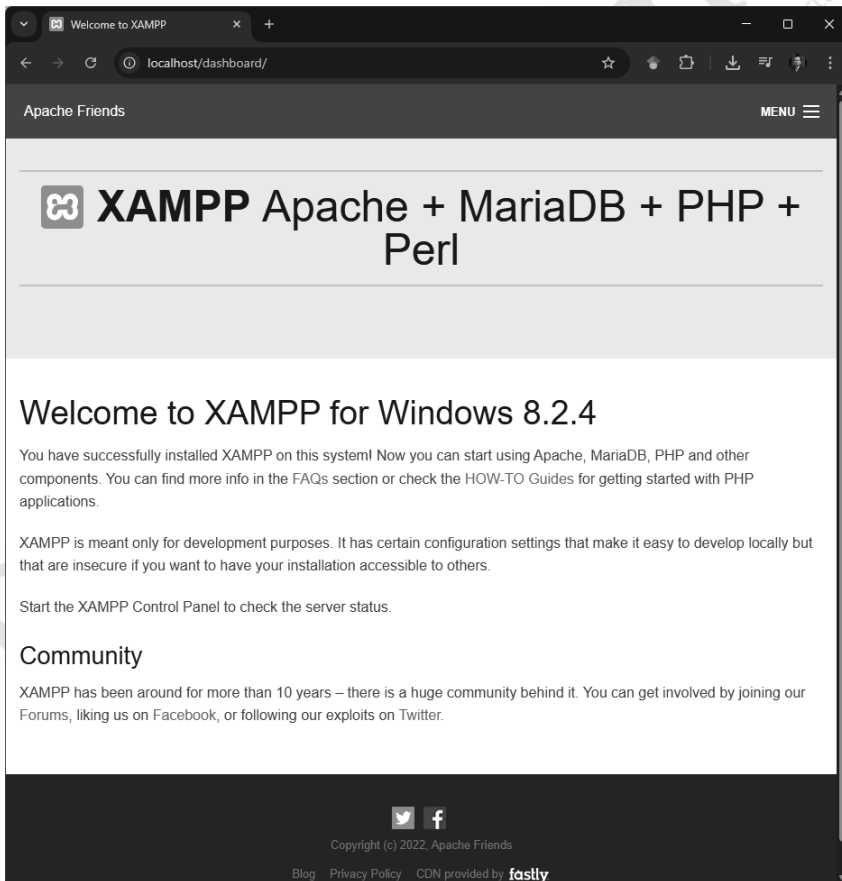
Fungsi utama dari perangkat ini adalah dipakai sebagai tempat penyimpanan data-data metrik dan juga tempat membangun portal untuk mengolah data dari data-data metrik tersebut dengan bahasa skrip PHP. Situs dari XAMPP adalah di tautan <https://www.apachefriends.org/> seperti yang ditunjukkan pada Gambar 6.1 berikut ini.



Gambar 6.1 Situs XAMPP

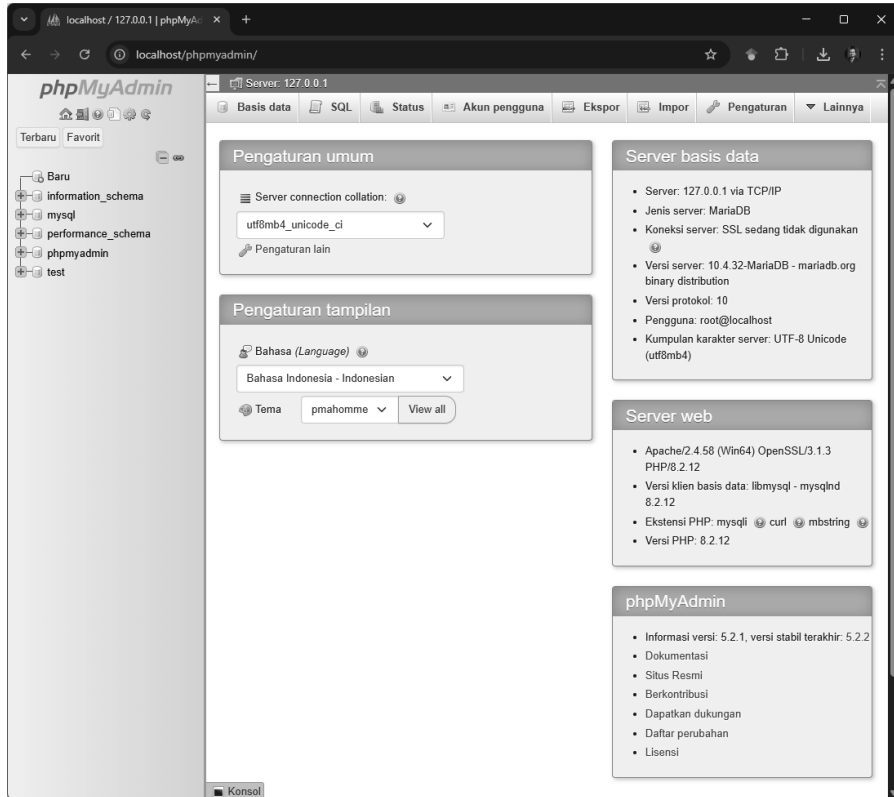
Perangkat ini perlu dipasang di komputer dengan Sistem Operasi Windows 11, MacOS ataupun Linux (misalnya Ubuntu). Langkah-langkah yang diperlukan adalah sebagai berikut.

- a. Unduh berkas instalasi dari situs XAMPP di situs <https://www.apachefriends.org/> sesuai dengan Sistem Operasi yang ada pada komputer.
- b. Pasanglah XAMPP pada komputer dengan menggunakan berkas instalasi tersebut. Petunjuk instalasi akan berbeda untuk berbagai Sistem Operasi yang didukung.
- c. Cara mengecek apakah perangkat sudah bekerja dengan membuka browser dan mengetikkan alamat <https://localhost>.



Gambar 6.2 Tampilan Depan XAMPP

- d. Pada perangkat ini, aplikasi yang paling penting adalah PHPMyAdmin dengan mengetikkan alamat *https://localhost/phpmyadmin*. Dengan aplikasi ini kita dapat melakukan berbagai macam pengaturan untuk basis data yang tersedia.

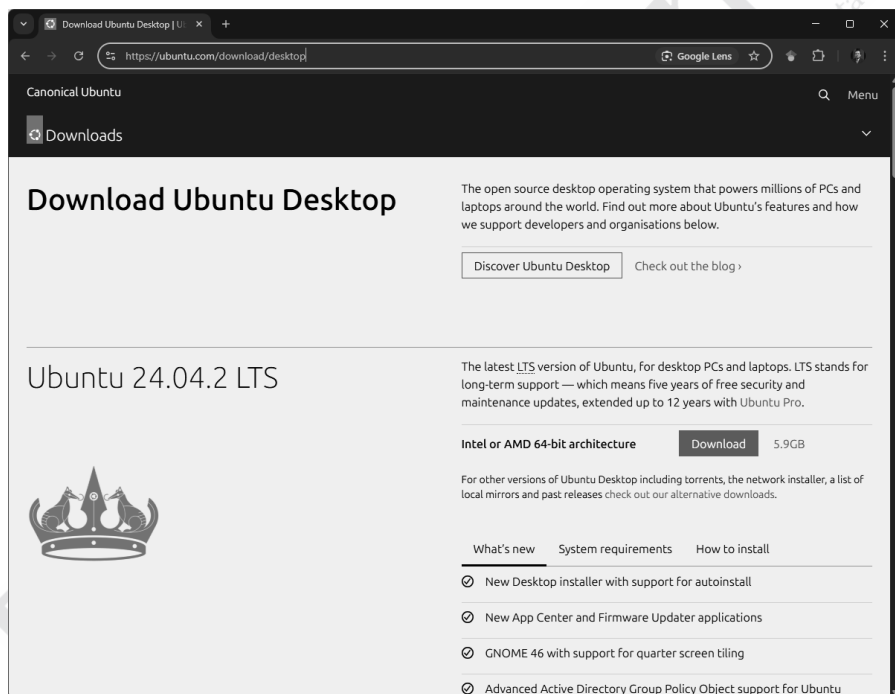


Gambar 6.3 Tampilan PHPMyAdmin

Kita bisa melakukan manipulasi basis data dengan menggunakan PHPMyAdmin ini. Basis data yang tersedia dapat dipergunakan untuk menyimpan data metrik yang dikumpulkan dari perangkat lainnya.

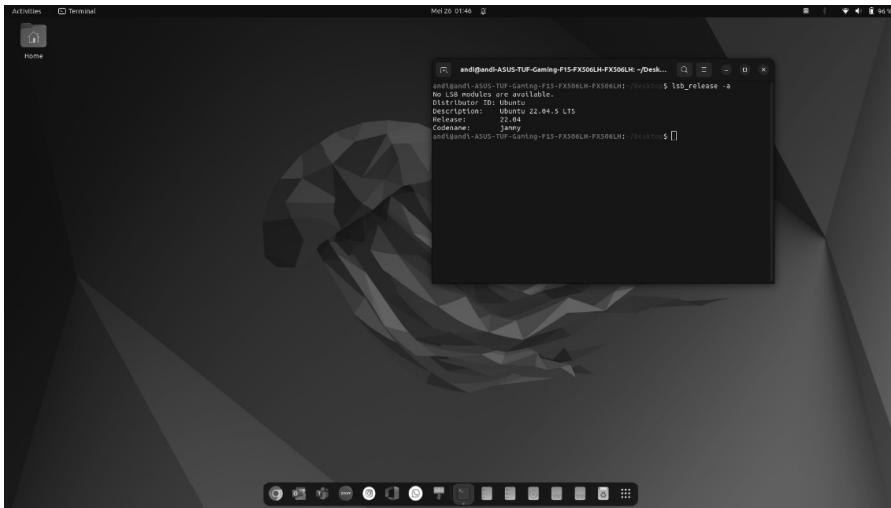
2. Sistem Operasi Ubuntu Linux

Meskipun Sistem Operasi yang populer untuk komputer personal dan laptop adalah Microsoft Windows 11 ataupun MacOS, Sistem Operasi Linux seperti Ubuntu merupakan Sistem Operasi yang mudah dipergunakan dan dikonfigurasi untuk keperluan penelitian dan pengumpulan data. Untuk melakukan instalasi Sistem Operasi ini, diperlukan komputer atau laptop dengan partisi hard disk yang dikosongkan atau hard disk tambahan sehingga komputer bisa dipasang secara *dual-boot* Windows 11 dan Ubuntu. Berkas untuk diunduh dapat diambil dari tautan <https://ubuntu.com/download/desktop> seperti yang ditunjukkan pada Gambar 6.4 berikut ini.



Gambar 6.4 Laman Situs Ubuntu

Apabila sudah selesai dipasang maka tampilan di layar Sistem Operasi Ubuntu adalah yang ditunjukkan pada Gambar 6.5 berikut ini. Pada Gambar ini Sistem Operasi Ubuntu yang dipasang adalah versi 22.04 LTS. Versi terbaru sekarang ini adalah versi 24.04 LTS.

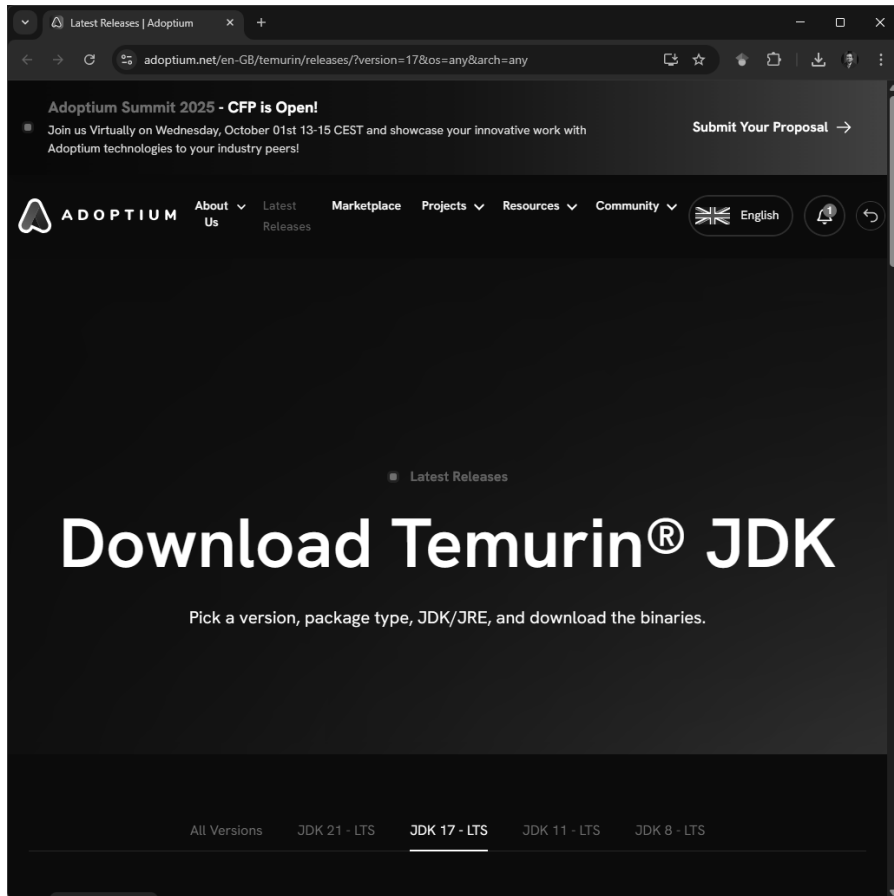


Gambar 6.5 Tampilan Sistem Operasi Ubuntu

Untuk pemilihan versi Ubuntu yang tepat, disarankan untuk mencari versi yang terdapat akhiran “LTS” yang berarti mendapatkan dukungan dalam jangka panjang (*Long Time Support*) yaitu sampai 2 tahun. Pemasangan versi LTS yang lebih lama juga disarankan karena Sistem Operasi yang terbaru terkadang belum disertai perubahan pada berbagai perangkat pendukung lainnya yang bisa mendukung penelitian atau perekaman.

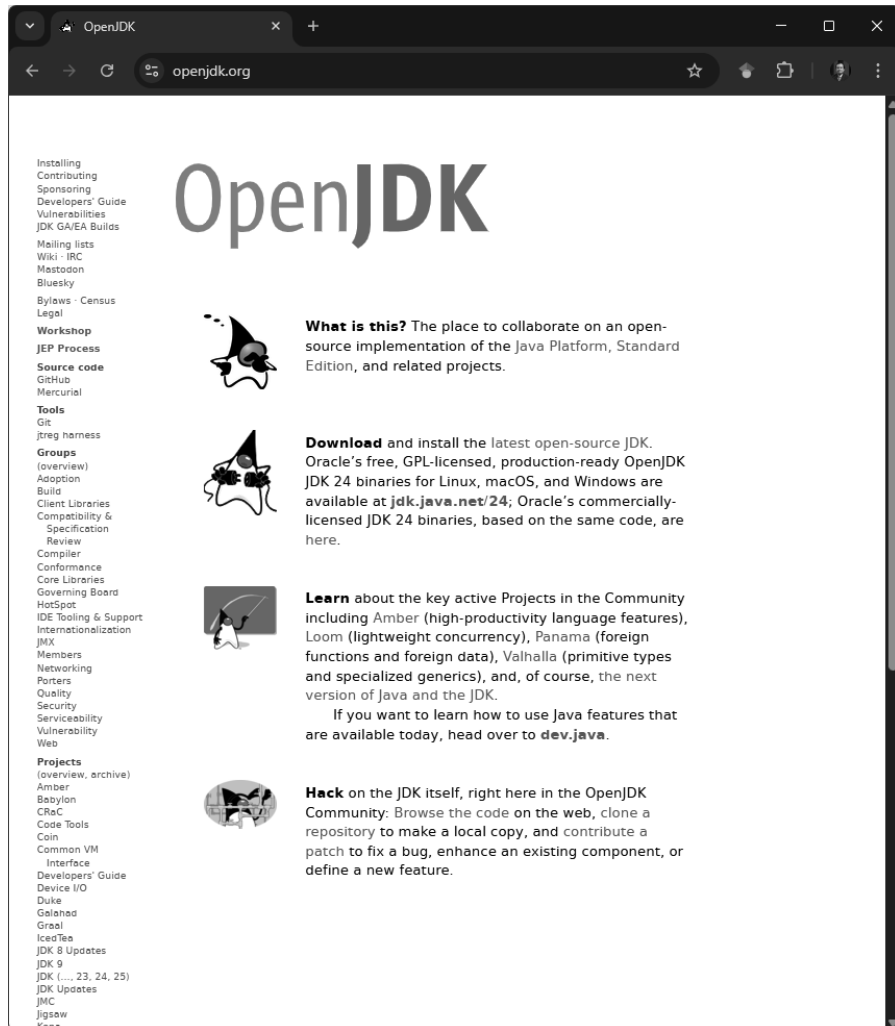
3. Java JDK

Untuk mengembangkan sistem perangkat lunak yang berbasis Java, diperlukan perlengkapan pendukung yaitu Java JDK. Perangkat ini awal mulanya disediakan secara gratis oleh *Sun Microsystems*, tetapi semenjak perusahaan tersebut diakuisisi oleh Oracle maka perangkat pendukung tersebut berubah menjadi berbayar. Untungnya bermunculan versi gratisnya yang didukung oleh banyak perusahaan, misalnya oleh perusahaan Adoptium yang bernama Temurin JDK. Situsnya dapat diakses di <https://adoptium.net/en-GB/temurin/releases/?version=17&os=any&arch=any>.



Gambar 6.6 Situs Temurin JDK oleh Adoptium

Perangkat Temurin JDK dapat diunduh dari situs tersebut dan mengikut petunjuk pemasangan untuk setiap Sistem Operasi yang dimiliki oleh pengguna. Opsi lain adalah memasang versi-versi OpenJDK lainnya dari berbagai sumber, misalnya dari situs OpenJDK di <https://openjdk.org/>.



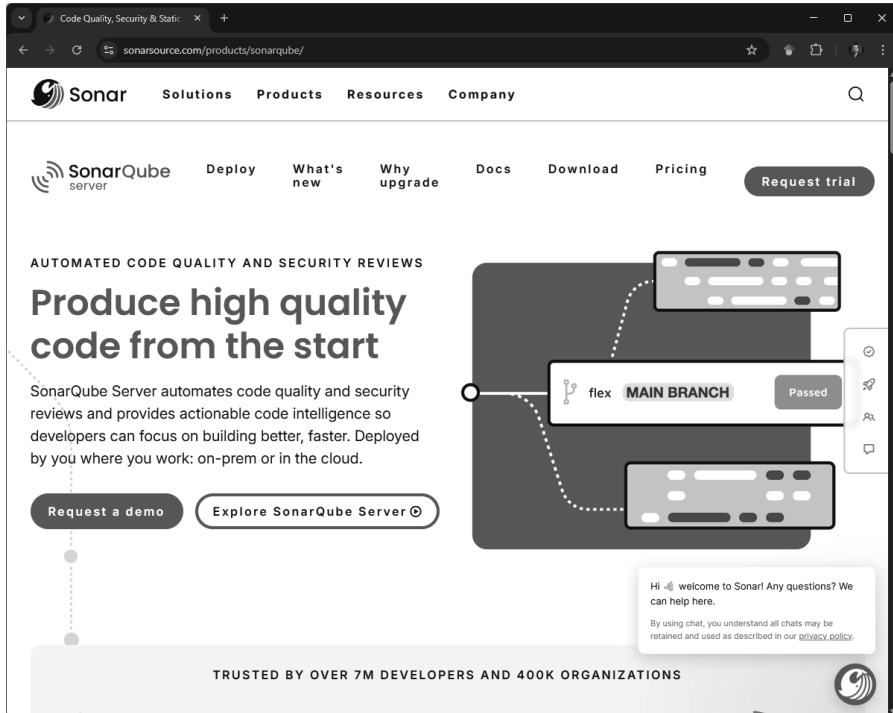
Gambar 6.7 Situs OpenJDK

C. Perangkat Penghitung Metrik

1. SonarQube Community Edition

SonarQube adalah perangkat kompilasi sistem perangkat lunak yang dilengkapi beberapa analisis kode statik yang bekerja untuk beberapa bahasa pemrograman seperti Java, C#, JavaScript,

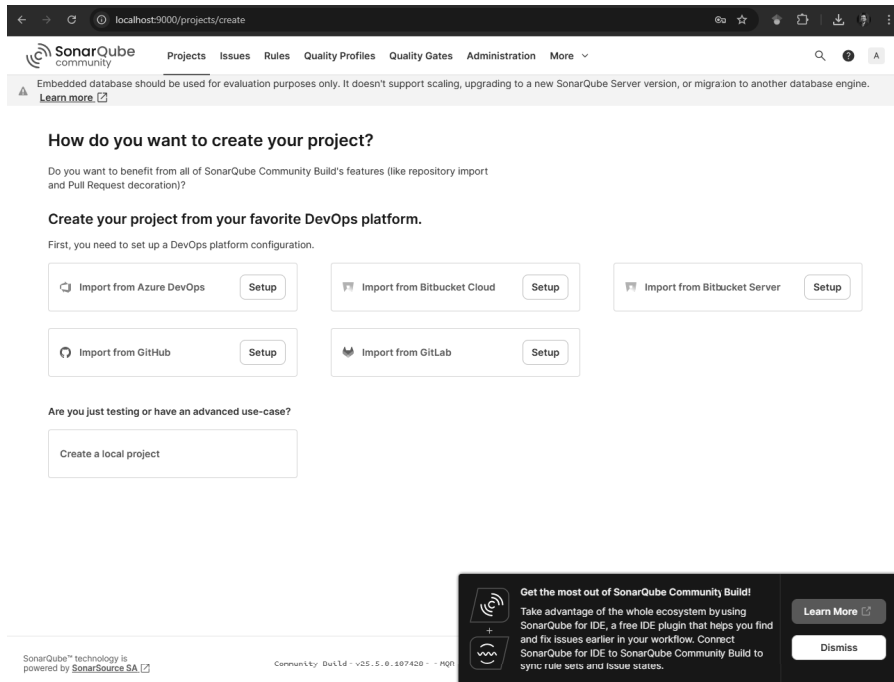
TypeScript, CloudFormation, Teraform, Docker, Kubernetes/Helm Charts, Kotlin, Ruby, Go, Scala, Flex, Python, PHP, HTML, CSS, XML, VB.NET, dan Azure Resource Manager/Bicep. Situs untuk SonarQube adalah <https://www.sonarsource.com/products/sonarqube/> seperti yang ditunjukkan pada Gambar 6.8 berikut ini.



Gambar 6.8 Situs *SonarQube*

Perlu diketahui bahwa penggunaan perangkat ini adalah berbayar. Meski dasar dari aplikasi sonar sendiri berbasis komunitas, tetapi produk yang ditawarkan di situs *SonarQube* sebagian besar berbayar. Terdapat pula versi gratis yang disebut *SonarQube Community Edition* di tautan <https://community.sonarsource.com/>.

Setelah melakukan instalasi pemasangan sesuai petunjuk yang ada di situs, membuka *SonarQube Community Edition* dengan cara mengetikkan alamat <https://localhost:9000> di browser.



Gambar 6.9 Tampilan SonarQube Community Edition

Selain versi komunitas yang gratis dan berbayar, *SonarQube* juga menyediakan versi yang dapat dimasukkan sebagai ekstensi dalam IDE lainnya seperti Visual Studio, Python, VS Code, Eclipse, dan lain-lainnya yang disebut SonarQube for IDE.

2. JavaNCSS

JavaNCSS (*Java Non-Comment Source Statements*) adalah perangkat lunak yang dapat memberikan informasi metrik seperti jumlah baris kode (baris kode tanpa memperhitungkan baris komentar), kompleksitas siklomatik, jumlah kelas, dan jumlah metode. Situs dari JavaNCSS adalah di tautan <https://javancss.github.io/> seperti yang diperlihatkan pada Gambar 6.10 berikut ini.



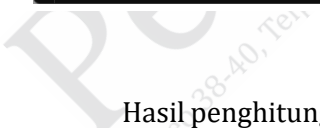
Gambar 6.10 Situs JavaNCSS

JavaNCSS dapat memberikan informasi tentang metrik baris kode dan juga jumlah kelas dengan sebuah perintah baris sederhana pada Terminal. JavaNCSS ini juga dapat diintegrasikan di Ant dengan memodifikasi berkas *build.xml* yang ada pada Apache Ant. Hasil luaran dari perangkat ini adalah berkas berformat XML yang perlu dikonversi untuk dapat dimasukkan dalam basis data seperti MySQL atau MariaDB yang berada di dalam XAMPP.

3. CCCC

CCCC (*C and C++ Code Counter*) merupakan analisis statistika yang lebih optimal untuk bahasa pemrograman C dan C++, tetapi bisa juga bekerja untuk bahasa pemrograman Java. Metrik yang diberikan yaitu: *Lines of Code*, *Non-Commenting Lines of Code*, *Physical Lines*,

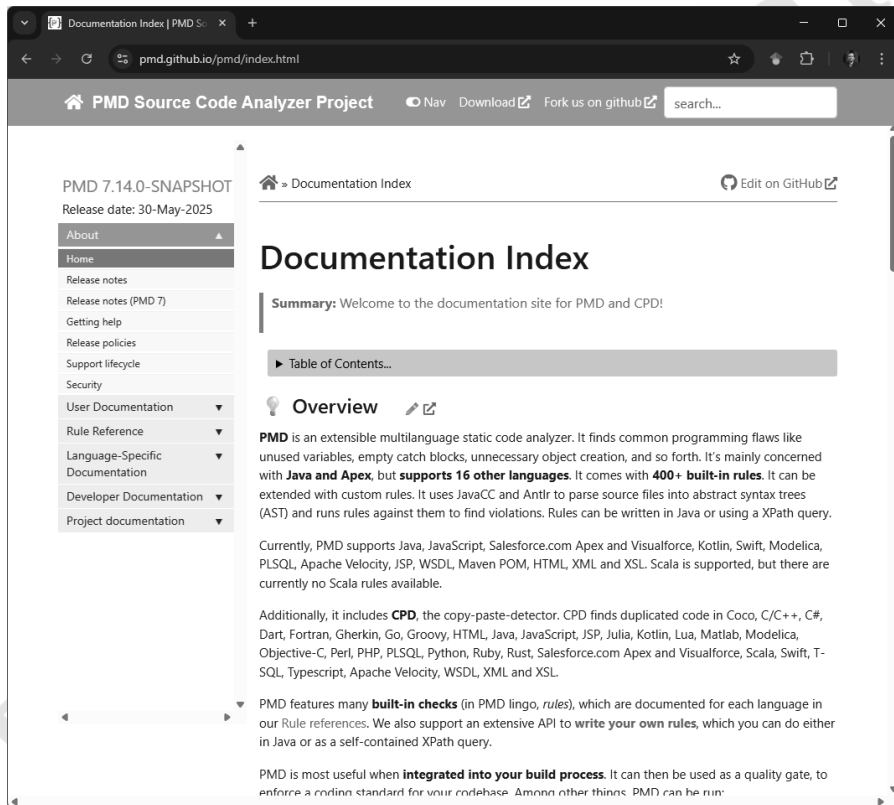
Situs dari CCCC adalah di <https://github.com/sarnold/cccc> seperti yang ditunjukkan pada Gambar 6.11 berikut ini.



penghitun

4. PMD

PMD (*Performance Measurement Dashboard*) adalah perangkat yang dapat memberikan berbagai informasi statistik dan metrik yang dibutuhkan, salah satunya Kompleksitas Siklomatik. Perangkat ini bisa diintegrasikan dengan maven dengan menambahkan pom.xml pada bagian <plugin> di berkas konfigurasi Maven. Situs dari PMD adalah di <https://pmd.github.io/pmd/index.html> seperti yang ditunjukkan pada Gambar 6.12 berikut ini.



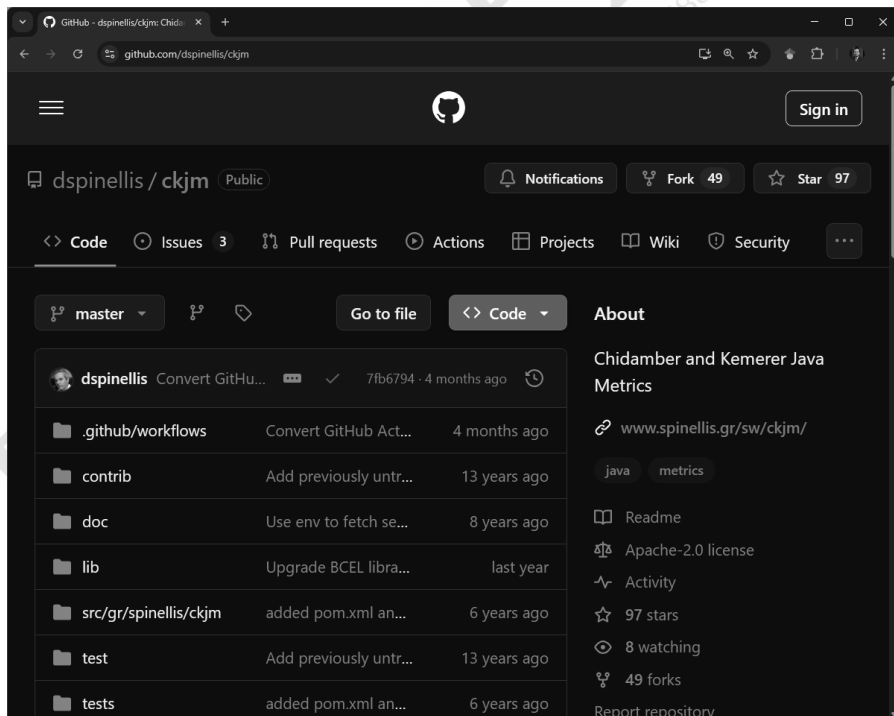
Gambar 6.12 Situs PMD

Perangkat PMD ini dapat memberikan berbagai macam metrik yang berkaitan dengan kinerja jaringan, kualitas layanan, kinerja server dan aplikasi, ketersediaan, keamanan jaringan, metrik pengguna, dan lain-lainnya.

5. CKJM

CKJM adalah perangkat untuk mendeteksi berbagai metrik perangkat lunak berorientasi objek yang ditemukan oleh Chidamber dan Kemerer dengan cara memproses kode byte atau *bytecode* dari kelas-kelas yang terkompilasi dari bahasa pemrograman Java. Perangkat ini diciptakan oleh Diomidis Spinellis dari Departemen of Management Science and Technology, Athens University of Economics and Business di Yunani.

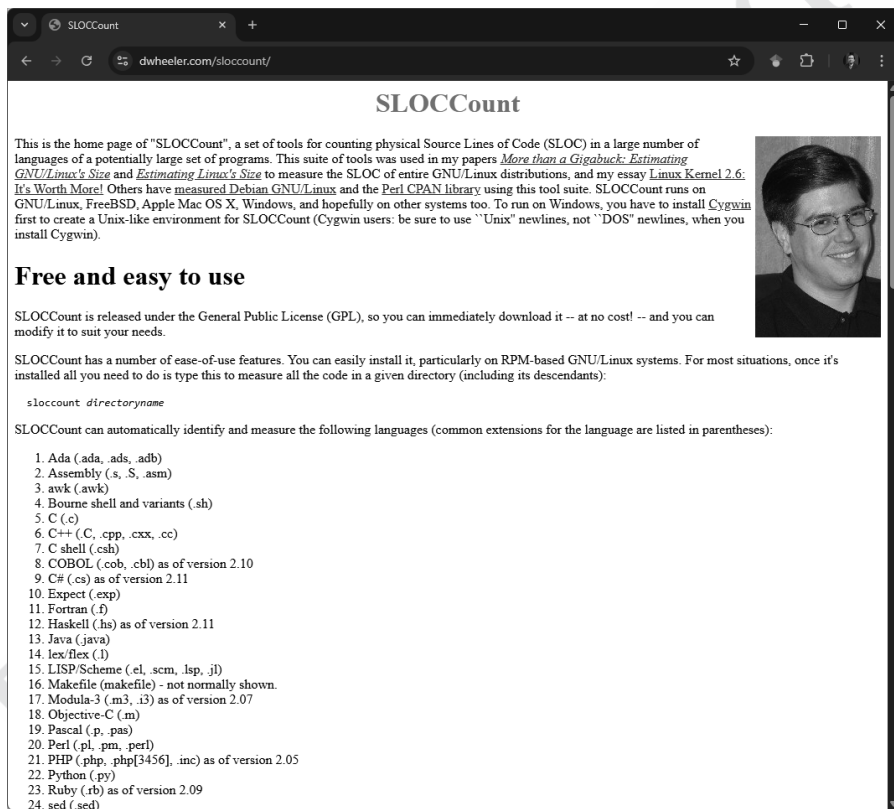
Adapun metrik perangkat lunak yang dihitung adalah: *WMC* (*Weighted Method per Class*), *DIT* (*Depth in Inheritance Tree*), *NOC* (*Number of Children*), *CBO* (*Coupling Between Object Classes*), *RFC* (*Response for a Class*), *LCOM* (*Lack of Cohesion in Methods*), *CA* (*Afferent Coupling*), dan *NPM* (*Number of Public Method*). Situs dari perangkat ini di portal Github adalah <https://github.com/dspinellis/ckjm> seperti yang ditunjukkan pada Gambar 6.13 ini.



Gambar 6.13 Situs Perangkat CKJM

6. SLOCCount

SLOCCount adalah perangkat yang dapat menghitung SLOC dari suatu berkas kode sumber. Perangkat ini mampu mengukur baris kode dari beberapa bahasa pemrograman seperti Ada, Assembly, awk, C, C++, C#, Java, dan lain-lainnya. Selain memberikan informasi tentang baris kode atau SLOC, perangkat ini bahkan mampu melakukan estimasi usaha dan biaya dengan penghitungan COCOMO II versi yang sederhana. Situsnya di <https://dwheeler.com/sloccount/> seperti yang ditunjukkan pada Gambar 6.14 berikut ini.



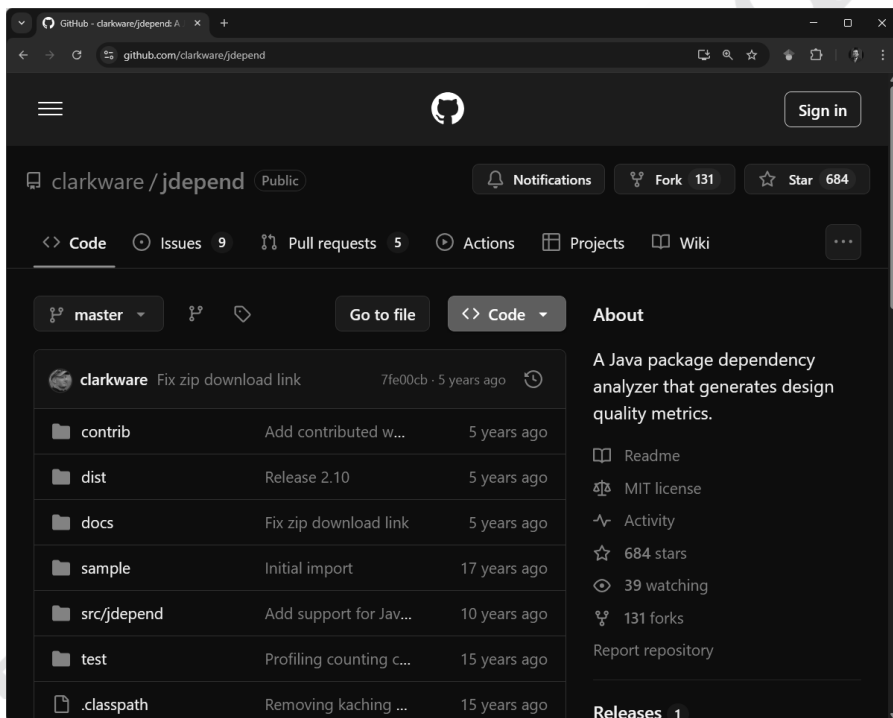
Gambar 6.14 Situs SLOCCount

Penggunaan perangkat lunak ini cukup sederhana karena hanya perlu dijalankan lewat Terminal di Sistem Operasi Linux seperti Ubuntu.

7. JDepend

Jdepend merupakan perangkat untuk mendeteksi berbagai metrik kualitas yang berkaitan dengan ekstensibilitas, penggunaan kembali, dan pemeliharaan untuk bahasa pemrograman Java. Perangkat ini tidak membutuhkan kode sumber dari aplikasi, tetapi hanya memerlukan kelas-kelas terkompilasi (berkas *.class) untuk dapat bekerja.

Situs perangkat ini adalah di <https://github.com/clarkware/jdepend> seperti yang ditunjukkan di Gambar 6.15 ini.

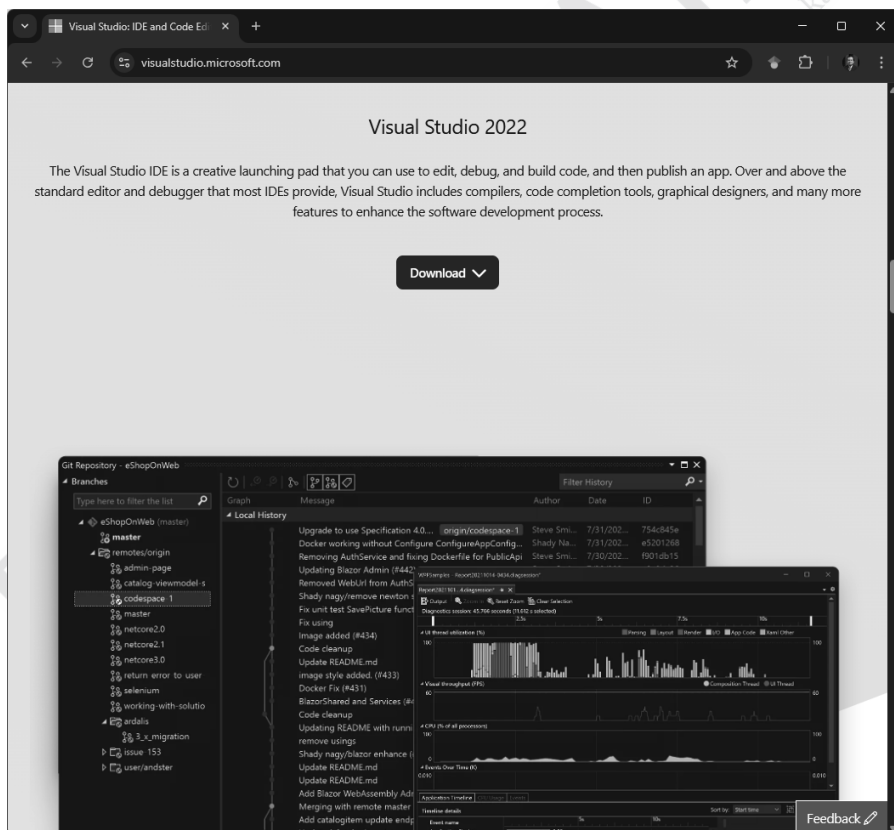


Gambar 6.15 Situs Jdepend

D. Perangkat Lingkungan Pengembangan Terintegrasi (Integrated Development Environment - IDE)

1. Visual Studio

Visual Studio merupakan sebuah lingkungan pengembangan terintegrasi (*Integrated Development Environment* – IDE) pilihan untuk mengembangkan aplikasi di lingkungan Sistem Operasi Windows. Terdapat versi gratis dari perangkat ini yaitu *Visual Studio 2022 Community Edition* yang didapatkan dengan mencari dan memasangnya lewat *Microsoft Store* atau mengunduhnya di situs <https://visualstudio.microsoft.com> seperti pada Gambar 6.16 berikut ini.



Gambar 6.16 Laman Web Situs Visual Studio 2022

Metric perangkat lunak yang bisa didapatkan apabila mengembangkan aplikasi dengan Visual Studio 2022 adalah:

- a. Indeks Pemeliharaan
- b. Kompleksitas Siklomatik
- c. *Depth of Inheritance*
- d. Ketergantungan Kelas
- e. Baris Kode

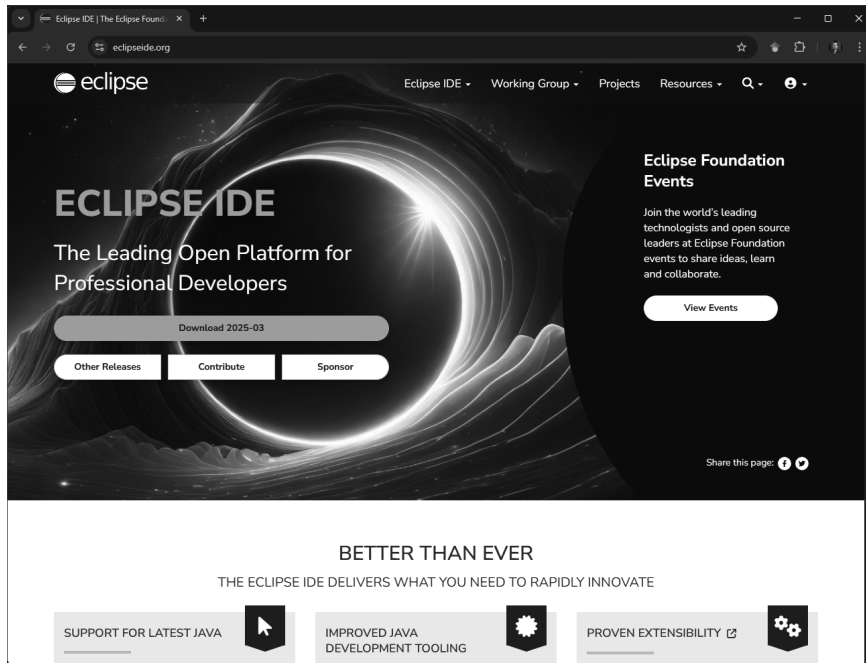
Cara mengakses data tersebut dengan membuka *Solution Explorer* > Klik kanan proyek/solusi > *Calculate Code Metrics*.

2. Eclipse

Eclipse adalah perangkat IDE yang bersifat gratis dan kode terbuka (Open Source). Eclipse IDE merupakan salah satu pilihan utama jika mengembangkan aplikasi berbasis bahasa pemrograman Java, khususnya untuk aplikasi-aplikasi korporasi yang sekarang ini bahasa pemrograman Java masih unggul. Perangkat ini didukung sepenuhnya oleh Eclipse Foundation, yang juga menghasilkan aplikasi *Temurin JDK* yang dibahas sebelumnya.

Metrik yang bisa dihasilkan saat mengembangkan aplikasi dengan menggunakan *Eclipse IDE* adalah:

- a. Baris kode atau LOC
- b. Jumlah metode atau jumlah kelas
- c. Kompleksitas Siklomatik
- d. *Coupling Between Object* (CBO)
- e. *Depth of Inheritance* (DIT)
- f. *Weighted Method per Class* (WMC)



Gambar 6.17 Situs Eclipse IDE

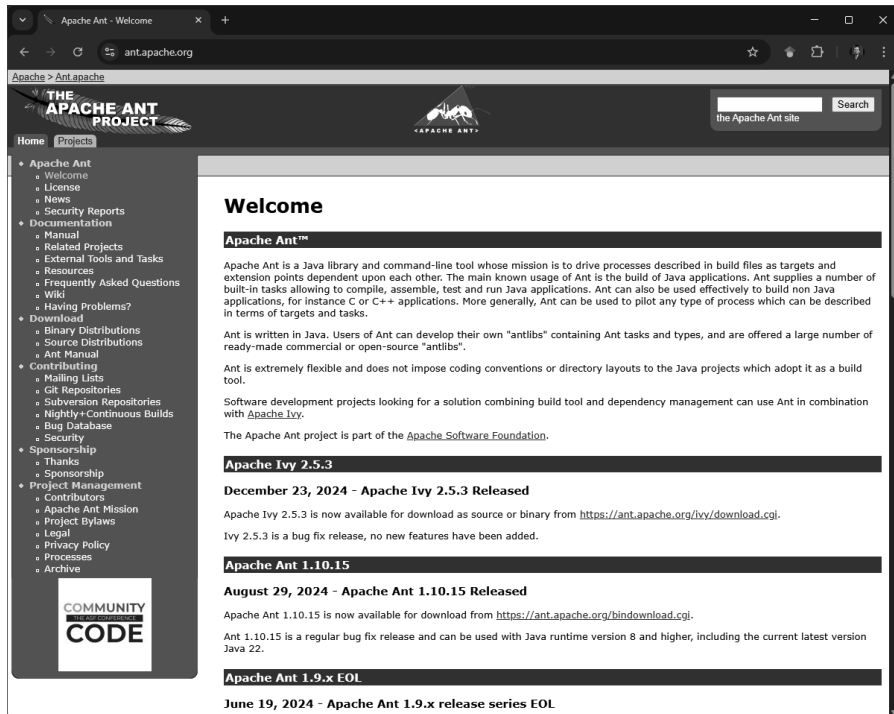
Dengan menambahkan plugin metrik beberapa metrik tambahan juga didapatkan, yaitu:

- Eclipse Metric Plugin: Response For a Class (RFC), Lack of Cohesion in Methods (LCOM), dan Fan-in/Fan-out*
- PMD: Kompleksitas Siklomatik, panjang metode yang eksekusi, dan kode sumber yang mati (*dead code*)

Situs dari Eclipse IDE dapat diakses di tautan <https://eclipseide.org/>.

3. Apache Ant

Apache Ant adalah perangkat untuk membangun aplikasi khususnya yang berbasis bahasa pemrograman Java, meskipun bisa juga dipergunakan untuk membangun aplikasi berbasis bahasa pemrograman C ataupun C++. Situs untuk Apache Ant adalah di <https://ant.apache.org/> seperti yang ditunjukkan pada Gambar 6.18 berikut ini.

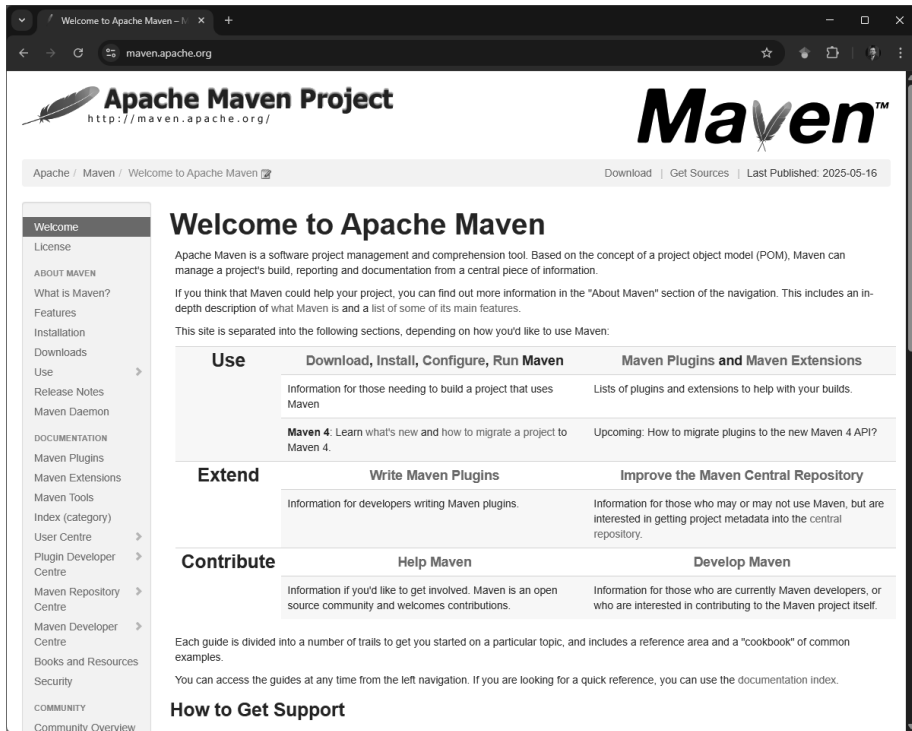


Gambar 6.18 Situs Apache Ant

Agar bisa mengumpulkan berbagai metrik yang diperlukan, maka Apache Ant ini perlu ditambahkan perangkat tambahan yaitu Sonar IDE yang untuk versi Java.

4. Apache Maven

Apache Maven merupakan perangkat untuk kompilasi sistem perangkat lunak yang lebih modern daripada *Apache Ant*. Untuk mengeksekusinya dilakukan dengan mengetikkan perintah “mvn” di Terminal (untuk Sistem Operasi Linux misalnya Ubuntu). Situs untuk *Apache Maven* adalah <https://maven.apache.org/> seperti yang ditunjukkan pada Gambar 6.19 berikut ini.

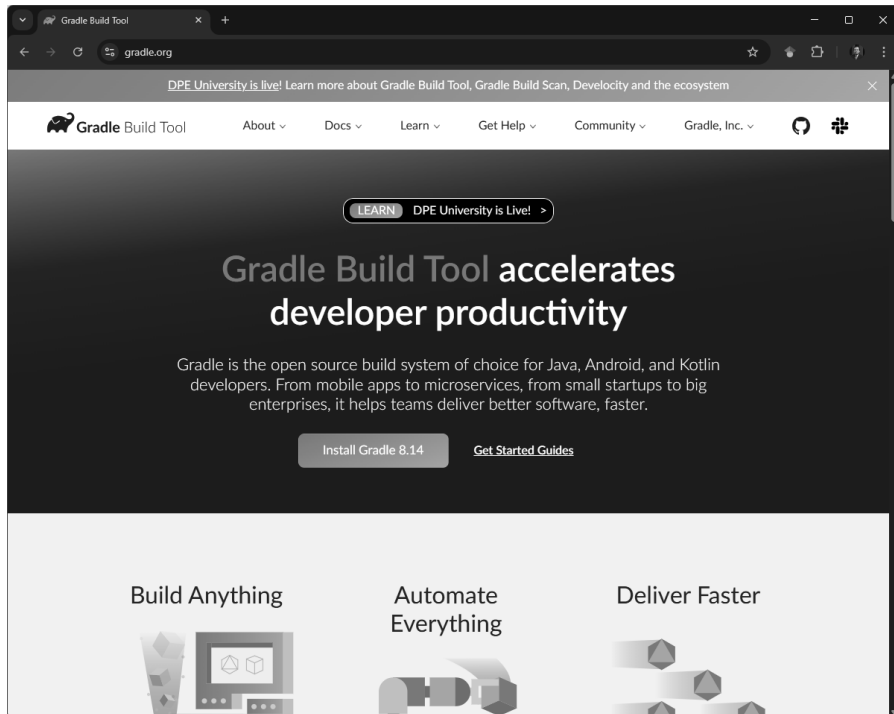


Gambar 6.19 Situs Apache Maven

Salah satu keunggulan *Apache Maven* adalah kemampuannya untuk ditambahkan berbagai macam tambahan atau plugin agar dapat melakukan berbagai macam pengukuran metrik. Caranya dengan memodifikasi berkas `pom.xml` yang berada dalam folder Apache Maven. Contoh plugin yang dapat ditambahkan yang berhubungan dengan metrik perangkat lunak: *jacoco-maven-plugin* (mengukur cakupan kode), *spotbugs-maven-plugin* (mendeteksi bug potensial), *maven-dependency-plugin* (menganalisis ketergantungan), dan lain-lainnya.

5. Gradle

Gradle adalah perangkat kompilasi sistem perangkat lunak mirip seperti *Apache Maven* yang bisa dipakai untuk bahasa pemrograman Java, Kotlin, maupun Android. Situsny adalah di <https://gradle.org/> seperti yang ditunjukkan pada Gambar 6.20 berikut ini.



Gambar 6.20 Situs Gradle

Cara penggunaan Gradle dapat dilihat di tautan https://docs.gradle.org/current/userguide/getting_started_eng.html. Perangkat Gradle ini bisa menghasilkan beberapa metrik yang berkaitan dengan kinerja build, dependensi, sumber daya, kualitas kode, struktur proyek, dan lain-lainnya. Beberapa metrik tersebut akan tersedia apabila ditambahkan plugin seperti *PMD*, *Jacoco*, dan lain-lainnya.

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

3. Bagi profesional: sebagai bahan referensi awal untuk dapat menerapkan berbagai strategi untuk pengukuran perangkat lunak di proyek perangkat lunak yang sedang berjalan. Buku ini dapat dijadikan acuan awal sebelum mencari referensi yang lebih detail dan lebih mendalam.

Buku ini tentu saja masih jauh dari sempurna dan memerlukan berbagai perbaikan ke depan. Untuk dapat menyajikan konsep pengukuran perangkat lunak secara komprehensif dalam sebuah buku adalah hal yang cukup menantang. Menyatukan berbagai macam komponen pengukuran untuk berbagai fase dalam pengembangan sistem perangkat lunak tentu saja memerlukan riset yang lebih mendalam lagi. Kami sebagai penulis tentu saja terbuka untuk berbagai masukan agar di edisi mendatang beberapa bagian dari buku ini yang masih belum sempurna dapat untuk diperbaiki agar manfaat buku ini dapat lebih luas bagi banyak kalangan.

Akhir kata, penulis menyampaikan permohonan maaf yang sebesar-besarnya apabila terdapat kesalahan ataupun kelalaian yang mungkin ditemukan. Kami akan senantiasa berusaha untuk memperbaiki diri secara terus-menerus. Terima kasih atas kesediaan membaca buku ini yang merupakan suatu kehormatan yang sangat mendalam bagi kami.

Daftar Pustaka

Albrecht A.J., "Measuring Application Development Productivity", IBM's SHARE/GUIDE Conference 1979.

Alsmadi I., Magel K., "OpenSource Evolution Analysis", 22nd IEEE International Conference on Software Maintenance 2006.

Anderson D.J., "Kanban: Successful Evolutionary Change for Your Techology Business", Blue Hole Press, 2010.

Boehm B.W., Abts C., Brown A.W., Chulani S., Clark B.K., Horowitz E., Madacy R., Reifer D, Steece B., "Software Cost Estimation with COCOMO II", 1st Edition, Prentice Hall, 2000.

Brooke J., "SUS: A Quick and Dirty Usability Scale", Taylor Francis, 1986.

Chidamber S.R., Kemerer C.F., "A Metrics suite for Object Oriented Design", IEEE Transaction on Software Engineering, Vol. 20 No. 6 June 1994, pp 476-493.

DeMarco T., "Controlling Software Projects: Management, Measurement, and Estimation", 1982.

UEQ-online, "A fast and reliable questionnaire to measure the User Experience of interactive product. Available in more than 30 languages. Easy to use due to rich supplementary material". UEQ-online website. Diakses: 22 Mei 2025.

Halstead M. H., "Elements of Software Science", Elsevier, 1977.

Hitz M., Montazeri B., "Measuring Coupling and Cohesion in Object-Oriented Systems", Proceeding International Symposium on Applied Corporate Computing, Oct. 25-27 1995, Monterrey, Mexico, 75-76, 197, 78-84.

- IBM Security, "Cost of a Data Breach Report 2023", IBM, 2023.
- IEEE Computer Society, "Software Engineering Body of Knowledge", tersedia: <https://www.computer.org/education/bodies-of-knowledge/software-engineering>, 2024.
- McCabe T., "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. Se-2, No. 4, December 1976, pp. 308–320.
- Meyer B., Oriol M., & Schoeller B., "Software Engineering: Lecture 17-18: Estimation Techniques and Software Metrics", (<https://slideplayer.com/slide/7747800/>), diakses: 20 Mei 2025.
- Micallef M., "Software Measurement", Lecture Slides, tersedia: <https://slideplayer.com/slide/17387950/>, diakses: 2022.
- Oman P., Hagemester J., "Metrics for Assessing a Software System's Maintainability", Proceedings of the Internasional Conference on Software Maintenance, 1992.
- Perlis A., Sayward F., Shaw M., "Software Metrics: An Analysis and Evaluation", Penerbit Mit Pr, 1981.
- Project Management Institute, "A Guide to the Project Management Body of Knowledge (PMBOK Guide)", 7th Edition, 2021.
- Riberu L.H., Emanuel A.W.R., "Vulnerability Testing and Analysis Using OWASP Top 10 on Academic Information System at University XYZ", Prosiding 1st International Conference of Adisutjipto on Aerospace Electrical Engineering and Informatics (ICAAEEI) 2024, 2024.
- OWASP.org, "OWASP Top 10", OWASP Foundation (<https://owasp.org/www-project-top-ten/>), diakses: 20 Mei 2025.
- QSM, Function Point Language Table (<http://www.qsm.com/resources/function-point-languages-table>), diakses 19 Mei 2025.
- Schrepp M., Hinderks A., Thomaschewski, "Applying the User Experience Questionnaire (UEQ) in Different Evaluation Scenarios. Design, User Experience, and Usability. Theories, Methods, and Tools for Designing the User Experience, 2014, Springer, pp 383–392.

Schrepp M., Hinderks A., Thomaschewski, "Design and Evaluation of Short Version of the User Experience Questionnaire (UEQ-S)", *Internasional Journal of Interactive Multimedia and Artificial Intelligence*, 4(6), 2017, pp 103–108.

Schwalbe K., "An Introduction to Project Management: Predictive, Agile, and Hybrid Approaches", 7th Edition, 2021.

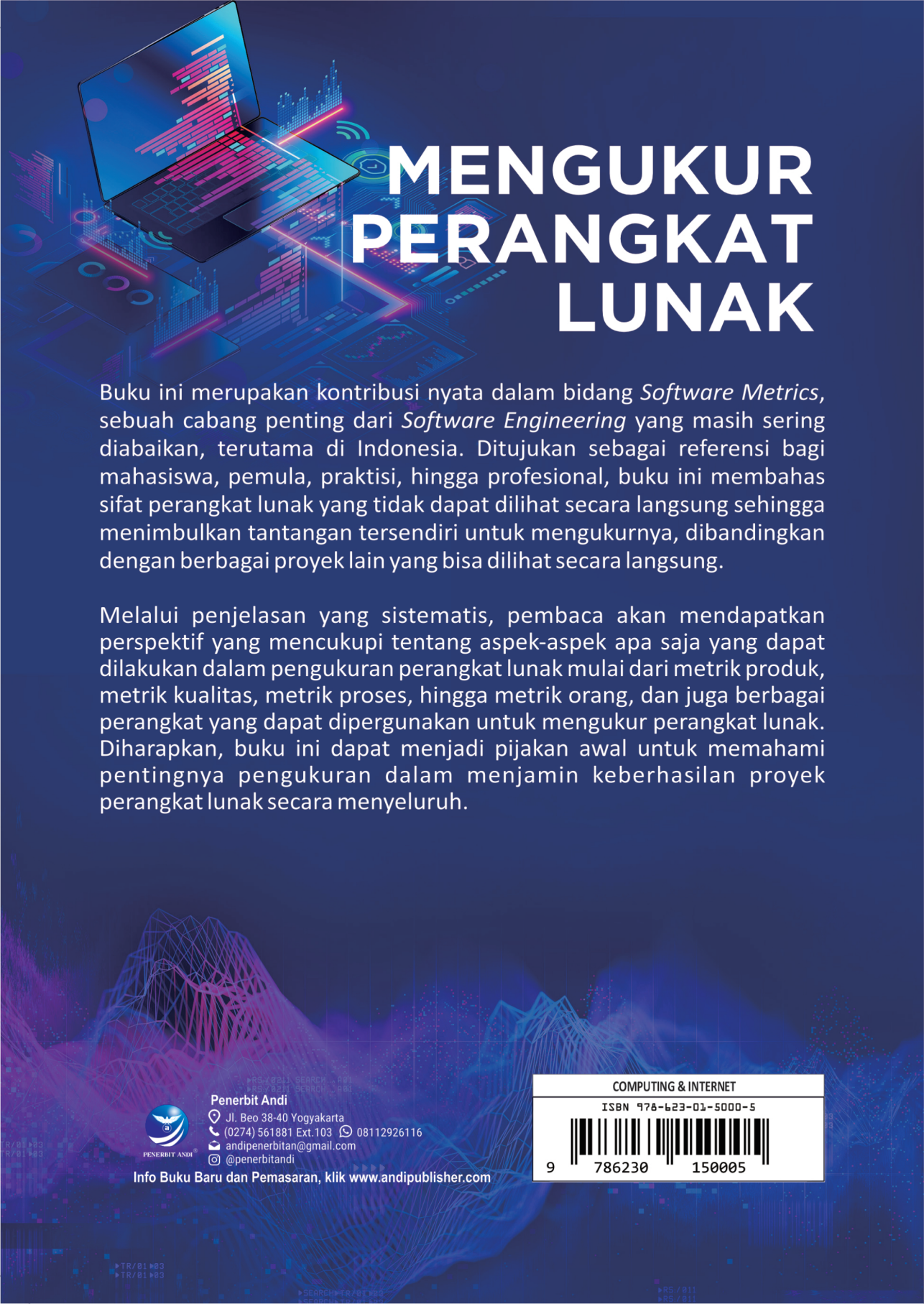
Scrum.org, What is Scrum (<https://www.scrum.org/resources/what-scrum-module>), diakses: 20 Mei 2025.

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281

Penerbit Andi
Jl. Beo 38-40, Telp. (0274) 561881 (Hunting), Fax. (0274) 588282 Yogyakarta 55281



MENGUKUR PERANGKAT LUNAK

Buku ini merupakan kontribusi nyata dalam bidang *Software Metrics*, sebuah cabang penting dari *Software Engineering* yang masih sering diabaikan, terutama di Indonesia. Ditujukan sebagai referensi bagi mahasiswa, pemula, praktisi, hingga profesional, buku ini membahas sifat perangkat lunak yang tidak dapat dilihat secara langsung sehingga menimbulkan tantangan tersendiri untuk mengukurnya, dibandingkan dengan berbagai proyek lain yang bisa dilihat secara langsung.

Melalui penjelasan yang sistematis, pembaca akan mendapatkan perspektif yang mencakup tentang aspek-aspek apa saja yang dapat dilakukan dalam pengukuran perangkat lunak mulai dari metrik produk, metrik kualitas, metrik proses, hingga metrik orang, dan juga berbagai perangkat yang dapat dipergunakan untuk mengukur perangkat lunak. Diharapkan, buku ini dapat menjadi pijakan awal untuk memahami pentingnya pengukuran dalam menjamin keberhasilan proyek perangkat lunak secara menyeluruh.



Penerbit Andi

Jl. Beo 38-40 Yogyakarta
(0274) 561881 Ext.103 08112926116
andipenerbitan@gmail.com
@penerbitandi

Info Buku Baru dan Pemasaran, klik www.andipublisher.com

COMPUTING & INTERNET

ISBN 978-623-01-5000-5



9

786230

150005