PENGUJIAN NONFUNGSIONAL DENGAN PENDEKATAN *MCCALL'S FACTOR* PADA PERSPEKTIF *PRODUCT REVISION*

Disusun Oleh:

N a m a : Sheilla Fajriah Muthmainnah

NIM : 19523140

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2023

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGUJIAN NONFUNGSIONAL DENGAN PENDEKATAN *MCCALL'S FACTOR* PADA PERSPEKTIF *PRODUCT REVISION*

TUGAS AKHIR

Disusun Oleh:

N a m a	: Sheilla Fajriah Muthmainnah
NIM	: 19523140

Yogyakarta, 13 Juli 2023

Pembimbing,

(Hanson Prihantoro Putro, S.T., M.T.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGUJIAN NONFUNGSIONAL DENGAN PENDEKATAN *MCCALL'S FACTOR* PADA PERSPEKTIF *PRODUCT REVISION*

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Teknik Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 13 Juli 2023

Tim Penguji

Hanson Prihantoro Putro, S.T., M.T. _____

Anggota 1

Dr. Raden Teduh Dirgahayu, S.T., M.Sc. _____

Anggota 2

Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. _____

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Sheilla Fajriah Muthmainnah

NIM : 19523140

Tugas akhir dengan judul:

PENGUJIAN NONFUNGSIONAL DENGAN PENDEKATAN *MCCALL'S FACTOR* PADA PERSPEKTIF *PRODUCT REVISION*

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 13 Juli 2023

(Sheilla Fajriah Muthmainnah)

HALAMAN PERSEMBAHAN

Puji syukur saya panjatkan kepada Allah SWT, yang telah memberikan kesehatan, rahmat dan hidayahNya, serta menghadirkan orang-orang baik yang selalu memberikan dukungan sehingga saya dapat menyelesaikan tugas akhir ini. Dengan mengucap rasa syukur kepada Allah SWT, saya persembahkan Laporan Tugas Akhir yang telah saya susun untuk Mama dan Papa tersayang, terima kasih atas doa terbaik yang tak pernah putus, serta semangat dan motivasi yang selalu diberikan untuk menyelesaikan tugas akhir ini. Serta untuk orang-orang terdekat yang selalu memberikan dukungan dan motivasi, tidak lupa kepada Bapak Hanson Prihantoro Putro, S.T., M.T., selaku Dosen Pembimbing Tugas Akhir yang telah membimbing saya sehingga tugas akhir ini dapat terselesaikan, kepada Bapak Kholid Haryono, S.T., M.Kom., selaku Dosen Pembimbing Akademik yang telah membimbing selama masa perkuliahan.

HALAMAN MOTO

“Barangsiapa yang belum pernah merasakan pahitnya menuntut ilmu walau sesaat, ia akan menelan hinanya kebodohan sepanjang hidupnya”
(Imam Asy-Syafi’i)

“Bangun kesuksesan dari kegagalan. Keputusasaan dan kegagalan adalah dua batu loncatan yang paling baik menuju kesuksesan.”
(Dale Carnegie)

"Jalan awal terbaik untuk mewujudkan segala impian anda adalah bangun dan bangkit dari tempat tidur."
(Paul Valéry)

“Jangan menilai saya dari kesuksesan, tetapi nilai saya dari seberapa sering saya jatuh dan berhasil bangkit kembali”
(Nelson Mandela)

“Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya”
(QS. Al-Baqarah: 286)

KATA PENGANTAR

Segala puji syukur saya panjatkan kepada Allah SWT yang telah memberikan rahmat dan ridhoNya, sehingga saya dapat menyelesaikan pembuatan Tugas Akhir yang berjudul "Pengujian Nonfungsional dengan Pendekatan *McCall's Factor* pada Perspektif *Product Revision*" dengan baik.

Tujuan dari pembuatan Tugas Akhir ini yaitu sebagai salah satu syarat untuk memperoleh gelar sarjana di Jurusan Informatika Universitas Islam Indonesia. Tanpa doa, dukungan, dan bimbingan dari berbagai pihak, penelitian ini tidak akan selesai. Oleh karena itu, saya ingin menyampaikan rasa terima kasih kepada:

1. Allah SWT yang telah memberikan kesehatan dan kekuatan, serta rahmat dan hidayahNya.
2. Kedua orang tua dan adik tercinta yang senantiasa medoakan dan memberi dukungan semangat selama masa perkuliahan serta untuk menyelesaikan Tugas Akhir.
3. Bapak Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D., selaku Ketua Program Studi Informatika – Program Sarjana Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Bapak Kholid Haryono, S.T., M.Kom., selaku Dosen Pembimbing Akademik
5. Bapak Hanson Prihantoro Putro, S.T., M.T., selaku Dosen Pembimbing Tugas Akhir yang telah membimbing saya dengan sangat sabar dan membantu ketika saya mengalami kendala selama mengerjakan tugas akhir.
6. Seluruh Bapak/Ibu dosen Jurusan Informatika yang telah memberikan banyak ilmu yang bermanfaat selama saya menjalani masa perkuliahan.
7. Seluruh *developer* Aplikasi Ivent yang telah berkenan mengizinkan saya untuk menggunakan Aplikasi Ivent sebagai objek dalam penelitian tugas akhir.
8. Sahabat-sahabat saya yaitu Karina Khoiriyah Pertiwi, Azzahra Nisa Lutfia, Zerlina Tsaabitah, dan Obhin Atilariz Huda. Terima kasih atas segala doa dan dukungan yang senantiasa diberikan, sebagai tempat berbagi keluh kesah dan kenangan indah selama masa perkuliahan.

Harapan penulis terhadap penelitian tugas akhir ini yaitu dapat bermanfaat bagi semua pihak yang membaca dan mempelajari materi pengujian *McCall's Factor* dari isi laporan tugas akhir ini terutama pada perspektif *Product Revision*.

Yogyakarta, 13 Juli 2023

(Sheilla Fajriah Muthmainnah)

SARI

Perangkat lunak merupakan kumpulan program yang berisikan perintah atau prosedur yang digunakan untuk mengolah informasi. Dalam pengembangannya, banyak ditemukan permasalahan pada suatu produk perangkat lunak. Misalnya seperti banyak terdapat *bug* dalam pengembangan sebuah fitur. Oleh karena itu, diperlukan sebuah pengujian untuk melihat perangkat lunak dapat berfungsi dengan baik atau tidak. Salah satu metode yang digunakan dalam pengujian perangkat lunak secara manual adalah menggunakan metode *McCall's Factor*.

Penelitian ini bertujuan untuk menguji perangkat lunak dengan menggunakan pendekatan *McCall's Factor* pada aspek *product revision* terhadap aplikasi Ivent. Proses pengujian pada penelitian ini memiliki enam tahapan, yaitu mengidentifikasi jenis tes yang dilakukan, lalu menentukan metrik pengujian dan perumusan pada setiap karakteristik yang telah ditentukan. Kemudian melakukan pengaturan lingkungan uji untuk menentukan syarat *software* telah benar-benar siap untuk diuji dan selanjutnya dilakukan pengujian. Setelah pengujian dilakukan tahap selanjutnya akan dilakukan penutupan siklus uji.

Dari perhitungan implementasi, didapatkan nilai *Software Quality* untuk faktor *Maintainability* sebesar 26%, faktor *Flexibility* sebesar 13%, dan *Testability* sebesar 14%. Sedangkan nilai dari hasil pengujian perangkat lunak pada fase desain untuk faktor *Maintainability* sebesar 14%, faktor *Flexibility* sebesar 7.5%, dan faktor terakhir yaitu *Testability* sebesar 14%. Jika nilai fase desain dan implementasi digabung, maka faktor *maintainability* mendapatkan nilai sebesar 20%, *flexibility* sebesar 10.3%, dan *testability* sebesar 20.5%. Perolehan nilai *Software Quality* tersebut dapat menjadi acuan bagi *developer* dalam melakukan perbaikan dan pengembangan kualitas perangkat lunak.

Kata kunci: Software quality, McCall's Factor, Product Revision, Ivent.

GLOSARIUM

Developer	Seseorang yang memiliki keahlian dalam mengembangkan suatu perangkat lunak atau aplikasi komputer, dapat terlibat dalam pengembangan aplikasi web, mobile, desktop, atau perangkat lunak lainnya.
E-Commerce	Aktifitas jual beli produk dan layanan antara penjual dan pembeli secara *online* melalui media internet.
Closure	Tahapan terakhir dari *software testing life cycle* (STLC) untuk mengevaluasi seluruh tahapan yang telah dilakukan sebelumnya.
Activity Diagram	Jenis diagram dalam pemodelan perangkat lunak yang digunakan untuk menggambarkan alur atau urutan aktivitas dalam suatu proses.
Use Case Diagram	Jenis diagram dalam pemodelan perangkat lunak yang digunakan untuk menggambarkan interaksi antara aktor-aktor eksternal dan sistem perangkat lunak.
Test Plan	Dokumen rencana pengujian yang merinci langkah-langkah yang akan diambil untuk proses pengujian sistem perangkat lunak, jenis pengujian yang akan dilakukan, sumber daya yang dibutuhkan, dan kriteria kelulusan pengujian.
Test Case	Sebuah deskripsi rinci mengenai langkah-langkah, kondisi yang harus terpenuhi, dan hasil yang digunakan untuk menguji sebuah perangkat lunak.
PHP	*PHP* (*Hypertext Preprocessor*) merupakan salah satu bahasa pemrograman paling populer yang umumnya digunakan untuk pengembangan aplikasi web dinamis.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Secara umum perangkat lunak adalah semua perintah yang digunakan untuk mengolah informasi yang ada. Perangkat lunak dapat berupa program atau prosedur yang dapat dieksekusi. Keberhasilan pengembangan perangkat lunak tergantung pada manajemen keseluruhan proyek perangkat lunak. Namun dibalik kesuksesan tersebut, ada juga proses pengujian perangkat lunak atau dikenal dengan *software testing* yang sangat penting untuk dilakukan oleh *software tester*. Pengujian perangkat lunak adalah proses untuk menunjukkan bahwa perangkat lunak berfungsi seperti yang diharapkan (Naik & Tripathy, 2011).

Mendeteksi keberadaan *bug* dalam perangkat lunak dan memeriksa apakah perangkat lunak yang dikembangkan telah memenuhi persyaratan yang diharapkan dan memastikan produk perangkat lunak bebas cacat merupakan hal yang diperlukan selama pengujian suatu produk perangkat lunak. Proses ini bertujuan dalam penemuan kesalahan dan kelalaian, yang tidak diharapkan selama proses pengembangan sistem (Pohan & Kom, 2018). Proses pengujian juga bertujuan untuk menunjukkan kesesuaian fungsi-fungsi perangkat lunak terhadap spesifikasinya. Oleh karena itu, mekanisme pengujian perangkat lunak yang sesuai standar diperlukan agar perangkat lunak dapat berfungsi sebagaimana mestinya.

Dalam kehidupan sehari-hari banyak ditemukan permasalahan pada sebuah produk perangkat lunak. Misal seperti fitur yang terdapat pada perangkat lunak saat ini sudah tertinggal dengan perangkat lunak yang lain, terdapat bug dalam pengembangan sebuah fitur baru pada sistem perangkat lunak, dan lain sebagainya. Pengujian nonfungsional pada perangkat lunak merupakan hal yang jarang dilakukan oleh pengembang. Padahal pengujian ini juga tidak kalah penting dengan pengujian fungsional. Bayangkan jika suatu perangkat lunak tidak dapat dilakukan perawatan, tidak dapat diuji, bahkan tidak fleksibel, maka perangkat lunak tersebut tidak bisa berkembang dan tidak akan bertahan lama.

Pengujian nonfungsional merupakan jenis pengujian perangkat lunak yang memverifikasi aspek-aspek nonfungsional (kinerja, kegunaan, keandalan, keamanan, muatan, dan kompatibilitas dari aplikasi perangkat lunak yang tidak dilakukan pada pengujian fungsional (Hamilton, 2020). Jika pengujian fungsional menjelaskan apa yang harus dilakukan program seperti fitur, maka pada pengujian nonfungsional akan menjelaskan bagaimana cara kerjanya. Pengujian nonfungsional dapat dilakukan dengan 2 cara, yaitu secara otomatis dan

manual. Pengujian secara otomatis akan dilakukan dengan bantuan aplikasi *automated testing*. Sedangkan pengujian secara manual dilakukan tanpa bantuan aplikasi melainkan dengan tangan sendiri.

McCall mengklasifikasikan tiga hal penting yang mempengaruhi kualitas perangkat lunak dari sudut pandang pengembangan produk yang dikenal sebagai *product perspective* (Siregar & Arif, 2018). Menurut *McCall* tiga hal penting dari perspektif produk yaitu: 1) Transisi Produk (*Product Transition*) kemampuan suatu produk untuk beradaptasi terhadap lingkungan baru; 2) Operasional Produk (*Product Operations*) kemampuan suatu produk yang berkaitan dengan karakteristik kinerja suatu aplikasi perangkat lunak; 3) Revisi Produk (*Product Revision*) yaitu kemampuan suatu produk untuk mengalami sebuah perubahan.

Pada perspektif revisi produk, pengujian berfokus pada kemampuan suatu produk untuk mengalami sebuah perubahan. Perangkat lunak yang dirancang dan dikembangkan dengan baik, dapat dengan mudah diubah sesuai dengan kebutuhan. Revisi produk terdiri dari *maintainability*, *testability*, *flexibility*. Dalam *maintainability testing*, kemampuan untuk memelihara dan memperbaiki suatu sistem. Artinya, semakin bagus *maintainability* maka semakin cepat perbaikan pada suatu sistem. *Flexibility* pada perangkat lunak dapat berarti banyak hal, tetapi ketika digunakan untuk menggambarkan keseluruhan sistem, biasanya mengacu pada kemampuan solusi untuk beradaptasi dengan kemungkinan atau perubahan di masa depan. *Testability testing* produk perangkat lunak adalah tentang proses dari sebelum pengujian. Dengan melakukan pengujian-pengujian tersebut maka dapat meningkatkan kemudahan penggunaan, efisiensi, perawatan, dan mengurangi resiko biaya. Kecenderungan dan kepercayaan pengguna sehubungan dengan produk perangkat lunak selalu dipengaruhi oleh kualitas nonfungsional, maka dari itu melakukan pengujian nonfungsional sangat penting.

Dengan dilakukan pengujian nonfungsional pada aspek *Product Revision*, memastikan kesesuaian dengan kebutuhan fungsionalitas teknis *software* dan dapat meningkatkan efisiensi *software* berikut juga aktivitas pemeliharaannya. Penelitian ini akan membahas pengujian pada perspektif *Product Revision*. Pengujian tahap revisi produk dipilih karena membantu dalam memastikan bahwa perangkat lunak yang direvisi memenuhi standar kualitas yang diharapkan. Hal ini membantu dalam mengidentifikasi dan memperbaiki cacat atau masalah yang mungkin muncul setelah melakukan perubahan pada produk. Penelitian ini bertujuan untuk mengetahui cara pengujian perangkat lunak dengan menggunakan pendekatan *McCall's Factor* pada aspek *Product Revision* terhadap aplikasi *e-commerce* Ivent. Ivent dipilih sebagai kasus dalam penelitian karena aplikasi ini belum pernah dilakukan pengujian secara nonfungsional. Adanya

penelitian ini diharapkan *developer* dapat terbantu dalam memperbaiki kerusakan dan kekurangan yang terdapat pada perangkat lunak. Dengan demikian, kualitas perangkat lunak juga dapat berkembang lebih baik dengan peningkatan performa implementasinya.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan diatas, rumusan masalah yang dapat diangkat yaitu bagaimana melakukan pengujian perangkat lunak pada aplikasi *e-commerce* berbasis web Ivent menggunakan pendekatan *McCall's Factor* pada aspek *Product Revision*?

1.3 Batasan Masalah

Batasan masalah yang ditentukan dalam penelitian ini sebagai berikut:

a. Pengujian perangkat lunak dilakukan menggunakan pendekatan *McCall's Factor* pada aspek *Product Revision*.
b. Pengujian perangkat lunak dilakukan pada aplikasi *e-commerce* berbasis web 'Ivent'.

1.4 Tujuan Penelitian

Tujuan yang ingin dicapai pada penelitian ini adalah untuk mengetahui cara pengujian perangkat lunak dengan menggunakan pendekatan *McCall's Factor* pada perspektif *Product Revision* terhadap aplikasi *e-commerce* berbasis web Ivent, untuk mengetahui apakah sistem sudah menunjukkan bahwa sistem dapat berjalan seperti yang diharapkan atau tidak.

1.5 Manfaat

Manfaat yang akan didapat dari penelitian ini adalah:

a. Mengetahui cara menguji perangkat lunak pada perspektif *Product Revision* menggunakan pendekatan *McCall's Factor.*
b. Mengetahui kesalahan apa saja yang terdapat pada perangkat lunak selama pengujian.
c. Membantu pengembang aplikasi untuk menemukan cara agar aplikasi Ivent mudah untuk dipelihara dan fleksibel.

1.6 Metodologi Penelitian

Ada 6 tahapan proses pengujian yang harus dilaksanakan secara sistematis dan terencana pada aplikasi berbasis website Ivent. Langkah-langkahnya adalah sebagai berikut:

a. *Requirement Analysis*

Software requirement yang sudah didapatkan dari *stakeholder* akan dianalisis oleh tim QA. Menganalisa apa saja yang dapat diuji secara fungsional dan non-fungsional.

b. *Test Planning*

Tahapan ini tim QA menyiapkan rencana seperti *tools* yang akan digunakan serta estimasi waktu dan sumber daya yang akan digunakan.

c. *Test Case Development*

Tahapan ini menjadi acuan dalam pengujian untuk membuat *test case, test data, test script*. Agar sistem yang diuji dapat memenuhi ketentuan dan standar tertentu.

d. *Environment Setup*

Tahap ini menentukan syarat *software* yang diuji apakah dapat berjalan dengan baik dan benar-benar siap.

e. *Test Execution*

Pengujian dilakukan berdasarkan *test plan* dan *test case* yang telah dibuat pada tahapan sebelumnya.

f. *Test Cycle Closure*

Tahap ini akan mengidentifikasi dan mengevaluasi seluruh tahapan yang telah dilakukan sebelumnya.

1.7 Sistematika Penelitian

Sistematika penulisan pada laporan penelitian Tugas Akhir yang dijelaskan secara singkat mencakup:

a. **BAB I PENDAHULUAN**

Pada bab ini menguraikan mengenai latar belakang penelitian, rumusan masalah, batasan dalam penulisan penelitian, tujuan dilakukannya penelitian, manfaat dari penelitian, metodologi secara umum, dan sistematika penulisan.

b. **BAB II KAJIAN PUSTAKA**

Bab ini membahas teori dan hasil penelitian terdahulu baik dari jurnal, buku, maupun artikel. Dimana mencakup metode atau cara yang digunakan serta alat atau teknologi yang digunakan.

c. **BAB III METODOLOGI PENELITIAN**

Bab metodologi penelitian menjelaskan mengenai metode-metode yang dilakukan seperti prosedur penelitian dan pengujian perangkat lunak. Bab in merupakan inti dari pengerjaan penelitian yang dilakukan.

d. **BAB IV HASIL DAN PEMBAHASAN**

Setelah melakukan pengujian pada perangkat lunak dengan metode yang telah diputuskan, maka pada bab ini akan membahas hasil temuan dari pengujian yang telah dilakukan tersebut.

e. **BAB V KESIMPULAN DAN SARAN**

Bab ini memuat tentang kesimpulan yang didapatkan dari penelitian mengenai pengujian perangkat lunak yang telah dilakukan. Selain itu juga berisi poin-poin saran untuk penelitian terkait pengujian perangkat lunak selanjutnya.

BAB II
KAJIAN PUSTAKA

2.1 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses di mana kita menunjukkan bahwa perangkat lunak berjalan sesuai apa yang diharapkan (Naik & Tripathy, 2011). Pengujian memiliki peran penting dalam mencapai kualitas suatu produk perangkat lunak. Memproduksi produk berkualitas telah diakui sebagai faktor kunci dalam keberhasilan jangka panjang. Mendeteksi keberadaan *bug* dalam perangkat lunak dan memeriksa apakah perangkat lunak yang dikembangkan telah memenuhi persyaratan yang diharapkan dan memastikan produk perangkat lunak bebas cacat merupakan hal yang diperlukan selama pengujian suatu produk perangkat lunak.

Keberadaan *bug* dalam perangkat lunak dapat menyebabkan produk perangkat lunak tidak berjalan sesuai fungsinya, adanya bug pada produk perangkat lunak akan berbahaya dan berpotensi menyebabkan kerugian. Dengan melakukan pengujian dapat diidentifikasi lebih awal dan dapat diselesaikan sebelum produk perangkat lunak siap pakai. Produk perangkat lunak yang diuji dengan benar akan memastikan keamanan, keandalan, dan kinerja tinggi, akan menghasilkan keuntungan, efisiensi waktu, dan kepuasan pelanggan. Oleh karena itu, mekanisme pengujian perangkat lunak yang standar diperlukan agar perangkat lunak dapat berfungsi sebagaimana mestinya.

Tujuan dari pengujian produk perangkat lunak yaitu mengidentifikasi kesalahan, kelalaian, atau persyaratan yang hilang dari persyaratan yang sebenarnya. Pengujian produk perangkat lunak akan mendatangkan manfaat, *cost-effective* (penghematan biaya), *security* (keamanan), *product quality* (kualitas produk), *customer satisfaction* (kepuasan pelanggan).

a. *Cost Effective*

   Menguji produk perangkat lunak tepat waktu akan membantu menghemat biaya pengeluaran untuk jangka panjang. Jika bug diketahui lebih awal saat pengujian maka biaya yang keluar akan lebih murah untuk memperbaikinya.

b. *Security*

   Pengujian produk perangkat lunak keamanan merupakan manfaat yang paling sensitif dan rentan, hal ini disebabkan orang-orang akan mencari produk yang terpercaya.

c. *Product Quality*

Persyaratan paling penting dalam suatu produk perangkat lunak merupakan kualitas dari produk. Pengujian akan memastikan produk yang sampai kepada pelanggan berkualitas.

d. *Customer Satisfaction*

Memberikan kepuasan kepada pelanggan merupakan tujuan utama dari setiap produk, memastikan pengalaman terbaik bagi pengguna dengan melakukan pengujian UI/UX.

2.2 *Non-Functional Requirements*

Terdapat pada sebuah artikel (Chung & do Prado Leite, 2009) menjelaskan bahwa di bidang persyaratan perangkat lunak, istilah persyaratan nonfungsional telah digunakan untuk merujuk pada masalah yang tidak terkait dengan fungsionalitas perangkat lunak. Namun, penulis yang berbeda menggambarkan perbedaan ini dalam definisi informal dan tidak setara. Setelah berargumen mengenai definisi persyaratan nonfungsional maka dapat didefinisikan istilah ini sebagai tambahan di mana persyaratan nonfungsional merupakan spesifikasi yang menggambarkan kemampuan dan kendala operasi sistem yang meningkatkan fungsionalitasnya. Pada hal ini mungkin kecepatan, keamanan, kendala, dan lainnya.

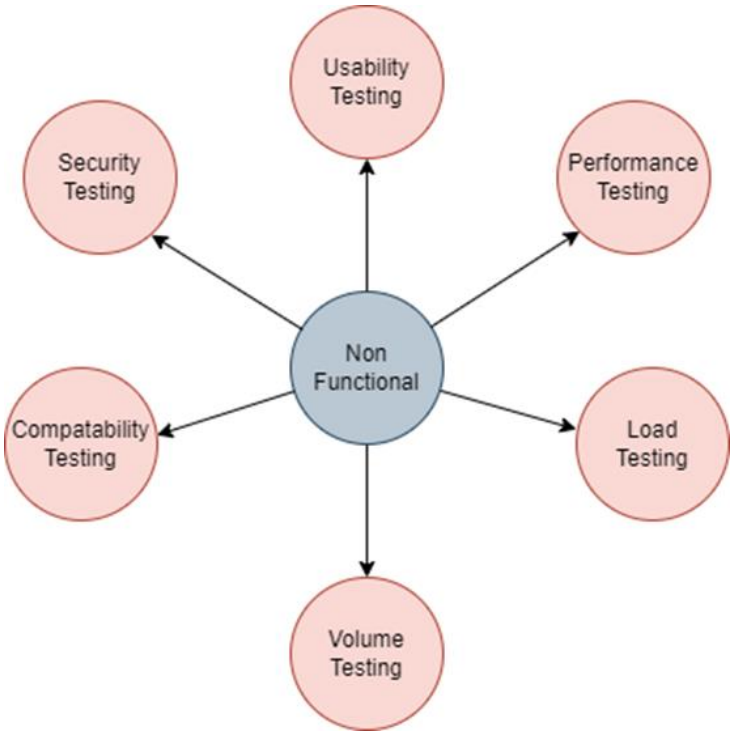
Dari sumber dan panduan yang berbeda menggunakan istilah yang berbeda. Misalnya, pada kerangka standar ISO/IEC 25000 mendefinisikan persyaratan nonfungsional sebagai persyaratan kualitas sistem dan persyaratan kualitas perangkat lunak. BABOK, salah satu sumber pengetahuan utama bagi analis bisnis, menciptakan istilah persyaratan nonfungsional (NFR), yang saat ini merupakan definisi yang paling umum.

Jenis-jenis paling umum yang akan dimasukkan dalam daftar periksa persyaratan nonfungsional. Beberapa diantaranya yaitu, performa dan skalabilitas (*performance and scalability*), portabilitas dan kompatibilitas (*portability and compatability*), keandalan (*reliability*), ketersediaan (*availability*), pemeliharaan (*maintainability*), keamanan (*security*), lokalisasi (*localization*), dan kegunaan (*usability*).

2.3 Pengujian Nonfungsional Perangkat Lunak

Pengujian nonfungsional merupakan jenis pengujian perangkat lunak yang memverifikasi aspek-aspek nonfungsional (kinerja, kegunaan, keandalan, keamanan, muatan, dan kompatibilitas) seperti yang digambarkan pada Gambar 2. 1, dari aplikasi perangkat lunak yang tidak dilakukan pada pengujian fungsional (Hamilton, 2020). Ini memeriksa apakah operasi pada sistem diperlukan atau tidak. Jika pengujian fungsional menjelaskan apa yang harus dilakukan program, maka pada pengujian nonfungsional akan menjelaskan bagaimana

cara kerjanya. Penelitian yang dilakukan oleh Fata Nidaul Khasanah menguji aplikasi telepon darurat berbasis android secara nonfungsional. Penelitian tersebut menggunakan metode BETA untuk pengujiannya. Pengujian tersebut dilakukan menggunakan survei dengan menyebarkan kuesioner yang berisikan penilaian pada setiap butir pertanyaan dengan menggunakan skala *likert* ke beberapa responden. Terdapat empat kriteria yang akan diuji dalam penelitian ini yaitu, *usefulness*, *ease of use*, *ease of learning*, dan *satisfaction*. Hasil dari survei diketahui bahwa nilai persentase yang diperoleh berdasarkan kriteria penilaian yaitu *usefulness* 81%, *ease of use* 80%, *ease of learning* 83%, *satisfaction* 81% (Khasanah, 2018). Dari nilai persentase tersebut, aplikasi telepon darurat berbasis android telah berjalan sesuai dengan hasil yang diharapkan. Berdasarkan penelitian yang telah dilakukan diketahui bahwa metode penelitian nonfungsional tersebut hanya menguji dari sisi pengguna dan tidak dari segala sisi. Oleh karena itu, tidak dapat diketahui fleksibilitas sistem tersebut jika diberikan fitur baru dan dilakukan pemeliharaan.



Gambar 2. 1 Aspek-aspek pengujian nonfungsional perangkat lunak

Dalam pengujian nonfungsional memiliki beberapa karakteristik di mana suatu pengujian nonfungsional harus terukur, sehingga tidak ada ruang untuk karakteristik subjektif misal lebih baik, terbaik dan lainnya. Pada pengujian nonfungsional sangat penting untuk memprioritaskan

persyaratan atau permintaan, hal ini berpengaruh pada kepuasan klien. Selanjutnya, selalu pastikan atribut kualitas diidentifikasi dengan benar dalam Rekayasa Perangkat Lunak.

Pengujian nonfungsional memiliki 2 cara untuk dilakukan, yaitu dengan cara otomatis dan cara manual. Pengujian dengan cara otomatis dilakukan dengan menggunakan bantuan aplikasi *automated testing*. Teknik yang digunakan dalam bantuan aplikasi *automated testing* ini ada 2, yaitu *white box testing* dan *black box testing*. *White-box testing* merupakan pengujian untuk menguji perangkat lunak dengan menganalisis dan mempelajari struktur internal dan kode perangkat lunak. Tidak seperti *black-box testing* yang berfokus pada aliran *input* dan *output* pada perangkat lunak. Sedangkan untuk pengujian secara manual merupakan pengujian yang dilakukan tanpa bantuan aplikasi melainkan dengan tangan sendiri. Pengujian ini mempelajari apakah sebuah fitur telah berjalan dengan semestinya dengan memverifikasi fitur yang tertera pada dokumen persyaratan. Pengujian perangkat lunak secara manual memiliki berbagai metode salah satunya yaitu metode *McCall's Factor*. Pada metode *McCall's* mengukur kualitas dari sebuah perangkat lunak memiliki cara dengan menilai dan membandingkan faktor-faktor kualitas perangkat lunak, serta kriteria yang berkualitas pada sebuah perangkat lunak.

2.4 McCall's Factor

Metode *McCall's Factor* adalah model pengujian tertua yang dikembangkan pada tahun 1996 yang dikembangkan oleh McCall Richards dan Walter pada tahun 1977. Pengujian dengan metode *McCall's Factor* dipilih karena menggunakan skala dan bobot untuk memberikan penilaian kualitatif terhadap faktor-faktor yang dinilai, hal ini dapat membantu mengidentifikasi area yang memerlukan perbaikan atau peningkatan. Dalam sebuah artikel (Siregar & Arif, 2018)*McCall* mengklasifikasikan tiga hal penting yang memengaruhi kualitas perangkat lunak dari sudut pandang pengembangan produk yang dikenal sebagai perspektif produk (*product perspective*). Menurut *McCall* tiga hal penting dari perspektif produk sebagai berikut, seperti yang terlihat pada Gambar 2. 2:

a. Transisi Produk (*Product Transition)*

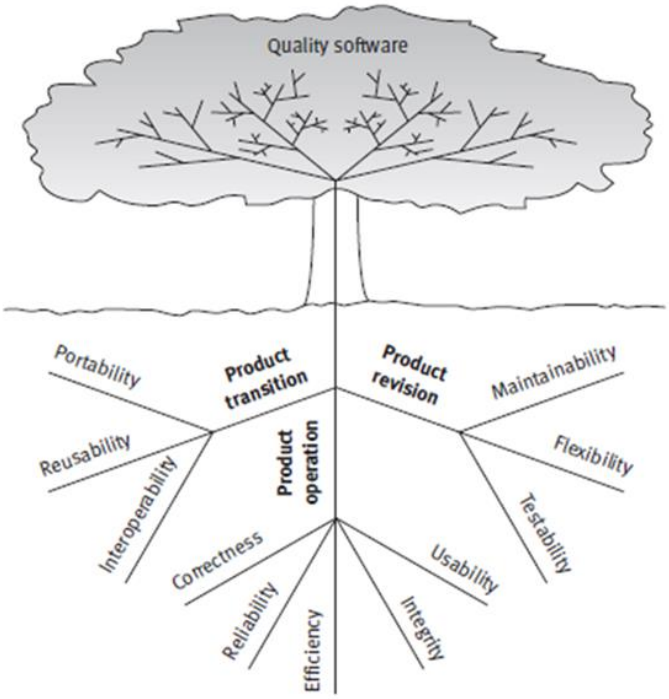
   Kemampuan suatu produk untuk beradaptasi terhadap lingkungan baru. Bagaimana perangkat lunak dapat dijalankan pada platform yang beragam. Faktor yang berhubungan dengan transaksi produk adalah *portability*, *reusability*, *interoperability*.

b. Kinerja produk (*Product Operations*)

Kemampuan suatu produk yang berkaitan dengan karakteristik kinerja suatu aplikasi perangkat lunak. Faktor yang berhubungan dalam kinerja produk terdiri dari *integrity, correctness, usability, efficiency*.

c. Revisi produk (*Product Revision*)

Kemampuan suatu produk untuk mengalami sebuah perubahan. Perangkat lunak yang dirancang dan dikembangkan dengan baik, dapat dengan mudah diubah sesuai dengan kebutuhan. Revisi produk terdiri dari *maintainability, testability, flexibility*.



Gambar 2. 2 *McCall's Software Quality Factors*

Sumber: (Weerasinghe, 2022)

Dalam artikel yang sama (Siregar & Arif, 2018) rumus yang ditunjukkan pada Persamaan (2. 1) digunakan secara tidak langsung untuk mengukur faktor kualitas suatu produk perangkat lunak.

$$Fq = w_1.n_1 + w_2.n_2 + \cdots + w_n.n_n \quad (2.1)$$

di mana:

Fq = Faktor *software quality*

w_1= Bobot yang tergantung pada produk dan kepentingannya

n_1 = Metrik yang mempengaruhi faktor *software quality*

Persamaan (2. 1) memiliki bobot yang artinya nilai atau mutu yang tergantung pada kepentingan karakteristik dari sebuah faktor. Metrik dari Persamaan (2. 1) merupakan nilai yang didapatkan dari perhitungan setiap aturan karakteristik. Nilai dari faktor *software quality* yang telah didapatkan, nantinya akan diubah ke dalam bentuk nilai persentase (%). Nilai dalam bentuk persentase tersebut akan memberikan nilai untuk kualitas-kualitas karakteristik yang telah diuji. Rumus yang digunakan untuk menghitung besarnya nilai persentase menggunakan Persamaan (2. 2).

$$\% = \frac{Rerata\ Faktor\ Software\ Quality}{Nilai\ Maksimum} . 100\% \qquad (2.2)$$

Nilai maksimum yang terdapat pada Persamaan (2. 2) bernilai satu, nilai maksimum tersebut didapatkan dari perhitungan metrik pada metode *McCall's Factor* yang memiliki nilai maksimum bernilai satu. Penelitian ini akan membahas pengujian pada perspektif *Product Revision*. Oleh karena itu, pembahasan berikutnya akan difokuskan pada *maintainability*, *flexibility*, dan *testability testing*.

2.4.1 *Maintainability Testing*

Pemeliharaan perangkat lunak mencakup semua implementasi setelah perubahan pada entitas perangkat lunak. Pemeliharaan mengacu pada kemudahan atau kesulitan dari upaya yang diperlukan untuk membuat perubahan. Sebelum perubahan dapat dilakukan pada perangkat lunak entitas, perangkat lunak tersebut harus dipahami sepenuhnya. Setelah perubahan selesai, perangkat lunak entitas yang diubah juga harus diuji secara menyeluruh. Untuk alasan ini, pemeliharaan perangkat lunak dapat dilihat sebagai tiga atribut: dapat dimengerti, dapat diubah, dan dapat diuji. Pada sebuah artikel menjelaskan (Sharma & Baliyan, 2011) beberapa model telah diusulkan oleh *McCall*, Boehm dan ISO 9126. Tabel di bawah ini perbandingan antara atribut pemeliharaan yang tersedia untuk sistem berbasis perangkat lunak. Namun, tidak semua atribut mungkin sesuai untuk sistem berbasis komponen. Tabel 2. 1 Analisis Komparatif Aspek Pemeliharaan Model Mutu menjelaskan analisis komparatif dari sub atribut yang diusulkan oleh model kualitas Boehm, model kualitas *McCall*, dan model kualitas ISO 9126.

Tabel 2. 1 Analisis Komparatif Aspek Pemeliharaan Model Mutu

Quality Attribute	Sub Attribute	Model Boehm	Model McCall	Model ISO 9126
Maintainability	Testability	Y	N	Y
	Understandability	Y	N	N
	Modifiability	Y	N	N
	Stability	N	N	Y
	Analyzability	N	N	Y
	Changeability	N	N	Y
	Conciseness	N	Y	N
	Self Descriptiveness	N	Y	N
	Modularity	N	Y	N
	Compliance	N	N	N
	Simplicity	N	Y	N
	Consistency	N	Y	N

Huruf "Y" menunjukkan jika sub karakteristik ikut disertakan dalam model, sedangkan huruf "N" menunjukkan sub karakteristik tidak diikutsertakan dalam model. Dari perbandingan ini, karakteristik yang relevan untuk sistem berbasis komponen ditentukan. Berdasarkan metode *McCall's Factor*, *maintainability* dibagi menjadi 5 karakteristik. Karakteristik tersebut adalah *conciseness, self descriptive*, *modularity*, *simplicity, consistency*. Setiap karakteristik memiliki aturan pengujian perangkat lunak yang berbeda.

Pada sebuah siklus pengembangan perangkat lunak terdapat beberapa alasan mengapa diperlukan menghitung nilai *maintainability*, nilai dari *maintainability* dapat membantu dalam memutuskan apakah suatu perangkat lunak mudah dirawat atau perlu perancangan ulang (Chung & do Prado Leite, 2009). Semakin besar nilai dari perhitungan *maintainability*, maka mengindikasikan bahwa kode program memiliki tingkat *maintainability* yang baik. Sehingga membuat kode menjadi lebih dipahami dan akan lebih mudah dalam pencarian dan perbaikan (*Bugs)*.

2.4.2 *Flexibility Testing*

Fleksibilitas sangat penting untuk menghasilkan perangkat lunak yang berkualitas. Fleksibilitas perangkat lunak dapat berarti banyak hal, tetapi ketika digunakan untuk menggambarkan keseluruhan sistem, biasanya mengacu pada kemampuan solusi untuk beradaptasi dengan kemungkinan atau perubahan di masa depan. Fleksibilitas teknologi sering dievaluasi dari sudut pandang teknis semata, dengan hanya mempertimbangkan teknologi. Teknologi juga dapat secara tidak langsung mempengaruhi fleksibilitas melalui hubungan

antara organisasi pendukung teknologi dan pengguna, penanganan permintaan perubahan dan karakteristik respon lainnya (Nelson et al., 1997).

Fleksibilitas teknologi memiliki 2 dimensi (Nelson et al., 1997) yaitu fleksibilitas struktural dan fleksibilitas proses. Dimensi fleksibilitas struktural mencerminkan kemampuan desain teknologi untuk beradaptasi dengan perubahan proses bisnis dan untuk secara aktif berintegrasi dengan teknologi. Sedangkan dimensi fleksibilitas proses mengacu pada kemampuan rekayasa proses dan manajemen teknis untuk beradaptasi dengan perubahan. Untuk menangkap fleksibilitas teknologi, 2 dimensi yang disebutkan sebelumnya harus diukur.

Flexibility tidak memiliki analisis komparatif karena aspek fleksibility hanya terdapat dalam model kualitas *McCall's Factor*. Berdasarkan metode pengujian *McCall's Factor* memiliki 3 karakteristik, yaitu *self descriptive*, *generality*, dan *expandability*. Setiap karakteristik dalam *flexibility testing* pada perangkat lunak memiliki aturan masing-masing. Perhitungan setiap karakteristik dalam menguji perangkat lunak menggunakan rumus metrik.

2.4.3 *Testability Testing*

Pengujian perangkat lunak merupakan salah satu aspek penting dan yang paling mendasar untuk memastikan kualitas dari perangkat lunak (Mona, 2022). Kemampuan uji produk perangkat lunak adalah tentang proses dari sebelum pengujian. Kemampuan uji produk relatif terkait dengan kriteria pengujian dan artefak yang sedang diuji. Kerangka kerja mendefinisikan karakteristik pada kemampuan uji produk dan metrik yang sesuai untuk konteks kemampuan uji tertentu. Kerangka kerja juga menentukan pernyataan untuk menempatkan karakteristik dan metrik dalam hubungan yang terkait satu sama lainnya. Tabel 2. 2 Analisis Komparatif Aspek Pengujian Model Mutu menjelaskan analisis komparatif dari sub atribut yang diusulkan oleh model kualitas Boehm dan model kualitas *McCall* (Musa & Alkhateeb, 2013).

Tabel 2. 2 Analisis Komparatif Aspek Pengujian Model Mutu

Quality Attribute	Sub Attribute	Model Boehm	Model McCall
Testability	Accessibility	Y	N
	Communicativeness	Y	N
	Structuredness	Y	N
	Instrumentation	N	Y
	Self Descriptiveness	Y	Y
	Modularity	N	Y
	Simplicity	N	Y

Berikut adalah beberapa karakteristik untuk menentukan *testability testing* pada perangkat lunak berdasarkan metode *McCall's Factor*. Terdapat 4 karakteristik dalam *testability testing*, yaitu *self descriptive*, *modularity*, *simplicity*, dan *instrumentation*. Setiap karakteristik memiliki aturan perhitungan yang berbeda dalam pengujian perangkat lunak. Perhitungan setiap karakteristik menggunakan rumus metrik kemudian dicari nilai rata-rata.

2.4.4 *Conciseness*

Karakteristik *Conciseness* yang menunjukkan "Apakah sistem dibangun dengan jumlah baris kode yang relatif singkat?". *Conciseness* merupakan suatu karakteristik perangkat lunak yang menyediakan implementasi fungsi dengan jumlah kode yang minimum. Bagian *conciseness* menggunakan metrik yang didasarkan pada konsep panjang *Halstead*. Metrik *Halstead* ini merupakan sebuah alat komersial yang berguna untuk menghitung baris kode perangkat lunak dengan menghitung token dan menentukan mana yang merupakan operator dan operan. Jumlah total kemunculan dari operator dan operan dapat diketahui melalui Persamaan (2.3) berikut (McCall et al., 1977).

$$N_o = N_1 + N_2 \tag{2.3}$$

di mana:

N_1 = Total penggunaan semua operator pada sebuah kriteria

N_2 = Total penggunaan semua operan pada sebuah kriteria

Kemudian, untuk mencari estimasi panjang kode dari sebuah program dapat menggunakan rumus seperti Persamaan (2.4) sebagai berikut.

$$N_c = n_1 \log_2 n_1 + n_2 \log_2 n_2 \quad (2.4)$$

di mana:

n_1 = Jumlah operator yang berbeda.

n_2 = Jumlah operan yang berbeda.

Maka dari itu, metrik dapat dinormalisasi seperti Persamaan (2. 5)**Error! Reference source not found.** dan (2. 6) sebagai berikut.

$$1 - \frac{|N_c - N_o|}{N_o} \; atau, \quad (2.5)$$

$$0 \; jika \; \frac{|N_c - N_o|}{N_o} \; lebih \; dari \; 1 \quad (2.6)$$

Tingkatan sistem metrik merupakan sebuah nilai rata-rata dari semua nilai kriteria.

2.4.5 *Self Descriptive*

Self descriptive merupakan sebuah karakteristik perangkat lunak yang memberikan penjelasan tentang implementasi pada suatu fungsi. Bagian *self descriptive* ini dihitung menggunakan metrik yang dibagi menjadi tiga aturan metrik yaitu, *quantity of comments*, *effectiveness of comments measure*, dan *descriptiveness of implementation language measure*.

Quantity of comments

Metrik yang digunakan untuk mengukur *quantity of comments* ini dengan cara jumlah baris komentar dibagi dengan jumlah total baris di setiap modul dan baris kosong tidak dihitung. Kemudian menghitung nilai rata-ratanya untuk metrik tingkat sistem(McCall et al., 1977).

Effectiveness of comments measure

Metrik yang digunakan untuk mengukur *effectiveness of comments measure* ini merupakan penjumlahan skor dari tujuh aturan yang diamati dengan jumlah aturan yang berlaku. Aturan-aturan tersebut adalah sebagai berikut.

a. Modul memiliki komentar prolog berformat standar

   Komentar prolog ini merupakan: nama modul, *author, date, purpose, input, output, function, assumption, limitations and restrictions, accuracy requirements, error recovery procedures, references.* Informasi tersebut sangat penting untuk personel baru yang harus

bekerja dengan perangkat lunak setelah pengembangan, pemeliharaan, pengujian, dan perubahan.

b. Penulisan kode komentar yang seragam

Penggunaan kode komentar yang seragam ini akan memudahkan dalam menentukan komentar. Sehingga perhitungannya didasarkan pada modul yang tidak mengikuti konvensi yang ditetapkan untuk menulis komentar.

c. Isi dari komentar yang sesuai dengan logika pada kode

Sehingga sebuah komentar harus berisikan penjelasan dari logia kode tersebut. Oleh karena itu untuk perhitungannya didasarkan pada jumlah komentar pada modul yang tidak sesuai.

d. Semua kode yang bersifat *machine dependent* harus dijelaskan pada komentar

Komentar tidak hanya menjelaskan fungsi dari sebuah kode, melainkan juga menjelaskan sifat dari kode tersebut. Sehingga perhitungannya adalah didasari pada penjumlahan kode komentar yang tidak menjelaskan kode yang bersifat *machine dependent.*

e. Semua kode yang bersifat *non standard* HOL *(High Order Language)* statement juga harus dijelaskan pada komentar

Seperti penjelasan pada poin sebelumnya, komentar tidak hanya menjelaskan fungsi dari sebuah kode, melainkan juga menjelaskan sifat dari kode tersebut.

f. Semua variabel dari suatu atribut harus dijelaskan pada komentar

Maka dari itu, perhitungannya berdasar pada jumlah modul yang tidak sesuai dengan praktik ini.

g. Komentar tidak hanya mengulang kode yang dituliskan pada program

Komentar adalah untuk menjelaskan mengapa bukan apa, sehingga perhitungannya merupakan penjumlahan dari komentar yang tidak menjelaskan alasannya.

Descriptiveness of implementation language measure

Perhitungan metrik ini merupakan penjumlahan dari enam aturan yang berlaku. Aturan-aturan tersebut adalah sebagai berikut.

a. Bahasa HOL jauh lebih deskriptif dibandingkan *assembly language*

Ukurannya berdasarkan pada jumlah modul yang diimplementasikan sebagian atau seluruhnya.

b. Format standar dalam penulisan modul seperti, komentar prologue, pernyataan deklaratif

Pernyataan yang digunakan dengan seragam dalam modul. Pengukuran ini didasarkan pada banyaknya modul yang tidak sesuai dengan format standar yang telah ditetapkan.

c. Penulisan nama variabel yang subyektif dengan mempertimbangkan deskripsi diri
   Sehingga akan mempermudah dalam mengidentifikasi variabel tersebut. Penamaan seperti NAMA dan SALRY lebih baik dibandingkan penamaan seperti A1 dan A2. Oleh karena itu, pengukuran ini didasarkan pada jumlah modul yang tidak menggunakan nama deskriptif.
d. Teknik seperti pemblokiran, indentasi, pembuatan paragraf untuk konstruksi tertentu telah ditetapkan dengan baik dan harus diikuti secara seragam.
   Pengukuran ini berdasar pada banyaknya modul yang tidak sesuai dengan teknik yang seragam.
e. Satu pernyataan per baris
   Penggunaan pernyataan kelanjutan dengan beberapa pernyataan garis akan menyebabkan kesulitan dalam membaca kode. Sehingga pengukurannya merupakan jumlah kelanjutan ditambah dengan jumlah beberapa baris pernyataan dibagi dengan jumlah total baris untuk setiap modul pada kode, kemudian dirata-rata untuk semua modul dalam sistem.
f. Tidak ada kata perintah yang digunakan sebagai variable
   Hal tersebut akan membingungkan pembaca. Pengukurannya berdasar pada jumlah modul dimana kata perintah digunakan sebagai nama variabel.

2.4.6 *Modularity*

Modularity adalah sebuah atribut perangkat lunak yang menyediakan struktur untuk modul independen. Bagian *modularity* ini menggunakan aturan metrik yang dibagi menjadi dua yaitu, menghitung berdasarkan *stability measure* dan *modular implementation measure.*

Stability Measure

Pengukuran ini didasari oleh kategorisasi pada modul G. Meyer berdasarkan kekuatan modul dan modul kopling. Metrik ini menggabungkan kedua ukuran untuk menghitung jumlah modul yang memerlukan modifikasi apabila ada perubahan pada salah satu modul yang dibuat dibagi dengan jumlah total modul.

Modular implementation measure

Metrik ini merupakan penjumlahan dari lima aturan yang berlaku dibagi dengan jumlah aturan lainnya. Aturan-aturan tersebut adalah sebagai berikut.

a. Struktur Hirarki
   Aturan ini merupakan implementasi modular dari struktur desain top-down. Struktur haruslah mencontohkan aturan bahwa interaksi antar modul dibatasi aliran kontrol antara

modul pendahulu dan modul penerus langsungnya. Perhitungan ini berdasar pada jumlah pelanggar aturan tersebut.

b. Ukuran modul harus sesuai dengan standar yaitu 100 pernyataan procedural

   100 disini dipilih karena sering disebutkan dalam berbagai literatur. Perhitungan ini didasarkan pada jumlah modul yang memiliki ukuran yang melebihi standar.

c. Semua modul mewakili satu fungsi

   Konsep modularitas ini didasarkan pada setiap fungsi yang diimplementasikan dalam modul unik. Perhitungan ini didasari oleh jumlah modul yang mewakili lebih dari satu fungsi.

d. Mengontrol parameter yang ditentukan dengan memanggil modul

   Empat modul berikutnya akan membahas modul pemanggil akan menentukan parameter pengontrol, data input yang diperlukan, dan output yang diperlukan. Oleh karena itu, langkah ini didasarkan pada jumlah pelanggaran aturan ini.

e. Modul tidak berbagi penyimpanan sementara

   Aturan ini merupakan *binary measure,* 1 untuk tidak berbagi penyimpanan sementara dan 0 jika ada. Hal tersebut akan menekankan hilangnya sebuah independensi pada suatu modul jika penyimpanan sementara dibagi antar modul.

2.4.7 *Simplicity*

Simplicity merupakan sebuah karakteristik yang menyediakan implementasi fungsi dengan cara yang paling mudah dipahami dan menghindari kompleksitas. Perhitungan yang digunakan dalam karakteristik ini merupakan penjumlahan dari empat aturan metrik yaitu, *design structure, use of structured language or structured language preprocessor, data and control flow complexity,* dan *measure of simplicity of coding techniques.*

Design structure

Pengukuran ini didasari oleh penjumlahan dari setiap aturan dibagi dengan aturan lainnya. Aturan-aturan tersebut adalah sebagai berikut.

a. Desain diatur dalam mode top down

   Bagian hirarki modul sistem biasanya mudah dibuat dari dokumentasi desain.

b. Tidak ada fungsi yang diduplikat

   Deskripsi fungsi pada yang akan dilakukan oleh masing - masing modul pada desain dan fungsi aktual yang dilakukan oleh modul yang dikodekan harus dievaluasi untuk memastikan tidak diduplikasi oleh modul lain.

c. Module independence

   Pemrosesan yang dilakukan dalam modul tidak bergantung pada sumber input atau tujuan output. Ukuran elemen ini didasarkan pada jumlah modul yang tidak mengikuti aturan ini.

d. Pemrosesan modul tidak bergantung pada pemrosesan sebelumnya

   Misalnya, pertama kali melalui modul. Aturan ini diterapkan pada desain dan implementasi.

e. Setiap deskripsi modul mencakup input, output, pemrosesan, batasan

   Pengukuran untuk elemen ini merupakan jumlah modul yang tidak mendokumentasikan informasi ini.

f. Setiap modul memiliki pintu masuk tunggal dan pintu keluar tunggal

   Penentuan jumlah modul yang melanggar aturan ini pada saat desain dan implementasi dapat dibuat dan menjadi dasar metrik

g. Tidak ada data global

   Hal ini merupakan perhitungan biner yang mana jika tidak ada data global maka memiliki ukuran 1 dan jika terdapat data global maka ukurannya bernilai 0.

Use of structured language or structured language preprocessor

Aturan ini merupakan konstruksi yang mirip dengan pernyataan IFTHENELSE, DOWHILE, DOUNTIL, dan CASE yang terkait dengan pemrograman terstruktur.

Data and control flow complexity

Metrik ini dapat diukur dari representasi desain (misalnya diagram alir) dan kode secara otomatis. Analisis aliran jalur dan informasi penggunaan variabel di sepanjang jalur dalam program. Variabel dianggap “hidup” di sebuah node jika dapat digunakan lagi di sepanjang jalur dalam program. Sehingga pengukuran pada elemen ini adalah penjumlahan “kehidupan” semua variabel di sepanjang jalur dalam program. Hal tersebut dinormalisasi dengan membaginya dengan kompleksitas maksimum program (semua variabel hidup di seluruh jalur).

Measure of simplicity of coding techniques

Perhitungan pada metrik ini merupakan kuantitas rata-rata dari semua ukuran modul untuk sistem. Macam-macam aturan pada metrik ini adalah sebagai berikut.

a. Aliran modul dari atas ke bawah

   Hal ini merupakan perhitungan biner yang mana jika tidak ada data global maka memiliki ukuran 1 dan jika terdapat data global maka ukurannya bernilai 0.

b. Kode yang memiliki operator Boolean yang majemuk yaitu melibatkan dua atau lebih boolean

   Pengukuran ini didasarkan pada jumlah operator yang rumit tersebut per pernyataan yang dapat dieksekusi pada sebuah modul.

c. Keluar masuk pada loop

   Dalam sebuah modul harus hanya terdapat satu pintu masuk dan satu pintu keluar. Pengukuran ini didasari pada jumlah loop yang memenuhi aturan ini dibagi dengan jumlah total loop.

d. Modifikasi loop

   Sebuah modifikasi indeks loop tidak hanya akan memperumit logika modul, tetapi menyebabkan masalah pada saat proses debugging. Pengukuran ini didasarkan pada jumlah indeks loop yang dimodifikasi dibagi dengan jumlah total loop.

e. Modul tidak memodifikasi sendiri

   Jika sebuah modul memiliki kemampuan untuk memodifikasi logika pemrosesannya, maka akan menjadi sangat sulit untuk mengenali statusnya saat terjadi kesalahan. Pengukuran ini menekankan kompleksitas tambahan dari modul yang memodifikasi sendiri.

f. Semua argumen yang diteruskan ke modul bersifat parametrik

   Hal ini adalah perhitungan biner yang mana bernilai 1 jika semua parameter parametrik, 0 jika semuanya tidak. Pengukuran ini didasarkan pada potensi masalah yang dapat muncul jika konstanta atau data global digunakan sebagai argumen.

g. Jumlah label pernyataan

   Pengukuran ini didasari pada premis bahwa semakin banyak label pernyataan ditambahkan ke modul, semakin kompleks untuk dipahami.

h. Nama untuk untuk penggunaan variable

   Hal ini merupakan pengukuran biner, diberi nilai 1 jika menggunakan nama yang unik dan 0 jika tidak.

i. Penggunaan variabel tunggal

   Variabel harus digunakan hanya untuk satu tujuan dengan satu cara. Hal ini merupakan perhitungan biner. Nilai 1 jika variabel digunakan hanya dalam satu cara 0 jika digunakan sebagai tujuan.

j. Tidak ada ekspresi mode campuran

Jika terdapat ekspresi mode campuran maka kode akan menjadi lebih rumit. Pengukuran ini adalah 1 jika tidak terdapat ekspresi mode campuran dan 0 jika terdapat ekspresi mode campuran.

k. *Nesting level*

Semakin banyaknya tingkatan loop maka akan semakin besar kompleksitasnya. Pengukurannya pada elemen ini adalah kebalikan dari tingkat sarang maksimum.

l. Jumlah cabang

Semakin banyak jalur dan jumlah cabang yang ada pada sebuah modul, semakin besar pula kompleksitasnya. Perhitungan yang digunakan pada elemen ini didasarkan pada jumlah pernyataan keputusan per pernyataan yang dapat dieksekusi.

m. Jumlah GOTO

Banyak sekali literatur yang menjelaskan bahwa manfaat untuk menghindari GOTO. Perhitungan didasarkan pada jumlah pernyataan GOTO per pernyataan yang dieksekusi.

n. Tidak ada bahasa asing yang digunakan pada kode

Elemen ini merupakan perhitungan biner yang mana bernilai 1 jika tidak ada kode asing dan 0 jika ada. Kode asing merupakan kode yang tidak berfungsi atau tidak dapat dieksekusi.

o. Campuran variabel dalam modul

Dari sudut kesederhanaan variabel lokal lebih baik daripada variabel global. Ukuran ini adalah rasio variabel internal (lokal) terhadap variabel total (internal (total)) plus eksternal (global) pada modul.

p. Kepadatan variable

Semakin banyak variabel yang digunakan pada sebuah modul, maka semakin besar kompleksitas pada suatu modul. Perhitungan pada elemen ini didasarkan pada jumlah penggunaan variabel dalam sebuah modul dibagi dengan kemungkinan penggunaan maksimum.

2.4.8 *Consistency*

Consistency adalah karakteristik perangkat lunak yang menyediakan desain dan teknik implementasi dengan notasi yang seragam. Bagian *consistency* dibagi menjadi dua aturan metrik yaitu, menghitung berdasar pada *procedure consistency measure* dan *data consistency measure*.

Procedure consistency measure

Aturan ini menjelaskan mengenai ukuran konsistensi prosedur dalam desain, kesepakatan dalam modul. Macam-macam aturan pada metrik ini adalah sebagai berikut.

a. *Standard Design Representation*

   Flow charts, HIPO charts, program design language merupakan bentuk representasi desain. Standar mewakili elemen harus ditetapkan dan diikuti, namun hanya berlaku untuk bagian desain yang mana ukuran ini berdasar pada jumlah modul representasi desainnya tidak memenuhi standar.

b. *Calling sequence conventions*

   Standar harus ditetapkan selama mendesain dan selama mengimplementasikan. Ukuran ini berdasar pada jumlah modul yang tidak mengikuti kesepakatan.

c. *Input/Output Conventions*

   Kesepakatan untuk modul mana yang akan melakukan I/O, bagaimana untuk dapat tercapai, serta format untuk I/O harus ditetapkan dan diikuti. Ukuran ini untuk modul yang tidak mengikuti kesepakatan.

d. *Error Handling Conventions*

   Metode yang konsisten dalam penanganan *error* sangat diperlukan, setiap kesepakatan yang ditetapkan diikuti oleh implementasi dalam sebuah desain.

Implementation for generality measure

Aturan ini berisikan penjumlahan dari lima aturan sebagai yang dibagi dengan jumlah aturan yang dapat dihitung, aturan-aturan tersebut dapat dilihat sebagai berikut.

a. *Standard Data Usage Representation*

   Penggunaan data harus ditetapkan dalam desain dan diikuti dalam implementasi. Pada elemen ini hanya berlaku pada fase desain yang diukur melalui module yang tidak mengikuti standar yang telah ditetapkan.

b. *Naming Convention*

   Kesepakatan dalam menamakan sebuah variabel dan modul harus ditetapkan dan diikuti dalam fase implementasi.

c. *Unit Consistency*

   Satuan dalam variabel harus konsisten dalam penggunaanya dengan semua penggunaan variabel. Perhitungan elemen ini didasarkan pada jumlah modul dimana unit yang konsisten tidak digunakan.

d. *Consistent Global Definitions*

Elemen global harus didefinisikan dengan cara yang sama oleh semua modul. Perhitungan pada elemen ini didasarkan pada jumlah modul di mana elemen data global didefinisikan dengan cara yang tidak konsisten baik untuk desain maupun implementasi.

e. *Data Type Consistency*

Elemen data yang didefinisikan sebagai tipe data tertentu, maka variabel tersebut akan tetap pada elemen data tersebut pada semua kejadian. Pengukuran didasarkan pada banyaknya modul yang menggunakan tipe data secara tidak konsisten.

2.4.9 *Generality*

Generality adalah karakteristik perangkat lunak yang memberikan keluasan pada fungsi yang dilakukan. Pencarian nilai metrik karakteristik ini dilihat pada dua aturan metrik yaitu, *extent to which modules are referenced by other modules* dan *implementation for generality measure*.

Extent to which modules are referenced by other modules

Sebuah modul dianggap lebih bersifat umum jika digunakan oleh lebih dari satu modul, sehingga perhitungan ini adalah jumlah modul umum dibagi dengan jumlah total modul yang memberikan ukuran.

Implementation for generality measure

Aturan metrik ini akan dibagi menjadi lima aturan yang mana untuk mendapatkan hasil nilai aturan ini adalah penjumlahan skor aturan yang dibagi dengan jumlah aturan lainnya.

a. *Input, processing, output function are not mixed in a single function*

Sebuah modul yang melakukan I/O serta pemrosesan tidak sesering modul yang hanya menyelesaikan pemrosesan.

b. *Application and machine dependent functions are not mixed in a single module*

Referensi apapun ke sebuah fungsi *machine dependent* yang bergantung akan mengurangi *generality* atau keumuman pada modul tersebut. Contohnya seperti referensi jam sistem untuk tujuan pengaturan waktu.

c. *Processing not data volume limited*

Sebuah modul yang dirancang dan dikodekan hanya menerima tidak lebih dari 100 input item data untuk diproses, sehingga modul tersebut tidak bersifat umum yang mana akan menerima volume input apapun. Pengukuran elemen ini didasarkan pada jumlah modul yang dirancang atau diimplementasikan untuk membatasi volume data.

d. *Processing not data value limited*

Kemampuan ini diperlukan untuk mencegah penyediaan data ke fungsi yang tidak ditentukan atau tingkat presisinya tidak dapat diterima. Dari sudut pandang yang umum semakin kecil subset dari semua kemungkinan input yang dapat diterapkan suatu fungsi maka semakin kurang umum fungsi tersebut.

e. *All constants should be defined once*

Aturan ini pada dasarnya mendefinisikan konstanta sebagai nilai parametrik. Pada sebuah basis data, konstanta dapat diubah untuk mengakomodasi aplikasi yang berbeda dari fungsi modul itu. Contohnya untuk menghitung hubungan matematis pada tingkat presisi yang lebih tinggi.

2.4.10 *Expandability*

Expandability adalah perangkat lunak yang menyediakan perluasan persyaratan penyimpanan data pada sebuah fungsi komputasi. Bagian *expandability* ini dibagi menjadi dua aturan metrik yaitu, menghitung yang berdasarkan pada *data storage expansion measure* dan *extensibility Measure.*

Data storage Expansion Measure

Aturan metrik ini jumlah skor aturan yang berlaku tersebut akan dibagi dengan jumlah aturan lainnya, yang mana aturan metrik tersebut dibagi menjadi dua buah aturan yaitu, *logical processing independent of storage specification,* dan *percent of memory capacity uncommitted*.

a. *Logical processing independent of storage specification*

Logical processing dari suatu modul harus bersifat independen dari ukuran penyimpanan, ruang buffer, dan ukuran array. Penggunaan variabel di desain untuk menyesuaikan ukuran array secara parametrik.

b. *Percent of memory capacity uncommitted*

Jumlah memori yang tersedia untuk melakukan ekspansi merupakan hal yang sangat penting. Sehingga perhitungan ini mengidentifikasi persentase memori yang tersedia yang belum digunakan dalam mengimplementasikan sistem saat ini.

Extensibility measure

Aturan metrik ini jumlah skor aturan yang berlaku tersebut akan dibagi dengan jumlah aturan lainnya, yang mana aturan metrik tersebut dibagi menjadi tiga aturan yaitu, *accuracy, convergence, timing attribute which control processing are parametric, module table driven* dan *percent of speed capacity uncommitted.*

a. *Accuracy, convergence, timing attribute which control processing are parametric*

Sebuah modul yang dapat memberikan tingkat konvergensi atau pengaturan waktu yang bervariasi untuk mencapai presisi yang lebih tinggi.

b. *Modules table driven*

Penggunaan tabel pada modul untuk memfasilitasi representasi dan karakteristik pemrosesan yang berbeda.

c. *Percent of speed capacity uncommitted*

Fungsi tertentu mungkin diperlukan untuk spesifikasi persyaratan yang harus diselesaikan pada jangka waktu tertentu dalam waktu keseluruhan.

2.4.11 *Instrumentation*

Instrumentation adalah karakteristik perangkat lunak yang menyediakan pengukuran penggunaan atau identifikasi kesalahan. Bagian ini menggunakan metrik yang dibagi menjadi tiga aturan. Aturan-aturan tersebut adalah *module testing measure, integration testing measure,* dan *system testing measure*.

Module testing measure

Pengukuran ini yaitu rata-rata dari semua ukuran modul, aturan metrik ini dibagi lagi menjadi dua yaitu, *path coverage* dan *input parameter boundary tested*.

a. *Path Coverage*

Perhitungan aturan ini dengan melihat rencana untuk menguji beberapa jalur dalam modul yang dibuat selama desain dan *test case* yang dikembangkan selama implementasi. Perhitungan aturan ini mengukur jumlah jalur yang direncanakan untuk diuji dibagi dengan jumlah total jalur.

b. *Input Parameter Boundary Tested*

Aspek lain dari pengujian modul melibatkan pengujian rentang input ke modul. Hal ini dilakukan dengan menggunakan modul pada berbagai nilai batas dari parameter input.

Integration Testing Measure

Aturan metrik ini dibagi menjadi dua aturan yaitu, *module interface tested,* dan *performance requirements*. Oleh karena itu, pengukuran yang didapatkan pada elemen ini merupakan rata-rata dari dua elemen tersebut.

a. *Module Interface Tested*

Salah satu aspek pengujian integrasi adalah pengujian semua modul ke antarmuka modul. Perencanaan untuk menyelesaikan pengujian ini telah disiapkan selama proses desain dan test dilakukan selama proses pengembangan.

b. *Performance Requirements*

Aspek selanjutnya dalam pengujian interaksi adalah pemeriksaan yang melibatkan penyesuaian modul dan subsistem dengan kinerja persyaratan.

System testing measure

Aturan metrik ini dibagi menjadi dua aturan yaitu, *module coverage,* dan *identification of test inputs and output in summary form.* Pengukuran yang didapatkan pada aturan metrik ini merupakan rata-rata dari dua aturan tersebut.

a. *Module Coverage*

Untuk semua skenario sistem pengujian yang telah direncanakan, diperlukan persentase dimana kode tersebut telah diuji. Dapat diketahui kualitas dari kode yang telah dibuat.

b. *Identification of test inputs and outputs in summary form*

Pengukuran ini mengidentifikasi apakah sebuah kode dapat menampilkan *input* dan *output* dalam bentuk ringkasan. Hasil tes dan cara hasil ini ditampilkan merupakan hal yang penting untuk efektivitas pengujian terutama berlaku selama pengujian sistem, karena volume *input* dan *output* data yang berpotensi besar. Ukuran tersebut dapat diterapkan pada rencana dan spesifikasi dalam fase desain dan pengembangan kapabilitas.

BAB III
ANALISIS DAN PERANCANGAN PENGUJIAN

Pembahasan analisis dan perancangan pengujian akan membahas mengenai tiga hal yaitu analisis kebutuhan pengujian untuk menganalisis apa saja yang dapat diuji serta fitur fungsional dan nonfungsional. Kemudian, perencanaan untuk melakukan pengujian seperti menyiapkan rencana *tools* serta estimasi waktu dan sumber daya yang akan digunakan. Selanjutnya, pengembangan kasus uji untuk menjadi acuan dalam pengujian dengan membuat *tase case metric* untuk setiap aturan karakteristik.

3.1 Analisis Kebutuhan Pengujian (*Requerement Analisis*)

Menganalisa apa saja yang dapat diuji secara fungsional dan nonfungsional (Hamilton, 2020). Kegiatan yang dilakukan pada fase ini berupa mengidentifikasi jenis tes yang akan dilakukan, mengumpulkan detail mengenai prioritas *testing* hingga identifikasi lingkungan (Auliyaa, 2020). Pada penelitian ini akan melakukan *requirement analysis* yang merupakan tahapan pertama *software testing* pada aplikasi Ivent. Ivent merupakan aplikasi *e-commerce* berbasis web yang menghubungkan pemilik usaha *event organizer* dan vendor dengan konsumen (Aikal, 2021). Para pengguna aplikasi Ivent dapat bebas memilih dan menentukan *event organizer* atau vendor sesuai kebutuhan dengan mudah, karena berbagai pemilik usaha *event organizer* dan vendor dikumpulkan pada satu *platform*. Berdasarkan tahapan analisis kebutuhan, penelitian ini akan menguji dengan melihat penulisan kode yang ditulis oleh *developer*, contohnya pembuatan akun pengguna, apakah email yang didaftarkan dapat dipastikan terlebih dahulu melalui verifikasi. Selain itu pengujian juga melihat dari dokumentasi desain pembuatan aplikasi web Ivent. Contohnya dengan melihat dari *activity diagram* dan *use case diagram*. Dari hasil analisis aplikasi ivent memiliki delapan modul yang akan diuji. Modul-modul tersebut, yaitu *admin* memiliki total 180 baris kode modul ini berfungsi untuk melakukan transaksi, *auth* memiliki total 207 baris kode modul ini berfungsi untuk memproteksi halaman atau fitur dari web yang hanya dapat diakses oleh orang tertentu, *controller* memiliki total 1110 baris kode modul ini berfungsi untuk mengolah data, *middleware* memiliki total 127 baris kode modul ini berfungsi untuk memverifikasi setiap *request* yang masuk ke dalam aplikasi, *http* memiliki total 56 baris kode modul ini berfungsi melakukan *routing* pada setiap modul untuk digunakan dalam modul lain, *app* memiliki total 184 baris kode modul ini berfungsi melakukan *order* dan melihat rincian produk, *migrations*

memiliki total 401 baris kode modul ini berfungsi untuk membuat *database*, *seed* memiliki total 70 baris kode modul ini berfungsi untuk menghasilkan data secara otomatis ketika versi awal dibuat. Dengan demikian *software* ini masih tergolong *software* yang kecil.

Kebutuhan nonfungsional melekat pada kebutuhan fungsional karena keduanya saling terkait dan saling memengaruhi dalam menyusun suatu sistem atau perangkat lunak. Pengujian dengan perhitungan nonfungsional yang dijelaskan pada metode *McCall's Factor* dapat dilakukan pada aplikasi Ivent. Mengukur kualitas dari perangkat lunak dengan cara menilai dan membandingkan faktor kualitas pada perangkat lunak. *Maintainability* berdasarkan metode *McCall's Factor* akan menguji beberapa karakteristik, salah satu contoh perhitungannya adalah mengukur apakah sistem dibangun dengan jumlah baris kode yang relative singkat atau bertele-tele. *Flexibility* berdasarkan metode *McCall's Factor* juga akan menguji beberapa karakteristik, salah satu contoh perhitungannya adalah mengukur apakah sistem memiliki keluasan pada fungsi yang dilakukan. Salah satu contoh perhitungan yang akan dilakukan pada *Testability* berdasarkan metode *McCall's Factor* adalah pengukuran jalur pengujian dengan melihat dokumentasi desain dan pengimplementasiannya.

3.2 Perencanaan Pengujian (*Test Planning*)

Tahapan ini tim QA menyiapkan rencana seperti *tools* yang akan digunakan serta estimasi waktu dan sumber daya (Rizky, 2019). Aktivitas yang dilakukan pada tahapan ini adalah menentukan *test plan* sebelum pengujian dilakukan, perencanaan *resource* dan menentukan *roles*. Diperlukan perencanaan dalam membuat *test case* sebelum pengujian dilakukan, agar pengujian lebih terstruktur prosesnya. *Tools* yang digunakan untuk melakukan pengujian, yaitu *Visual Studio Code*, *XAMPP*, *Google Spreadsheets*, *Github*, *zoom*, dan *desmos scientific calculator*. Estimasi waktu dalam melakukan pengujian aplikasi ivent, yaitu dua bulan. Sumber daya yang digunakan seperti laptop pribadi dan kuota internet.

Akan ditentukan beberapa faktor nonfungsional pada aplikasi perangkat lunak, faktor tersebut akan menentukan karakteristik-karakteristik yang akan diuji seperti pada Tabel 3. 1 Pembagian Karakteristik Setiap Faktor.

Tabel 3. 1 Pembagian Karakteristik Setiap Faktor

Faktor / Karakteristik	*Maintainability*	*Flexibility*	*Testability*
Conciseness	✓		
Self Descriptive	✓	✓	✓
Modularity	✓		✓
Simplicity	✓		✓
Consistency	✓		
Generality		✓	
Expandability		✓	
Instrumentation			✓

Perhitungan nilai akan menggunakan metrik untuk mengukur faktor pada aplikasi perangkat lunak. Metrik yang digunakan tersebut pada tingkatan sistem akan berguna untuk mengukur keseluruhan tentang bagaimana kemajuan sistem tersebut dari berbagai faktor kualitas. Jika nilai pada metrik rendah, itu memungkinkan adanya kegagalan, sehingga diperlukan tindakan yang korektif. Namun, nilai rendah pada metrik tertentu mungkin juga tidak memiliki kegagalan secara umum, tetapi metrik dapat mengetahui faktor mana yang bermasalah sehingga tindakan korektif dapat diaplikasikan untuk mengembangkan faktor tersebut agar lebih baik kedepannya.

Perhitungan metrik tersebut dinilai berdasarkan *requirements, design,* dan *implementation*. Pada *requirements* perangkat lunak akan dinilai dari proses identifikasi jenis tes yang akan dilakukan. Kemudian, *design* perangkat lunak akan dinilai berdasarkan dokumentasi dari *developer*. Sedangkan, tahap *implementation* akan dinilai berdasarkan implementasi kode yang ditulis oleh *developer*. Namun, pada pengujian ini perhitungan hanya dinilai dari fase desain dan fase implementasi. Karena perhitungan metrik pada metode *McCall's Factor* tidak menghitung nilai dari fase *requirements*. Setelah mendapatkan nilai-nilai karakteristik, maka nilai tersebut akan dihitung menggunakan rumus *Software Quality McCall's Factor*. Untuk menghitung menggunakan rumus *Software Quality McCall's Factor* dibutuhkan bobot yang bergantung pada kepentingannya. Setelah berdiskusi dengan *developer,* maka bobot ini dianggap sama rata untuk semua karakteristik yaitu 0.125. Hal ini untuk penyederhanaan kasus pengujian, semua karakteristik dalam penelitian ini dianggap penting.

3.3 Pengembangan Kasus Uji (*Test Case Developer*)

Tahapan ini menjadi acuan untuk pengujian dalam membuat *test case*. Agar sistem yang diuji dapat memenuhi ketentuan dan standar tertentu. Dari Tabel 3. 2 *Test Case Metric* dapat

dilihat bahwa setiap karakteristik memiliki aturan yang harus dihitung menggunakan rumus metrik masing-masing. Setelah didapatkan nilai dari aturan, selanjutnya akan dirata-rata menggunakan rumus *System Matric Value*. Nilai dari *System Matric Value* nantinya akan digunakan untuk perhitungan *Factor Software Quality* seperti pada Persamaan (2. 1). Dalam melakukan pengujian menggunakan metode *McCall's Factor* hanya akan dihitung berdasarkan dua fase atau bagian, yaitu fase desain dan fase implementasi. Pembahasan fase desain tidak melihat dari *mock up* dan hanya akan mengacu pada dokumentasi pembuatan aplikasi Ivent yang telah *developer* buat. Fase implementasi akan mengacu pada kode yang telah diketik oleh *developer*.

Tabel 3. 2 *Test Case Metric*

Karakteristik	Metric	Bobot	Design		Implementasi	
			1/0	Value	1/0	Value
Conciseness	1. Halstead Measure $1 - \frac{\lvert calculated - observed \rvert}{Module}$	0.125				
	System Matrix Value $\frac{Sum\ Halstead\ measure\ each\ module}{Module}$					
Self Descriptive	1. Quantitiy Of Comments $\frac{\#\ Of\ Comments}{Total\ \#\ lines}$	0.125				
	System Matrix Value $\frac{Sum\ Quantity\ of\ Comments}{Total\ \#\ modules}$					
	2. Effectiveness of Comments Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$					
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					

	3. Descriptiveness Of Implementation Language $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$					
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
Modularity	1. Stability Measure $\frac{Expected\ \#\ modules}{Total\ \#\ modules}$	0.125				
	System Matrix Value $\frac{Expected\ \#\ modules}{Total\ \#\ modules}$					
	2. Modular Implementation Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$					
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
Simplicity	1. Design Structure Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$	0.125				
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
	2. Use Of Structured Language Or Preprocessor 1 jika menggunakan dan 0 jika tidak					
	System Matrix Value $\frac{Sum\ Of\ Modules\ Scores}{\#\ modules\ scores}$					
	3. Complexity Measure					
	System Matrix Value $\frac{Total\ Scores\ complexity\ measure}{Total\ \#\ modules}$					
	4. Measure Of Coding Simplicity					

	$1 - \frac{\#\ of\ elements}{executable\ statements}$					
	Module Matrix value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
	System Matrix Value $\frac{Scores\ Coding\ Simplicity\ measure}{\#\ modules}$					
Consistency	1. Procedure Consistency Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$	0.125				
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
	2. Data Consistency Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$					
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
Generality	1. Extent To Which Module Is Referenced By Other Modules $\frac{\#\ Common\ Modules}{Total\ \#\ modules}$	0.125				
	System Matrix Value $\frac{\#Common\ Modules}{Total\ \#\ modules}$					
	2. Implementation For Generality $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$					
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
Expandability	1. Data Storage Expansion Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$	0.125				
	System Matrix Value					

	$\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
	2. Extensibility Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$					
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
Instrumentation	1. Module Testing Measure $\frac{Path\ to\ be\ tested}{Total\ \#\ Path}$	0.125				
	Modul Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
	System Matrix Value $\frac{Total\ Scores\ modules\ testing\ measure}{Total\ \#\ modules}$					
	2. Integration Testing Measure $\frac{\#\ to\ be\ tested}{Total\ \#\ interface}$					
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$					
	3. System Testing Measure $\frac{\#modules\ to\ be\ executed}{Total\ \#\ of\ modules}$					
	System Matrix Value $\frac{Total\ Scores\ modules\ testing\ measure}{Total\ \#\ modules}$					

di mana:

Karakteristik = Pembagian sub faktor *McCalls* yang akan diuji.

Metrik = Rumus setiap aturan yang akan diuji

Bobot = Nilai atau mutu yang tergantung pada kepentingan karakteristik

Desain = Fase yang dilihat untuk melakukan pengujian berdasarkan dokumentasi

Implementasi = Fase yang dilihat untuk melakukan pengujian berdasarkan *source code*

1/0 = Pengujian yang membutuhkan *binary measure*, 1 untuk ya dan 0 untuk tidak.

Value = Nilai yang didapatkan dari perhitungan metrik

Skenario yang dilakukan untuk menghitung metrik yang ada Tabel 3. 2 *Test Case Metric* pada adalah sebagai berikut.

Conciseness

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Kemudian mencari operan dan operator serta total penggunaan operan dan operator dari delapan modul untuk mendapatkan nilai dari metode *Halstead Measure* menggunakan rumus pada Tabel 3. 2 *Test Case Metric*. Selanjutnya nilai setiap modul akan dirata-rata menggunakan rumus *system matrix value*.

Self Descriptive

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Karakteristik *Self Descriptive* memiliki tiga aturan yang harus dihitung, aturan yang pertama mencari banyaknya jumlah baris komentar dan total baris dari delapan modul. Dari jumlah baris komentar dan total baris yang ditemukan akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric*, kemudian nilai tersebut akan dirata rata menggunakan rumus *system matrix value*. Aturan kedua mencari penjumlahan skor dari tujuh aturan yang berlaku, setiap aturan akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata rata menggunakan rumus *system matrix value.* Aturan ketiga terdiri dari enam aturan yang akan dijumlah, setiap aturan akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value.*

Modularity

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Karakteristik *Modularity* memiliki dua aturan yang harus dihitung, aturan pertama mencari modul yang dikategorisasi oleh modul G. Meyer berdasarkan kekuatan modul dan modul kopling. Aturan kedua mencari penjumlahan skor dari lima aturan yang berlaku, setiap aturan akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*.

Simplicity

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Karakteristik *Simplicity* memiliki empat aturan yang harus dihitung, aturan pertama mencari penjumlahan skor dari tujuh aturan yang berlaku, setiap aturan akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan kedua mencari penyataan seperti IFTHENELSE, DOWHILE, DOUNTIL, dan CASE. Kemudian menghitung aturan tersebut sesuai dengan metrik pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan ketiga mencari variabel yang dianggap sebagai "hidup" atau sering digunakan pada sepanjang jalur. Kemudian menghitung aturan tersebut sesuai dengan metrik menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan keempat mencari penjumlaham skor dari 16 aturan yang berlaku, setiap aturan akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*.

Consistency

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Karakteristik *Consistency* memiliki dua aturan yang harus dihitung, aturan pertama mencari penjumlahan skor dari empat aturan yang berlaku, setiap aturan akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan kedua mencari penjumlahan skor dari lima aturan yang harus dihitung, setiap aturan akan dihitung menggunakan rumus pada pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*.

Generality

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Karakteristik *Generality* memiliki dua aturan yang harus dihitung, aturan pertama mencari modul yang digunakan pada modul lainnya. Banyaknya modul yang digunakan pada modul lainnya akan dihitung menggunakan rumus pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan kedua mencari penjumlahan skor dari lima aturan yang harus dihitung, setiap aturan akan dihitung menggunakan rumus pada pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*.

Expandability

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Karakteristik *Expandability* memiliki dua aturan yang harus dihitung, aturan pertama mencari penjumlahan skor dari dua aturan yang harus dihitung, setiap aturan akan dihitung menggunakan rumus pada pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan kedua memiliki dua aturan yang akan dijumlah, setiap aturan akan dihitung menggunakan rumus pada pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*.

Instrumentation

Langkah awal yang dilakukan, yaitu membuka *source code* dengan aplikasi *Visual Studio Code*. Karakteristik *Instrumentation* memiliki tiga aturan yang harus dihitung, aturan pertama memiliki dua aturan yang akan dijumlah, setiap aturan akan dihitung menggunakan rumus pada pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan kedua mencari penjumlahan skor dari dua aturan yang akan dihitung, setiap aturan akan dihitung menggunakan rumus pada pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*. Aturan ketiga mencari penjumlahan dari dua aturan, setiap aturan akan dihitung menggunakan rumus pada pada Tabel 3. 2 *Test Case Metric* dan dirata-rata menggunakan rumus *system matrix value*.

3.3.1 Klasifikasi Metrik

Klasifikasi metrik digunakan untuk melihat hasil perhitungan pada metrik, semakin besar nilai metrik menandakan bahwa web dapat bekerja dengan baik. Tabel 3. 3 Rentang Kategori Kualitas menjelaskan rentang kategori untuk klasifikasi kualitas metrik agar memenuhi ketentuan dan standar tertentu. Pembagian kualitas dibagi menjadi 5, skala ini memperhatikan nilai dari persentase. Nilai maksimal yang diharapkan akan bernilai 100% dan nilai minimum sebesar 0%.

Tabel 3. 3 Rentang Kategori Kualitas

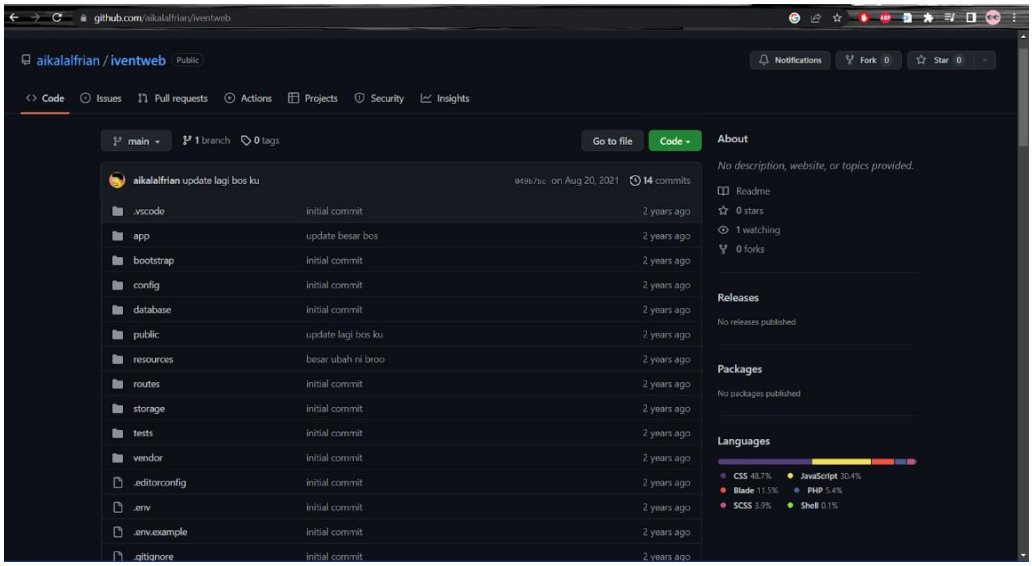
Kategori	**Persentase**
Sangat Baik	81% - 100%
Baik	61% - 80%
Cukup Baik	41% - 60%
Tidak Baik	21% - 40%
Sangat Tidak Baik	<20%

BAB IV
HASIL DAN PEMBAHASAN

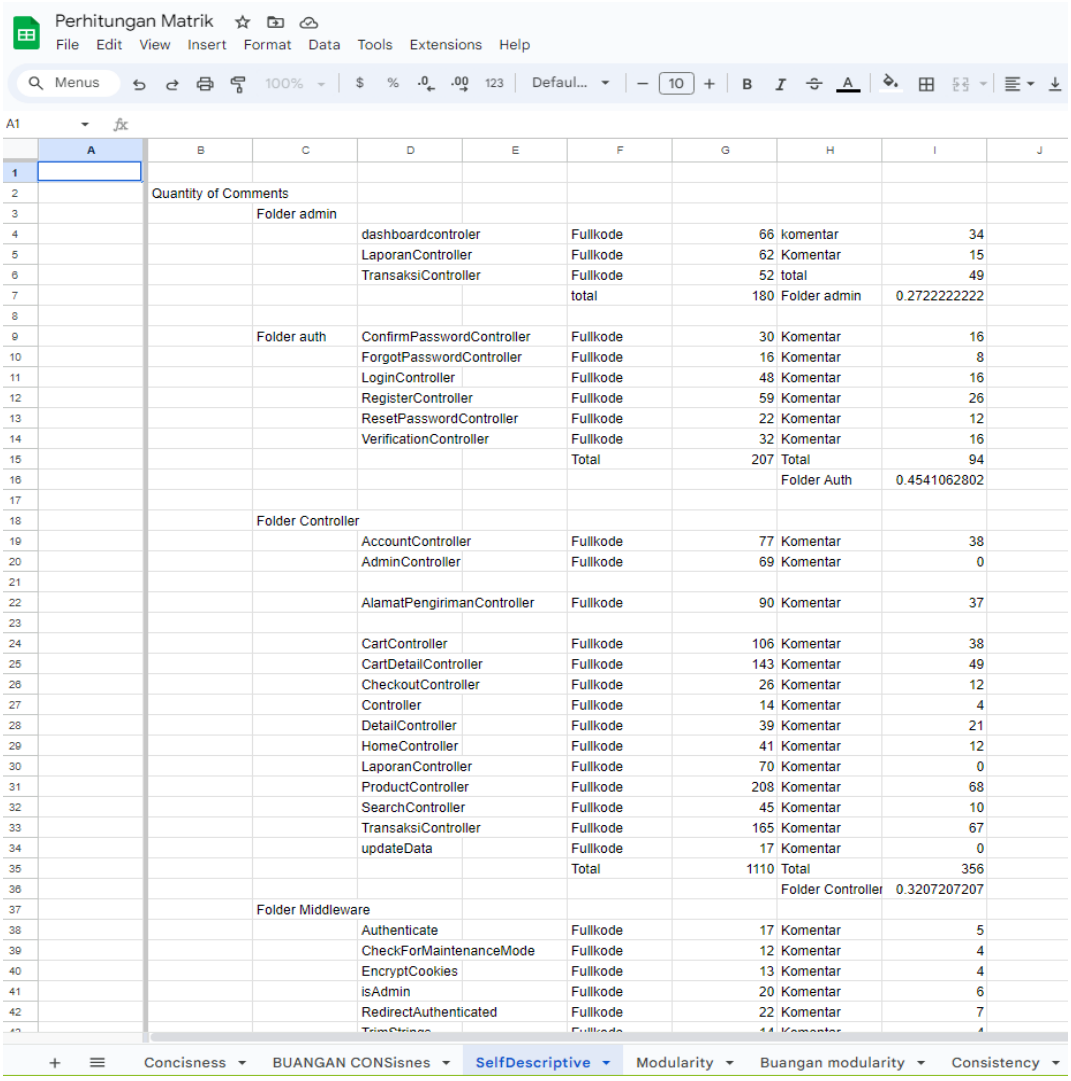
Setelah menentukan analisis kebutuhan, rencana untuk melakukan pengujian, serta pengembangan kasus uji yang menjadi acuan untuk pengujian dalam membuat *test case metric* pada pembasahan sebelumnya. Dalam pembahasan kali ini akan membahas tiga hal yaitu pengaturan lingkungan uji dalam menentukan syarat *software* yang akan diuji. Selanjutnya, eksekusi pengujian berdasarkan *test case metric* yang telah dibuat sebelumnya, dan untuk penutupan siklus uji yang berisikan nilai metrik dan nilai akhir untuk setiap karakteristik berdasarkan faktor-faktornya.

4.1 Pengaturan Lingkungan Uji (*Environment Setup*)

Program atau *source code* aplikasi Ivent didapatkan melalui *github* seperti Gambar 4.1 *Source Code* Aplikasi Ivent. Ketika ingin mengakses web Ivent, hal pertama yaitu perlu untuk mengunduh kode pada *github* tersebut. Setelah itu, diperlukan instalasi *Visual Studio Code* sebagai *code editor* dan berguna untuk menjalankan aplikasi. Selanjutnya dilakukan instalasi *tools* XAMPP yang digunakan untuk menyimpan basis data. Pengujian ini menggunakan *spreadsheets* untuk melakukan pencatatan hasil pengujian seperti Gambar 4.2 *Spreadsheets* Pencatatan Metrik. Perhitungan metrik dilakukan dengan menggunakan *tools desmos scientific calculator*. Dalam pengujian juga perlu untuk menyiapkan dokumentasi desain pembuatan aplikasi web Ivent.



Gambar 4.1 *Source Code* Aplikasi Ivent



Perhitungan Matrik

File Edit View Insert Format Data Tools Extensions Help

Quantity of Comments								
	Folder admin							
		dashboardcontroler	Fullkode	66	komentar	34		
		LaporanController	Fullkode	62	Komentar	15		
		TransaksiController	Fullkode	52	total	49		
			total	180	Folder admin	0.2722222222		
	Folder auth	ConfirmPasswordController	Fullkode	30	Komentar	16		
		ForgotPasswordController	Fullkode	16	Komentar	8		
		LoginController	Fullkode	48	Komentar	16		
		RegisterController	Fullkode	59	Komentar	26		
		ResetPasswordController	Fullkode	22	Komentar	12		
		VerificationController	Fullkode	32	Komentar	16		
			Total	207	Total	94		
					Folder Auth	0.4541062802		
	Folder Controller							
		AccountController	Fullkode	77	Komentar	38		
		AdminController	Fullkode	69	Komentar	0		
		AlamatPengirimanController	Fullkode	90	Komentar	37		
		CartController	Fullkode	106	Komentar	38		
		CartDetailController	Fullkode	143	Komentar	49		
		CheckoutController	Fullkode	26	Komentar	12		
		Controller	Fullkode	14	Komentar	4		
		DetailController	Fullkode	39	Komentar	21		
		HomeController	Fullkode	41	Komentar	12		
		LaporanController	Fullkode	70	Komentar	0		
		ProductController	Fullkode	208	Komentar	68		
		SearchController	Fullkode	45	Komentar	10		
		TransaksiController	Fullkode	165	Komentar	67		
		updateData	Fullkode	17	Komentar	0		
			Total	1110	Total	356		
					Folder Controller	0.3207207207		
	Folder Middleware							
		Authenticate	Fullkode	17	Komentar	5		
		CheckForMaintenanceMode	Fullkode	12	Komentar	4		
		EncryptCookies	Fullkode	13	Komentar	4		
		isAdmin	Fullkode	20	Komentar	6		
		RedirectAuthenticated	Fullkode	22	Komentar	7		

Concisness | BUANGAN CONSisnes | SelfDescriptive | Modularity | Buangan modularity | Consistency

Gambar 4.2 *Spreadsheets* Pencatatan Metrik

4.2 Eksekusi Pengujian (*Test Execution*)

Pengujian dilakukan berdasarkan *test plan* dan test *case* yang telah dibuat pada tahapan sebelumnya yang terbagi menjadi tiga faktor nonfungsional. Faktor-faktor tersebut dibagi sesuai dengan karakteristiknya masing-masing. Tahap pengujian ini berdasarkan pada karakteristik dari masing-masing faktor nonfungsional tersebut.

Dalam melakukan pengujian menggunakan metode *McCall's Factor* hanya akan dihitung berdasarkan dua fase atau bagian, yaitu fase desain dan fase implementasi. Pembahasan fase desain akan mengacu pada dokumentasi pembuatan aplikasi Ivent yang telah *developer* buat dan untuk fase implementasi akan mengacu pada kode yang telah diketik oleh *developer*. Fase desain memiliki tiga modul yang akan dibahas berdasarkan pada dokumen dengan melihat *activity diagram* dan *use case diagram* yaitu, modul *app*, *auth*, dan *admin*. Fase

implementasi terdapat 8 modul yang *developer* ketik kode pengembangan aplikasinya, yaitu modul *admin*, modul *auth*, modul *controller*, modul *middleware*, modul *app*, modul *HTTP*, modul *migration*, dan *seeds*. Pengujian dari setiap karakteristik adalah sebagai berikut:

4.2.1 *Conciseness*

Karakteristik ini hanya melakukan pengujian pada fase implementasi. Pengujian ini menggunakan metode *Halstead Measure* menggunakan Persamaan (2.3) dan (2.4). Persamaan tersebut digunakan untuk menguji modul implementasi, yaitu modul *admin*, *auth*, *controller*, *middleware*, *app*, *HTTP*, *migration*, dan *seeds*. Dalam mencari *Halstead Measure* diperlukan jumlah operator dan operan, kemudian total penggunaan operator dan operan pada sebuah modul. Operator merupakan sebuah fungsi atau simbol yang berguna untuk melakukan suatu operasi, sedangkan operan adalah sebuah argumen bersama dengan operator, contoh dari operan adalah sebuah variabel atau konstanta. Untuk mencari perhitungan di atas pada setiap modul adalah sebagai berikut.

Modul *Admin*

Modul *admin* terdapat 35 jenis operator dan 158 jumlah total operator yang digunakan. Modul *admin* juga terdapat 24 jenis operan dan 56 jumlah total operan. Berdasarkan informasi yang telah didapatkan, maka tahap selanjutnya yaitu menghitung nilai Nc dan No sebagai berikut.

$$N_c = 35 \log_2 35 + 24 \log_2 24$$

$$N_c = 105$$

$$N_o = 158 + 56$$

$$N_o = 214$$

Setelah menemukan nilai dari Nc dan No maka selanjutnya, yaitu mencari nilai dari *Halstead Measure* dengan Persamaan (2. 5). Nilai dari pembagian $\frac{|105-214|}{214} = 0{,}5$ selanjutnya perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|105-214|}{214} = 0{,}49$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *Admin* yaitu 0,49.

Modul *Auth*

Modul *auth* terdapat 36 jenis operator dan 238 merupakan jumlah total operator yang digunakan. Modul *auth* juga terdapat 62 jenis operan dan 128 jumlah total operan. Setelah

mendapatkan nilai dari operator dan operan, maka tahap selanjutnya yaitu menghitung nilai dari Nc dan No sebagai berikut.

$$N_c = 36\log_2 36 + 62\log_2 62$$
$$N_c = 196{,}66$$
$$N_o = 238 + 128$$
$$N_o = 366$$

Setelah menemukan nilai dari Nc dan No maka tahap selanjutnya yaitu mencari nilai dari *Halstead Measure* dengan rumus yang telah dijelaskan sebelumnya. Nilai dari pembagian $\frac{|196{,}66-366|}{366} = 0{,}46$ kurang dari 1, maka perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|196{,}66-366|}{366} = 0{,}54$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *Auth* yaitu 0,54.

Modul *Controller*

Terdapat 91 jenis operator dan 2063 jumlah total operator yang digunakan pada modul *controller*. Terdapat juga 228 jenis operan dan 1068 jumlah total operan yang digunakan pada modul *controller*. Setelah mendapatkan nilai dari operator dan operan, maka tahap berikutnya yaitu menghitung nilai dari Nc dan No sebagai berikut.

$$N_c = 91\log_2 91 + 228\log_2 228$$
$$N_c = 811{,}91$$
$$N_o = 2063 + 1068$$
$$N_o = 3131$$

Setelah didapatkan nilai dari Nc dan No maka tahap berikutnya yaitu mencari nilai dari *Halstead Measure* dengan rumus seperti penjelasan sebelumnya. Nilai dari pembagian $\frac{|811{,}91-3131|}{3131} = 0{,}74$ kurang dari 1, maka perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|811{,}91-3131|}{3131} = 0{,}26$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *Controller* yaitu 0,26.

Modul *Middleware*

Terdapat 30 jenis operator dan 124 merupakan jumlah total operator yang digunakan pada modul *middleware*. Terdapat juga 38 jenis operan dan 71 jumlah total operan yang digunakan pada modul *middleware*. Setelah mendapatkan nilai dari operator dan operan, maka tahap selanjutnya yaitu menghitung nilai dari Nc dan No sebagai berikut.

$$N_c = 30\log_2 30 + 38\log_2 38$$
$$N_c = 124{,}82$$
$$N_o = 124 + 71$$
$$N_o = 195$$

Setelah didapatkan nilai dari Nc dan No maka tahap selanjutnya yaitu mencari nilai dari *Halstead Measure*. Nilai dari pembagian $\frac{|124{,}82-195|}{195} = 0{,}36$ kurang dari 1, maka perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|124{,}82-195|}{195} = 0{,}64$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *Middleware* yaitu 0,64.

Modul *HTTP*

Pada modul *http* terdapat 11 jenis operator dan 107 merupakan jumlah total operator yang digunakan. Pada modul *http* juga terdapat 42 jenis operan dan 44 merupakan jumlah total operan yang digunakan. Setelah mendapatkan nilai dari operator dan operan, maka tahap selanjutnya yaitu menghitung nilai dari Nc dan No sebagai berikut.

$$N_c = 11\log_2 11 + 42\log_2 42$$
$$N_c = 95{,}59$$
$$N_o = 107 + 44$$
$$N_o = 151$$

Setelah didapatkan nilai dari Nc dan No maka tahap selanjutnya yaitu mencari nilai dari *Halstead Measure*. Nilai dari pembagian $\frac{|95{,}59-151|}{151} = 0{,}37$ kurang dari 1, maka perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|95{,}59-151|}{151} = 0{,}63$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *HTTP* yaitu 0,63.

Modul *App*

Terdapat 29 jenis operator dan 251 merupakan jumlah total operator yang digunakan pada modul *app*. Terdapat juga 73 jenis operan dan 167 merupakan jumlah total operan yang digunakan pada modul *app*. Setelah mendapatkan nilai dari operator dan operan, maka tahap selanjutnya yaitu menghitung nilai dari Nc dan No sebagai berikut.

$$N_c = 29 \log_2 29 + 73 \log_2 73$$
$$N_c = 209{,}14$$
$$N_o = 251 + 167$$
$$N_o = 418$$

Setelah didapatkan nilai dari Nc dan No maka tahap selanjutnya yaitu mencari nilai dari *Halstead Measure*. Nilai dari pembagian $\frac{|209{,}14-418|}{418} = 0{,}50$ kurang dari 1, maka perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|209{,}14-418|}{418} = 0{,}50$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *App* yaitu 0,50.

Modul *Migrations*

Modul *migrations* memiliki 38 jenis operator dan 700 merupakan jumlah total operator yang digunakan. Terdapat juga 74 jenis operan dan 310 merupakan jumlah total operan yang digunakan pada modul *migrations*. Setelah mendapatkan nilai dari operator dan operan, maka tahap selanjutnya yaitu menghitung nilai Nc dan No sebagai berikut.

$$N_c = 38 \log_2 38 + 74 \log_2 74$$
$$N_c = 232{,}07$$
$$N_o = 700 + 310$$
$$N_o = 1010$$

Setelah didapatkan nilai dari Nc dan No maka tahap selanjutnya yaitu mencari nilai dari *Halstead Measure*. Nilai dari pembagian $\frac{|232{,}07-1010|}{1010} = 0{,}77$ kurang dari 1, maka perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|209{,}14-418|}{418} = 0{,}23$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *Migrations* yaitu 0,23.

Modul *Seeds*

Modul *seeds* memiliki 18 jenis operator dan 86 merupakan jumlah total operator yang digunakan. Terdapat juga 28 jenis operan dan 56 merupakan jumlah total operan yang digunakan pada modul *seeds*. Setelah mendapatkan nilai dari operator dan operan, maka tahap selanjutnya yaitu menghitung nilai Nc dan No sebagai berikut.

$$N_c = 18\log_2 18 + 28\log_2 28$$
$$N_c = 76{,}96$$
$$N_o = 86 + 56$$
$$N_o = 142$$

Setelah didapatkan nilai dari Nc dan No maka tahap selanjutnya yaitu mencari nilai dari *Halstead Measure*. Nilai dari pembagian $\frac{|76{,}96-142|}{142} = 0{,}46$ kurang dari 1, maka perhitungan menggunakan rumus Persamaan (2. 6), $1 - \frac{|209{,}14-418|}{418} = 0{,}23$. Maka dari itu, nilai yang didapatkan dari karakteristik *Conciseness* pada modul *Seeds* yaitu 0,56.

Dari nilai yang telah didapatkan untuk setiap modul, selanjutnya nilai-nilai tersebut akan dirata-rata menggunakan rumus Persamaan (4.1) seperti berikut.

$$\frac{Sum\ Halstead'sMeasure\ for\ each\ module}{Total\ \#\ module} \quad (4.1)$$

$$\frac{0{,}49 + 0{,}53 + 0{,}26 + 0{,}64 + 0{,}63 + 0{,}50 + 0{,}23 + 0{,}24}{8} = 0{,}48$$

4.2.2 *Self Descriptive*

Karakteristik ini akan dilakukan pengujian pada fase implementasi. Karakteristik ini dibagi menjadi tiga aturan yang perlu dihitung, yaitu *quantity of comments*, *effectiveness of comments measure*, dan *descriptiveness of implementation language measure*.

Quantity Of Comments

Aturan ini dihitung dengan melihat banyaknya komentar yang ada pada sebuah program dibagi dengan total kode program. Baris kosong yang ada pada kode program tidak akan dihitung, aturan ini dihitung menggunakan rumus Persamaan (4.2).

$$\frac{Baris\ Komentar}{Seluruh\ kode\ program} \quad (4.2)$$

a. Modul *Admin*

Modul *admin* memiliki 180 baris kode program dan 49 baris komentar pada seluruh kode. Dari data tersebut dapat dihitung nilai *quantity of comments* untuk modul *admin* dengan menggunakan rumus seperti Persamaan (4.2). Maka dari itu, nilai yang didapatkan dari aturan *Quantity of Comments* pada modul *Admin* bernilai 0,27.

$$\frac{49}{180} = 0{,}27$$

b. Modul *Auth*

Selanjutnya, modul *auth* memiliki 207 total baris kode program dan terdapat 94 total baris komentar. Oleh karena itu, dapat dihitung nilai aturan *quantity of comments* untuk modul *auth* dengan menggunakan rumus seperti Persamaan (4.2). Maka dari itu, nilai yang didapatkan dari aturan *Quantity of Comments* pada modul *Auth* yaitu 0,45.

$$\frac{94}{207} = 0{,}45$$

c. Modul *Controller*

Modul *controller* memiliki kode yang lebih banyak dibandingkan dengan modul lainnya dengan jumlah 1110 baris kode program dan 356 total baris komentar. Oleh karena itu, nilai aturan *quantity of comments* untuk modul *controller* ini dapat dicari dengan menggunakan rumus seperti Persamaan (4.2). Maka dari itu, nilai yang didapatkan dari aturan *Quantity of Comments* pada modul *Controller* yaitu 0,32.

$$\frac{356}{1110} = 0{,}32$$

d. Modul *Middleware*

Terdapat 127 total baris kode program pada modul *middleware* dan terdapat 42 total baris komentar yang ditemukan. Nilai aturan *quantity of comments* untuk modul *middleware* dapat dihitung dengan menggunakan rumus seperti Persamaan (4.2). Maka dari itu, nilai yang didapatkan dari aturan *Quantity of Comments* pada modul *middleware* yaitu 0,33.

$$\frac{42}{127} = 0{,}33$$

e. Modul *HTTP*

Selanjutnya, modul *http* memiliki baris kode program yang paling sedikit dibandingkan dengan modul lainnya dengan jumlah 56 baris kode program dan juga jumlah baris komentar yang paling sedikit yaitu berjumlah 15 baris komentar. Nilai aturan *quantity of comments* untuk modul *http* dapat dihitung dengan menggunakan rumus seperti Persamaan (4.2). Dari perhitungan yang dilakukan, maka nilai dari aturan *Quantity of Comments* yang didapatkan pada modul *HTTP* yaitu 0,27.

$$\frac{15}{56} = 0{,}27$$

f. Modul *App*

Modul *app* memiliki 184 total baris kode program dan terdapat 21 total baris komentar. Dapat dihitung nilai dari aturan *quantity of comments* pada modul *app* dengan menggunakan rumus seperti Persamaan (4.2). Maka dari itu, nilai yang didapatkan dari aturan *Quantity of Comments* pada modul *App* yaitu 0,11.

$$\frac{21}{184} = 0{,}11$$

g. Modul *Migrations*

Selanjutnya, terdapat 401 total baris kode program pada modul *migrations* dan terdapat 108 total baris komentar. Nilai aturan *quantity of comments* pada modul *migrations* dapat dihitung dengan menggunakan rumus seperti Persamaan (4.2). Maka dari itu, nilai yang didapatkan dari aturan *Quantity of Comments* pada modul *Migrations* yaitu 0,27.

$$\frac{108}{401} = 0{,}27$$

h. Modul *Seeds*

Modul terakhir yaitu modul *seeds* terdapat 70 total baris kode program dan terdapat 17 total baris komentar. Untuk mendapatkan nilai dari aturan *quantity of comments* pada modul *seeds* dihitung dengan menggunakan rumus seperti Persamaan (4.2). Maka didapatkan nilai dari aturan *Quantity of Comments* pada modul *Seeds* yaitu 0,24.

$$\frac{17}{70} = 0{,}24$$

Dari nilai yang telah didapatkan untuk setiap modul, selanjutnya nilai-nilai tersebut akan dirata-rata menggunakan rumus Persamaan (4.3).

$$\frac{Penjumlahan\ Quantity\ of\ Comment\ Setiap\ Module}{Total\ \#\ module} \tag{4.3}$$

$$\frac{0{,}27 + 0{,}45 + 0{,}32 + 0{,}33 + 0{,}27 + 0{,}11 + 0{,}27 + 0{,}24}{8} = 0{,}28$$

Effectiveness Of Comments Measure

Elemen ini dibagi menjadi beberapa aturan yang perlu diamati, elemen-elemen tersebut adalah sebagai berikut, modul memiliki komentar prolog berformat standar, penulisan kode komentar yang seragam, isi dari komentar yang sesuai dengan logika pada kode, semua kode yang bersifat *machine dependent* harus dijelaskan pada komentar, semua kode yang bersifat *non standard* HOL *(High Order Language) statement* juga harus dijelaskan pada komentar, semua variabel dari suatu atribut harus dijelaskan pada komentar, serta komentar tidak hanya mengulang kode yang dituliskan pada program. Dalam mendapati nilai *effectiveness of comments*, perlu dilakukan perhitungan pada kode program apakah kode program telah sesuai dengan standar *effectiveness of comments*. Rumus Persamaan (4.4) digunakan untuk melakukan perhitungan pada aturan ini.

$$\frac{Total\ Score\ From\ Applicable\ Elements}{Total\ applicable\ elements} \tag{4.4}$$

a. Modul memiliki komentar prolog berformat standar

Aturan ini merupakan menilai komentar pada sebuah kode telah sesuai dengan standar atau belum. Standar tersebut adalah sebuah komentar perlu berisikan nama modul, *author, date, purpose, input, output, function, assumption, limitations and restrictions, accuracy requirements, error recovery procedures, references.* Aturan ini dihitung dengan rumus perhitungan Persamaan (4.5) seperti berikut.

$$1 - \frac{Module\ Violate\ Rule}{Total\ module} \tag{4.5}$$

Diketahui pada kode program semua modul tidak mengikuti aturan ini. Maka dari itu, perhitungan pada aturan ini mendapatkan nilai yaitu bernilai 0.

$$1 - \frac{8}{8} = 0$$

b. Penulisan kode komentar yang seragam

Penggunaan kode yang seragam dalam membentuk komentar akan memudahkan dalaman menentukan kode mana yang bersifat sebagai komentar atau bukan. Aturan ini akan menggunakan rumus Persamaan (4.5), diketahui kode program telah sesuai pada aturan ini sehingga dari perhitungan mendapatkan nilai 1.

$$1 - \frac{0}{8} = 1$$

c. Isi dari komentar yang sesuai dengan logika pada kode

Penggunaan dari komentar harus berisikan penjelasan yang sesuai dengan logika pada setiap kode program. Maka dari itu, aturan ini menjelaskan modul yang tidak mengikuti aturan. Aturan ini akan menggunakan rumus Persamaan (4.5), dalam kode program diketahui semua modul tidak mengikuti aturan ini, maka didapatkan nilai pada aturan ini yaitu bernilai 0.

$$1 - \frac{8}{8} = 0$$

d. Semua kode yang bersifat *machine dependent* harus dijelaskan pada komentar

Kode program yang bersifat *machine dependent* harus diberikan komentar yang bertujuan untuk menjelaskan fungsi dan sifat dari kode tersebut. Sehingga perhitungan pada aturan ini yaitu modul yang tidak memiliki komentar yang bersifat *machine dependent*. Oleh karena itu, untuk aturan ini akan menggunakan rumus Persamaan (4.5). Tetapi dalam kode program tidak ditemukan kode yang bersifat *machine dependent* karena bahasa *php* yang digunakan merupakan bahasa *machine independent*. Oleh karena itu, seluruh device dapat mengerti bahasa ini, maka dari itu nilai pada aturan ini yaitu *Not Available*.

e. Semua kode yang bersifat *non standard* HOL *(High Order Language)* statement juga harus dijelaskan pada komentar

Seperti penjelasan pada poin sebelumnya bahwa komentar tidak hanya menjelaskan fungsi dari sebuah kode, tetapi juga menjelaskan sifat kode tersebut. Dalam kode program tidak ditemukan kode yang bersifat *non standard* HOL *(High Order Language)* statement karena bahasa *php* merupakan bahasa tingkat tinggi, sehingga bahasa *php* bersifat *standard* HOL *(High Order Language)* statement. Maka dari itu, nilai pada aturan ini yaitu *Not Available*.

f. Semua variabel dari suatu atribut harus dijelaskan pada komentar

Penggunaan komentar harus menjelaskan mengenai kegunaan semua variabel dari suatu atribut. Maka dari itu, perhitungannya didasari oleh jumlah modul yang tidak sesuai dengan aturan ini. Rumus yang digunakan untuk perhitungan pada aturan ini yaitu Persamaan (4.5). Terdapat 7 modul yang tidak mengikuti aturan dan terdapat 1 modul yang mengikuti aturan yaitu modul *controller*. Dalam modul *controller* terdapat penulisan kode

yang dijelaskan pada sebuah komentar. Dari perhitungan yang dilakukan, didapatkan nilai untuk aturan ini yaitu bernilai 0,13.

$$1 - \frac{7}{8} = 0{,}13$$

g. Komentar tidak hanya mengulang kode yang dituliskan pada program

Komentar ditulis untuk menjelaskan maksud dari kode program tersebut dan tidak hanya menulis ulang kode program. Maka dari itu, perhitungannya merupakan penjumlahan dari komentar yang tidak menjelaskan alasannya kemudian dibagi dengan total modul. Rumus yang digunakan untuk perhitungan untuk aturan ini yaitu Persamaan (4.5). Diketahui pada kode program semua modul tidak mengikuti aturan ini dengan hanya menulis ulang kode program. Dari perhitungan yang dilakukan, nilai untuk aturan ini yaitu bernilai 0.

$$1 - \frac{8}{8} = 0$$

Dari nilai yang telah didapatkan untuk setiap aturan turunan yang telah dihitung pada seluruh modul, selanjutnya nilai-nilai tersebut akan dirata-rata menggunakan rumus Persamaan (4.4).

$$\frac{0 + 1 + 0 + 0{,}13 + 0}{5} = 0{,}23$$

Descriptiveness of implementation language measure

Aturan ini dapat dihitung dengan melihat penulisan kode yang digunakan pada program Ivent. Aturan ini dibagi menjadi beberapa aturan, yaitu bahasa HOL jauh lebih deskriptif dibandingkan *assembly language*, format standar dalam penulisan modul seperti, komentar prologue, pernyataan deklaratif, penulisan nama variabel yang subyektif dengan mempertimbangkan deskripsi diri, teknik seperti pemblokiran, indentasi, pembuatan paragraf untuk konstruksi tertentu telah ditetapkan dengan baik dan harus diikuti secara seragam, satu pernyataan per baris, tidak ada kata perintah yang digunakan sebagai variabel. Maka dari itu, perhitungan pada aturan ini dapat dihitung dengan menjumlah semua aturan dibagi dengan total aturan menggunakan rumus Persamaan (4.4).

a. Bahasa HOL jauh lebih deskriptif dibandingkan *assembly language*

Aturan ini melihat jenis kode yang digunakan menggunakan *High Order Language* atau bahasa *assembly* dan *machine language* lainnya. Penggunaan *High Order Language* lebih

baik dibandingkan *assembly language* karena lebih mudah dimengerti, dan perhitungan untuk aturan ini menggunakan rumus Persamaan (4.6).

$$1 - \frac{Module\ with\ Direct\ Code}{Total\ module} \tag{4.6}$$

Semua module yang digunakan telah menggunakan bahasa *High Order*, karena bahasa yang digunakan merupakan bahasa *php* yang telah mengikuti aturan tersebut. Aturan ini dihitung sesuai rumus dan menghasilkan nilai 1.

$$1 - \frac{0}{8} = 1$$

b. Format standar dalam penulisan modul seperti, komentar prologue, pernyataan deklaratif
Aturan ini melihat penggunaan format spesifik yang seragam dalam kode yang digunakan. Artinya penulisan seperti *executable statements* dan penulisan format komentar ditulis dengan cara seragam, sehingga akan mempermudah *developer* untuk memahami kode yang digunakan. Dalam aturan ini terdapat 2 module yang tidak mengikuti aturan, yaitu modul *HTTP* dan *migrations*. Hal tersebut karena kedua modul tidak memiliki komentar *prologue* yang dapat memudahkan *developer* dalam memahami kode, dan perhitungan yang dilakukan untuk aturan ini menggunakan rumus Persamaan (4.5). Nilai yang didapatkan dari perhitungan untuk aturan ini bernilai 0,75.

$$1 - \frac{2}{8} = 0{,}75$$

c. Penulisan nama variabel yang subyektif dengan mempertimbangkan deskripsi diri
Aturan ini menilai penggunaan variabel yang digunakan pada program, apakah variabel yang digunakan telah subyektif dan sesuai dengan fungsinya atau tidak. Maka dari itu, perhitungan pada aturan ini menggunakan rumus Persamaan (4.5). Dalam modul controller dan admin terdapat variabel $q yang tidak mengikuti aturan ini, sehingga kedua modul tersebut dianggap tidak mengikuti aturan ini. Oleh karena itu, nilai yang didapatkan dari perhitungan untuk aturan ini bernilai 0,75.

$$1 - \frac{2}{8} = 0{,}75$$

d. Teknik seperti pemblokiran, indentasi, pembuatan paragraf untuk konstruksi tertentu telah ditetapkan dengan baik dan harus diikuti secara seragam

Aturan ini melihat teknik konstruksi pada kode, contohnya seperti *indentasi*. Aturan ini berguna untuk membuat kode menjadi lebih rapi, sehingga akan lebih mudah untuk dimengerti. Perhitungan dilakukan dengan menggunakan rumus Persamaan (4.5), semua modul telah mengikuti aturan ini sehingga dapat dilakukan perhitungan dan didapatkan nilai 1.

$$1 - \frac{0}{8} = 1$$

e. Satu pernyataan per baris

Aturan ini melihat pada *continuation statement* dan *multiple statement* pada kode program. Sehingga pengukurannya merupakan jumlah kelanjutan ditambah dengan jumlah beberapa baris pernyataan dibagi dengan jumlah total baris untuk setiap modul pada kode. Perhitungan yang dilakukan untuk aturan in menggunakan rumus Persamaan (4.7).

$$1 - \frac{Statement\ line}{Total\ line} \tag{4.7}$$

1. Modul *Admin*

   Modul *admin* memiliki 2 *continuation statement* serta 2 *multiple statement* dan modul ini memiliki 176 total baris kode, untuk perhitungannya dapat dilakukan dengan Persaman (4.7). Dari perhitungan yang telah dilakukan, modul *Admin* memiliki nilai aturan yaitu 0,98.

   $$1 - \frac{4}{176} = 0{,}98$$

2. Modul *Auth*

   Modul *auth* memiliki 4 *continuation statement* serta 9 *multiple statement* dan modul ini memiliki 219 total baris kode, sehingga perhitungan dapat dilakukan dengan menggunakan Persamaan (4.7). Dari perhitungan yang telah dilakukan, modul *Auth* memiliki nilai aturan yang bernilai 0,94.

   $$1 - \frac{13}{219} = 0{,}94$$

3. Modul *Controller*

Modul *controller* memiliki 32 *continuation statement* serta 80 *multiple statement* dan modul ini memiliki 1111 total baris kode, sehingga perhitungan dapat dilakukan dengan menggunakan Persamaan (4.7). Dari perhitungan yang telah dilakukan, modul *Controller* memiliki nilai aturan yang bernilai 0,90.

$$1 - \frac{112}{1111} = 0{,}90$$

4. Modul *Middleware*

   Modul *middleware* memiliki 1 *continuation statement* serta 2 *multiple statement* dan modul ini memiliki 126 total baris kode, sehingga perhitungan dapat dilakukan dengan menggunakan Persamaan (4.7). Dari perhitungan yang telah dilakukan, modul *Middleware* memiliki nilai aturan yang bernilai 0,98.

$$1 - \frac{3}{126} = 0{,}98$$

5. Modul *HTTP*

   Modul *http* memiliki 5 *continuation statement* serta 27 *multiple statement* dan modul ini memiliki 56 total baris kode, sehingga perhitungan dapat dilakukan dengan menggunakan Persamaan (4.7). Dari perhitungan yang telah dilakukan, modul *HTTP* memiliki nilai aturan yang bernilai 0,43.

$$1 - \frac{32}{56} = 0{,}43$$

6. Modul *App*

   Modul *app* memiliki 7 *continuation statement* serta 50 *multiple statement* dan modul ini memiliki 184 total baris kode, sehingga perhitungan dapat dilakukan dengan menggunakan Persamaan (4.7). Dari perhitungan yang telah dilakukan, modul *App* memiliki nilai aturan yang bernilai 0,69.

$$1 - \frac{57}{184} = 0{,}69$$

7. Modul *Migration*

   Modul *migration* tidak ditemukan nilai dari *continuation statement* maupun nilai dari *multiple statement*. Maka dari itu, modul *migration* memiliki nilai aturan yang bernilai 1 sesuai dengan perhitungan menggunakan rumus Persamaan (4.7).

$$1 - \frac{0}{400} = 1$$

8. Modul *Seeds*

   Modul *seeds* memiliki 1 *continuation statement* serta 12 *multiple statement* dan modul ini memiliki 70 total baris kode, sehingga perhitungan dapat dilakukan dengan menggunakan Persamaan (4.7). Dari perhitungan yang telah dilakukan, modul *Seeds* memiliki nilai aturan yang bernilai 0,81.

$$1 - \frac{13}{70} = 0{,}81$$

   Nilai dari setiap modul dijumlahkan sehingga didapatkan hasil 6,73, kemudian nilai tersebut dibagi dengan jumlah modul dan didapatkan nilai 0,84 untuk aturan ini.

f. Tidak ada kata perintah yang digunakan sebagai variabel

Aturan ini melihat kata perintah yang digunakan sebagai variabel, contohnya adalah perintah *WHILE* yang digunakan untuk looping, namun digunakan sebagai variabel. Hal tersebut akan sangat membingungkan, sehingga perhitungan untuk aturan ini menggunakan rumus Persamaan (4.5). Dalam seluruh modul yang ada tidak ditemukan kata perintah yang digunakan menjadi variabel. Dari perhitungan yang telah dilakukan, aturan ini memiliki nilai yang bernilai 1.

$$1 - \frac{0}{8} = 1$$

Dari nilai yang telah didapatkan untuk setiap aturan turunan yang telah dihitung pada seluruh modul, selanjutnya nilai-nilai tersebut akan dirata-rata menggunakan rumus Persamaan (4.4). Nilai rata-rata yang didapatkan yaitu 0,89.

$$\frac{1 + 0{,}75 + 0{,}75 + 1 + 0{,}84 + 1}{6} = 0{,}89$$

4.2.3 *Modularity*

Karakteristik ini terdapat dua aturan yang perlu dihitung, yaitu *stability measure* dan *modular implementation*. Aturan *stability measure* akan dihitung berdasarkan fase desain, sedangkan *modular implementation* akan dihitung berdasarkan fase desain dan implementasi.

Stability Measure

Perhitungan pada aturan ini didasari oleh perhitungan G. Meyer yang berdasarkan kekuatan modul dan modul kopling. Kekuatan modul merupakan hubungan antara aturan dengan karakteristik ketika terjadi suatu perubahan pada modul. Modul kopling adalah hubungan antar modul ketika terjadi sebuah perubahan pada modul. Maka dari itu, aturan ini diperlukan sebuah dokumen yang menjelaskan adanya sebuah perubahan pada modul. Namun, pada dokumentasi tidak terdapat data yang diperlukan, sehingga kekuatan modul dan modul kopling tidak dapat dihitung. Oleh karena itu, perhitungan ini tidak dapat dilakukan sehingga memiliki nilai *Not Available*.

Modular Implementation

Perhitungan ini akan menghitung lima aturan, yaitu struktur hirarki, ukuran modul harus sesuai dengan standar yaitu 100 pernyataan prosedural, semua modul mewakili satu fungsi, mengontrol parameter yang ditentukan dengan memanggil modul, modul tidak berbagi penyimpanan sederhana.

a. Struktur Hirarki

Aturan ini akan dihitung melalui fase desain dan implementasi. Pada fase desain diperlukan sebuah dokumen yang membahas bahwa modul telah menerapkan sistem *top down design*. Namun pada fase desain tidak ditemukan data tersebut, sehingga perhitungan tidak dapat dilakukan dan bernilai *Not Available*. Pada fase implementasi penulisan kode telah menerapkan sistem *top down*. Artinya modul yang ada pada sistem ini memiliki modul lain yang lebih kecil dengan fungsi nya masing - masing. Contoh nya adalah modul *controller* yang di dalamnya terdapat modul *auth*, kemudian modul *auth* memiliki modul yang lebih kecil lagi yang berfungsi untuk *login* pengguna. Perhitungan pada aturan ini menggunakan rumus Persamaan (4.8). Semua modul telah mengikuti aturan ini, sehingga perhitungannya, setelah dilakukan perhitungan dapat diketahui bahwa aturan ini bernilai 1 untuk fase implementasi.

$$1 - \frac{Violation\ of\ Hierarchy}{Total\ \#\ module} \tag{4.8}$$

b. Ukuran modul harus sesuai dengan standar yaitu 100 pernyataan prosedural

Aturan ini dihitung hanya melalui fase implementasi. Aturan ini melihat kode yang ditulis oleh developer untuk mencari modul mana yang melebihi 100 pernyataan prosedural. Pernyataan prosedural merupakan pernyataan yang dapat dieksekusi, contoh *condition statement* seperti *IF ELSE, for statement, do statement*, pemanggilan fungsi, dan lain sebagainya. Perhitungan pada aturan ini menggunakan rumus Persamaan (4.9).

$$1 - \frac{Module > 100}{Total\ \#\ module} \tag{4.9}$$

Terdapat satu modul yang melebihi 100 pernyataan prosedural, yaitu modul *controller* pada modul *controller* terdapat 155 pernyataan prosedural. Berdasarkan perhitungan yang dilakukan, dapat diketahui bahwa aturan ini memiliki nilai 0.88 seperti berikut.

$$1 - \frac{1}{8} = 0{,}88$$

c. Semua modul mewakili satu fungsi

Aturan ini dilihat berdasarkan fase desain dan implementasi. Fase desain diperlukan dokumentasi yang menjelaskan bahwa terdapat modul yang memiliki satu fungsi. Namun, berdasarkan dokumentasi tidak didapatkan informasi akan hal tersebut. Oleh karena itu,

nilai yang didapatkan pada fase desain ini bernilai *not available.* Fase implementasi dihitung berdasarkan banyaknya modul yang hanya memiliki satu fungsi. Perhitungan aturan ini dapat dihitung menggunakan rumus Persamaan (4.5). Berdasarkan perhitungan yang dilakukan, dapat diketahui bahwa aturan ini bernilai 0 untuk fase implementasi.

$$1 - \frac{8}{8} = 0$$

d. Mengontrol parameter yang ditentukan dengan memanggil modul

Aturan ini akan membahas lebih lanjut terkait hubungan antar modul pada aturan satu dengan lebih lanjut. Aturan ini membahas pemanggilan modul yang mendefinisikan parameter kontrol, input data, output data, dan kontrol yang diperlukan dalam kode. Pemanggilan modul tersebut nilai yang telah didapatkan dari proses pemanggilan modul, nilai tersebut wajib dikembalikan pada fungsi pemanggil modul. Aturan ini memiliki dua fase pengujian, yaitu desain dan implementasi. Fase desain memerlukan data pada dokumentasi yang menjelaskan alur cerita penggunaan dari setiap modul dan hasil nya. Namun berdasarkan dokumentasi tidak ditemukan data tersebut , sehingga nilai yang didapatkan *Not Available*.

Dalam fase implementasi, aturan ini mencari penggunaan fungsi pemanggil untuk di eksekusi tanpa menggunakan perintah *return*. Berdasarkan penulisan kode didapatkan hasil sebagai berikut.

1. Pemanggilan modul berdasarkan parameter kontrol

   Aturan ini memerlukan sebuah fungsi yang berisikan parameter kontrol, pada akhir fungsi nilainya dikembalikan pada fungsi pemanggil. Contoh nya adalah pemanggilan modul alamat pengiriman ke modul alamat pengiriman *controller*. Modul alamat pengiriman terdapat parameter *id_user*, kemudian modul alamat pengiriman *controller* akan menggunakan parameter tersebut untuk mengeksekusi program. Fungsi yang dieksekusi pada modul alamat pengiriman *controller* harus dikembalikan lagi pada modul pemanggil. Perhitungan yang digunakan pada aturan ini menggunakan rumus Persamaan (4.5). Berdasarkan perhitungan tidak didapatkan modul yang melanggar aturan ini, dari data yang didapatkan penilaian pada aturan ini bernilai 1 sebagai berikut.

   $$1 - \frac{0}{8} = 1$$

2. Pemanggilan modul berdasarkan *input* data

   Aturan ini memerlukan sebuah fungsi yang berisikan *input* data yang digunakan pada modul lain untuk diolah nilainya. Contohnya pada modul *auth* berisikan fungsi *login*. Fungsi *login* tersebut digunakan pada modul *middleware* untuk diproses apakah *user* yang *login* tersebut adalah admin atau bukan. Kemudian nilai tersebut dikembalikan pada fungsi *login*. Perhitungan yang digunakan pada aturan ini menggunakan rumus Persamaan (4.5). Berdasarkan perhitungan tidak didapatkan modul yang melanggar aturan ini, dari data yang didapatkan penilaian pada aturan ini bernilai 1 sebagai berikut.

$$1 - \frac{0}{8} = 1$$

3. Pemanggilan modul berdasarkan *output* data

   Aturan ini memerlukan sebuah fungsi yang berisikan *output* dari sebuah modul, yang mana *output* tersebut akan digunakan pada modul lain untuk mengolah datanya. Contoh ketika modul *app* terdapat sebuah parameter *order* yang mana ketika parameter tersebut berisikan sudah dibayar, maka nilai itu akan dibaca oleh modul *admin controller* untuk mengolah data dan memberikan laporan penjualan kepada *admin controller*. Perhitungan yang digunakan pada aturan ini menggunakan rumus Persamaan (4.5). Berdasarkan perhitungan tidak didapatkan modul yang melanggar aturan ini, dari data yang didapatkan penilaian pada aturan ini bernilai 1 sebagai berikut.

$$1 - \frac{0}{8} = 1$$

4. Fungsi *control* harus dikembalikan pada pemanggil modul

   Aturan ini memerlukan sebuah fungsi proses data yang mana proses tersebut digunakan pada modul lainnya. Fungsi proses tersebut harus dikembalikan kepada modul pemanggilnya. Perhitungan yang digunakan pada aturan ini menggunakan rumus Persamaan (4.5). Berdasarkan perhitungan tidak didapatkan modul yang melanggar aturan ini, dari data yang didapatkan penilaian pada aturan ini bernilai 1 sebagai berikut.

$$1 - \frac{0}{8} = 1$$

e. Modul tidak berbagi penyimpanan sementara

Aturan ini dapat dihitung melalui dua fase, yaitu implementasi dan desain. Fase desain dokumentasi yang dicari adalah menjelaskan bahwa terdapat penyimpanan sementara dalam sistemnya. Namun tidak ditemukan penjelasan tersebut sehingga penilaian *Not Available*. Fase implementasi juga tidak didapatkan modul yang memiliki fungsi sebagai penyimpanan sementara. Perhitungan pada aturan ini ialah perhitungan biner, yaitu jika terdapat maka memiliki nilai 1, dan 0 jika tidak memiliki penyimpanan sementara. Oleh karena itu penilaian pada fase implementasi bernilai 0.

Fase desain memiliki nilai rata-rata yang diperoleh adalah *Not Available*, dikarenakan tidak ditemukannya dokumentasi mengenai aturan yang ditulis oleh *McCall* pada dokumentasi Ivent. Pada fase implementasi nilai rata-rata menggunakan rumus Persamaan (4.4), nilai rata-rata yang didapatkan yaitu 0,74 seperti berikut.

$$\frac{1 + 0{,}88 + 0 + 1 + 1 + 1 + 1 + 0}{8} = 0{,}74$$

4.2.4 *Simplicity*

Pada karakteristik ini terdapat 4 aturan, yaitu *design structure* aturan ini dapat dihitung melalui fase desain dan implementasi, lalu terdapat *use of structured language or structured language preprocessor* yang diukur melalui fase implementasi, kemudian *data and control flow complexity* yang dihitung melalui fase desain dan implementasi, dan terakhir *measure of simplicity of coding techniques* yang dihitung melalui fase implementasi.

Design Structure

Aturan ini akan dibagi menjadi 7 aturan yang perlu dihitung untuk mencari nilai dari *simplicity*. Setiap aturan akan dihitung melalui fase desain dan implementasi.

a. Desain diatur dalam mode *top down fashion*

Aturan ini diukur melalui fase desain dan implementasi, fase desain diukur dengan melihat apakah terdapat *flowchart* atau *activity diagram* dalam dokumentasi. Jika terdapat data tersebut maka aturan ini akan bernilai 1, jika tidak ada maka bernilai 0. Terdapat *activity diagram* dalam dokumentasi, sehingga aturan ini memiliki nilai 1. Dalam fase implementasi aturan ini mencari bahasa program, apakah bahasa yang digunakan merupakan bahasa yang digunakan untuk mengeksekusi program dari baris atas ke bawah.

Bahasa yang digunakan adalah *php* yang mana bahasa ini mengeksekusi perintah dari baris atas ke bawah, sehingga aturan ini akan bernilai 1.

b. Tidak ada fungsi yang duplikat

Aturan ini akan diukur melalui fase desain dan implementasi, fase desain mencari pada dokumentasi jika terdapat duplikat fungsi pada *activity diagram*. Perhitungan yang dilakukan pada aturan ini adalah bernilai 1 jika tidak memiliki fungsi duplikat dan bernilai 0 jika memiliki fungsi duplikat. Dari dokumentasi tidak terdapat fungsi yang duplikat, sehingga pada fase desain ini bernilai 1. Fase implementasi mencari duplikat fungsi yang ada pada kode program. Dalam kode program terdapat fungsi yang duplikat, contoh fungsi *update* dan cetak bulan. Oleh karena itu pada aturan ini akan bernilai 0.

c. Modul *Independence*

Aturan ini akan diuji melalui fase desain dan implementasi, fase desain mencari pada dokumentasi dengan melihat modul yang tidak bergantung pada *input* dan tujuan *output*. Dalam dokumentasi terdapat modul yang memiliki ketergantungan pada inputnya, yaitu pada modul *auth* dan *app*. Pada modul *auth* terdapat fungsi *register* dan *login* dalam mengisi data diri. Selanjutnya untuk modul *app* terdapat fungsi membuat pesanan yang memerlukan *login* akun terlebih dahulu. Oleh karena itu, perhitungan yang digunakan yaitu Persamaan (4.5). Berdasarkan dokumentasi terdapat dua modul yang tidak mengikuti aturan ini, dari data tersebut didapatkan nilai sebesar 0.33.

$$1 - \frac{2}{3} = 0{,}33$$

Berdasarkan data yang didapatkan penilaian sebesar 0.33. Kemudian fase implementasi, aturan ini dilihat dari kode program yang telah dibuat *developer*. Berdasarkan kode program diketahui bahwa seluruh program sangat bergantung pada *input* dan tujuan *output*. Oleh karena itu, perhitungan yang digunakan yaitu Persamaan (4.5), dari data tersebut didapatkan nilai sebesar 0.

d. Pemrosesan modul tidak bergantung pada pemrosesan sebelumnya

Aturan ini melihat dari fase desain dan implementasi, fase desain mencari dokumentasi yang menjelaskan pemrosesan dari sebuah modul, apakah bergantung dari pemrosesan sebelumnya atau tidak. Pada desain dapat diketahui bahwa semua modul sangat dipengaruhi oleh pemrosesan sebelumnya. Seperti modul *auth* yang bergantung pada fungsi *login* dan *register*. Kemudian fungsi *app* yang sangat bergantung pada modul *auth*

untuk *login* terlebih dahulu. Kemudian, fungsi *admin* yang sangat bergantung pada *user*. Oleh karena itu, perhitungan yang dilakukan pada aturan ini menggunakan Persamaan (4.5). Berdasarkan dokumentasi seluruh modul tidak mengikuti aturan ini, sehingga aturannya dihitung sebagai berikut.

$$1 - \frac{3}{3} = 0$$

Berdasarkan perhitungan tersebut didapatkan nilai sebesar 0. Kemudian, fase implementasi aturan ini dilihat pada kode program yang telah dibuat *developer*. Dari kode program seluruh modul sangat bergantung pada pemrosesan sebelumnya.

e. Setiap deskripsi modul mencakup *input*, *output*, pemrosesan, batasan

Aturan ini mencari informasi mengenai *input* dan *output* serta batasan yang ada pada dokumentasi. Dalam dokumentasi hanya terdapat *activity diagram* yang menjelaskan fungsi dari modul *app* dan *auth*. Oleh karena itu, hanya terdapat 1 modul saja yang tidak dijelaskan yaitu modul *admin.* Maka dari itu, perhitungan yang digunakan pada pengujian ini menggunakan Persamaan (4.5). Berdasarkan dokumentasi modul *admin* tidak mengikuti aturan ini, sehingga aturannya dapat dihitung sebagai berikut dan didapatkan nilai sebesar 0,67.

$$1 - \frac{1}{3} = 0{,}67$$

f. Setiap modul memiliki pintu masuk tunggal dan pintu keluar tunggal

Aturan ini dapat dihitung melalui fase desain dan implementasi, fase desain mencari informasi dari dokumentasi mengenai proses yang dilalui pada sebuah modul memiliki pintu masuk tunggal dan pintu keluar tunggal atau tidak. Dalam dokumentasi modul *app* dan *auth* memiliki *activity diagram* yang menjelaskan bahwa modul tersebut memiliki pintu masuk yang tunggal dan pintu keluar tunggal. Maka, perhitungan yang dilakukan pada fase desain ini menggunakan Persamaan (4.5). Berdasarkan dokumentasi diketahui bahwa modul *admin* yang tidak dijelaskan pemrosesannya. Maka dari itu hanya modul *admin* saja yang tidak mengikuti aturan ini, dari perhitungan didapatkan nilai 0,67 seperti berikut.

$$1 - \frac{1}{3} = 0{,}67$$

Fase implementasi pengujian dilakukan dengan melihat kode program, dari kode program diketahui bahwa seluruh modul hanya memiliki satu pintu masuk dan satu pintu keluaran. Maka dari itu, dilakukan perhitungan dan didapatkan nilai 1 seperti berikut.

$$1 - \frac{0}{8} = 1$$

g. Tidak ada data global

Aturan ini dihitung dengan melihat fase desain dan implementasi, perhitungan yang dilakukan pada aturan ini adalah jika memiliki data global akan bernilai 1, jika tidak memiliki data global akan bernilai 0. Fase desain mencari informasi dari dokumentasi apakah terdapat data global atau tidak. Setelah melihat dokumentasi, diketahui bahwa tidak ada informasi yang menjelaskan adanya data global. Maka dari itu, perhitungan untuk fase desain bernilai *Not Available.* Sedangkan untuk fase implementasi diketahui bahwa tidak ada kode yang ditulis oleh *developer* menggunakan data global. Oleh karena itu, perhitungan pada fase implementasi bernilai 0.

Selanjutnya, dari masing-masing nilai yang didapatkan untuk setiap aturan dengan melihat fase desain dan implementasi akan dirata-rata menggunakan rumus Persamaan (4.4). Nilai rata-rata yang didapatkan dari fase desain seperti berikut.

$$\frac{1 + 1 + 0{,}33 + 0 + 0{,}67 + 0{,}67}{6} = 0{,}61$$

Nilai rata-rata yang didapatkan dari fase desain seperti berikut.

$$\frac{1 + 0 + 0 + 0 + 1 + 0}{6} = 0{,}33$$

Use of structured language or structured language preprocessor

Aturan ini menguji dari fase implementasi dengan melihat penjelasan pada dokumentasi jika terdapat *structured language*. *Structured language* adalah kalimat pernyataan seperti *IFTHENELSE, DOWHILE, DOUNTIL,* dan *CASE.* Perhitungan untuk aturan ini adalah perhitungan *biner*, yang mana akan bernilai 1 jika memiliki *structured language* dan 0 jika tidak memiliki.

a. Modul *App*

Modul ini tidak memiliki *structured language*, sehingga nilai pada modul ini bernilai 0.

b. Modul *HTTP*

Modul ini tidak memiliki *structured language*, sehingga nilai pada modul ini bernilai 0.

c. Modul *Migration*

Modul ini tidak memiliki *structured language*, sehingga nilai pada modul ini bernilai 0.

d. Modul *Admin*

Modul ini memiliki *structured language,* yaitu *IFTHENELSE* sehingga nilai pada modul ini bernilai 1.

e. Modul *Auth*

Modul ini memiliki *structured language,* yaitu *IFTHENELSE* sehingga nilai pada modul ini bernilai 1.

f. Modul *Controller*

Modul ini memiliki *structured language,* yaitu *IFTHENELSE* dan juga *CASE* sehingga nilai pada modul ini bernilai 1.

g. Modul *Middleware*

Modul ini memiliki *structured language,* yaitu *IFTHENELSE* sehingga nilai pada modul ini bernilai 1.

h. Modul *Seeds*

Modul ini tidak memiliki *structured language*, sehingga nilai pada modul ini bernilai 0.

Setelah mengetahui nilai dari seluruh modul untuk aturan ini, maka nilai tersebut akan dirata-rata menggunakan rumus Persamaan (4.4) seperti berikut.

$$\frac{0+0+0+0+1+1+1+1}{8} = 0{,}5$$

Data and control flow complexity

Aturan ini akan dihitung melalui fase desain dan implementasi, fase desain dihitung dengan melihat *flowchart* yang ada pada dokumentasi dan melihat variabel yang sering digunakan. Berdasarkan dokumentasi diketahui terdapat dua modul yang memiliki *flowchart*, kedua modul tersebut adalah modul *app* dan modul *auth*. Modul *auth* terdapat dua variabel yang digunakan yaitu, *register* dan *login*. Sedangkan, modul *app* terdapat 4 variabel yang digunakan, yaitu membuat pesanan, melakukan transaksi, tambah produk, dan *edit* produk. Maka dari itu, didapatkan total variabel adalah 6. Perhitungan yang dilakukan untuk aturan ini menggunakan rumus Persamaan (4.10).

$$\frac{Variable\ Considered\ Live}{Total\ Variable} \tag{4.10}$$

a. Modul *App*

Berdasarkan hasil dokumentasi didapatkan dua variabel pada aturan ini, sehingga dari perhitungan didapatkan nilai 0,67 dan perhitungannya dilakukan sebagai berikut.

$$\frac{4}{6} = 0{,}67$$

b. Modul *Auth*

Berdasarkan hasil dokumentasi didapatkan dua variabel pada aturan ini, sehingga dari perhitungan didapatkan nilai 0,33 dan perhitungannya dilakukan sebagai berikut.

$$\frac{2}{6} = 0{,}33$$

Setelah mendapatkan nilai dari dua modul tersebut, maka nilai-nilai tersebut akan dihitung menggunakan rumus Persamaan (4.11). Dari perhitungan diketahui nilai yang didapatkan yaitu 0,5 seperti berikut.

$$\frac{Sum\ of\ Coplexity\ Measure\ For\ Each\ Module}{\#module} \tag{4.11}$$

$$\frac{0{,}67 + 0{,}33}{2} = 0{,}5$$

Selanjutnya untuk fase implementasi akan dilihat pada kode yang dibuat oleh *developer*.

a. Modul *Admin*

Modul ini memiliki 5 variabel dan terdapat 2 variabel yang sering digunakan yaitu, variabel *itemorder* dan variabel *cart*. Maka, dari perhitungan yang dilakukan untuk modul ini didapatkan nilai 0,4 dan perhitungannya sebagai berikut.

$$\frac{2}{5} = 0{,}4$$

b. Modul *Auth*

Modul ini memiliki 15 variabel dan terdapat 3 variabel yang sering digunakan yaitu, variabel input, data dan redirecto. Maka, dari perhitungan yang dilakukan untuk modul ini didapatkan nilai 0,2 dan perhitungannya sebagai berikut.

$$\frac{3}{15} = 0{,}2$$

c. Modul *Controller*

Modul ini memiliki 307 variabel dan terdapat 42 variabel yang sering digunakan, maka dari perhitungan yang dilakukan untuk modul ini didapatkan nilai 0,14 dan perhitungannya sebagai berikut.

$$\frac{42}{307} = 0{,}14$$

d. Modul *Middleware*

Modul ini memiliki 6 variabel dan terdapat 1 variabel yang sering digunakan yaitu, variabel *except*. Maka dari perhitungan yang dilakukan untuk modul ini didapatkan nilai 0,17 dan perhitungannya sebagai berikut.

$$\frac{1}{6} = 0{,}17$$

e. Modul *HTTP*

Modul ini terdapat 3 variabel dan tidak terdapat variabel yang sering digunakan, maka dari perhitungan yang dilakukan untuk modul ini didapatkan nilai 0 dan perhitungannya sebagai berikut.

$$\frac{0}{3} = 0$$

f. Modul *App*

Modul ini memiliki 28 variabel dan 3 variabel yang sering digunakan, yaitu variabel *table*, *fillable*, dan this. Maka dari perhitungan yang dilakukan untuk modul ini didapatkan nilai 0,11 dan perhitungannya sebagai berikut.

$$\frac{3}{28} = 0{,}11$$

g. Modul *Migration*

Modul ini tidak memiliki variabel dan hanya terdapat fungsi saja. Oleh karena itu, nilai pada modul ini adalah 0.

h. Modul *Seeds*

Modul ini memiliki 12 variabel dan terdapat 3 variabel yang sering digunakan yaitu, *user*, inputan, dan *value*. Maka dari perhitungan yang dilakukan untuk modul ini didapatkan nilai 0,25 dan perhitungannya sebagai berikut.

$$\frac{3}{12} = 0{,}25$$

Setelah mendapatkan nilai dari fase implementasi untuk setiap modul, maka nilai-nilai tersebut akan dihitung menggunakan rumus Persamaan (4.11). Dari perhitungan diketahui nilai yang didapatkan yaitu 0,16 seperti berikut.

$$\frac{0{,}4 + 0{,}2 + 0{,}14 + 0{,}17 + 0 + 0{,}11 + 0 + 0{,}25}{8} = 0{,}16$$

Measure Of Coding Simplicity

Aturan ini akan membahas dengan melihat fase implementasi, aturan ini terdapat 16 aturan yang harus dilakukan untuk mendapatkan nilai *measure of coding simplicity*.

a. Aliran modul dari atas ke bawah

   Perhitungan yang dilakukan pada aturan ini adalah dengan menggunakan perhitungan biner. Aturan ini dilihat pada kode yang telah dibuat oleh *developer*, apakah kode bergerak dari atas ke bawah dan memiliki struktur hirarki atau tidak. Struktur hirarki merupakan sebuah struktur yang terdapat *parent* dan *child* dalam sebuah pemrograman. Berdasarkan hasil pengujian diketahui bahwa kode yang dibentuk telah memiliki struktur hirarki dan bergerak dari baris atas ke bawah. Maka dari itu, nilai dari pengujian yang dilakukan adalah 1.

b. Kode yang memiliki operator Boolean yang majemuk yaitu melibatkan dua atau lebih boolean

   Aturan ini melihat dari kode yang dibuat oleh *developer* dengan mencari apakah terdapat penggunaan operator *boolean* yang lebih dari satu. Operator *boolean* merupakan operator yang menunjukkan 2 hasil, yaitu *true* dan *false*. Berdasarkan hasil pengujian diketahui bahwa tidak ada modul yang memiliki operator *boolean* yang rumit, sehingga nilai pada semua modul pengujian aturan ini akan bernilai 1.

c. Keluar masuk pada *loop*

   Pengujian untuk aturan ini melihat pada jumlah *loop* yang digunakan dalam kode program dengan melihat pintu masuk pada *loop*. Berdasarkan kode yang ditulis *developer*, terdapat dua modul yang memiliki *loop* yaitu modul *seeds* dan modul *controller*. *Loop* yang digunakan keduanya adalah *for each* dengan total *loop* dari dua modul tersebut adalah 3

loop. Modul *controller* menggunakan 2 *loops* dan modul *seeds* menggunakan 1 *loop*. Maka dari itu perhitungan yang dilakukan menggunakan rumus Persamaan (4.12).

$$\frac{Jumps\ in\ and\ out\ of\ loops}{Total\ \#loops} \tag{4.12}$$

$$\frac{3}{3} = 1$$

d. Modifikasi *loop*

Pengujian yang dilakukan untuk aturan ini akan melihat *loop* yang telah dimodifikasi. Namun, pada aturan ini berdasarkan kode yang telah ditulis oleh *developer*, tidak ditemukan sebuah *loop* yang telah dimodifikasi. Maka dari itu nilai pada pengujian ini akan dihitung dengan rumus Persamaan (4.13) seperti berikut.

$$1 - \frac{\#\ Loop\ indices\ modified}{Total\ \#loops} \tag{4.13}$$

$$1 - \frac{0}{3} = 1$$

e. Modul tidak memodifikasi sendiri

Pengujian yang dilakukan untuk aturan ini adalah dengan mencari kode yang dapat memodifikasi sendiri logika pemrosesannya. Berdasarkan hasil pengujian diketahui bahwa semua modul tidak memiliki kode yang dapat memodifikasi logika pemrosesan sendiri. Perhitungan yang dilakukan untuk aturan ini adalah perhitungan biner, nilai 1 jika tidak ada modul yang dapat memodifikasi sendiri dan 0 jika dapat memodifikasi sendiri. Maka dari itu, perhitungan yang dilakukan bernilai 1.

f. Semua argumen yang diteruskan ke modul bersifat parametrik

Pengujian ini mencari jika terdapat nilai konstanta dan data global yang digunakan menjadi variabel. Penilaian yang dilakukan pada aturan ini adalah 1 jika terdapat modul yang bersifat parametrik dan 0 jika tidak. Berdasarkan hasil pengujian diketahui bahwa tidak ada data konstanta dan data global yang digunakan menjadi variabel, sehingga nilai yang didapatkan pada aturan ini adalah 0.

g. Jumlah label pernyataan

Pengujian yang dilakukan untuk aturan ini dengan melihat banyaknya komentar yang ada pada kode program. Perhitungan yang dilakukan pada pengujian ini dapat dilakukan dengan rumus Persamaan (4.14) sebagai berikut.

$$1 - \frac{\#\ Labels}{\#executable\ statements} \tag{4.14}$$

Modul *admin* ini terdapat 83 komentar dengan 19 *executable statements*. Kemudian terdapat modul *auth* memiliki 94 komenar dengan 43 *executable statements.* Setelah itu, modul *controller* memiliki 357 komentar dengan 449 *executable statements*. Selanjutnya, modul *middleware* memiliki 41 komentar dengan 15 *executable statements*. Kemudian terdapat modul *http* memiliki 15 komentar dengan 29 *executable statements*. Selanjutnya modul *app* memiliki 20 komentar dengan 69 *executable statements*. Setelah itu, modul *migrations* memiliki 77 komentar dengan 17 *executable statements*. Terakhir pada modul *seeds* memiliki 17 komentar dengan 29 *executable statements*. Kemudian setelah mendapatkan nilai dari modul-modul tersebut, maka nilai akan ditotal menjadi 704 komentar dengan 756 *executable statements*. Kemudian dapat dilakukan perhitungan sebagai berikut.

$$1 - \frac{704}{756} = 0{,}07$$

h. Nama untuk penggunaan variable

Pengujian yang dilakukan untuk aturan ini adalah dengan melihat apakah variabel yang digunakan bersifat unik dan tidak digunakan dalam module lain. Nilai yang didapatkan pada perhitungan ini adalah 1 karena penggunaan variabel telah sesuai dengan aturan.

i. Penggunaan variabel tunggal

Pengujian yang dilakukan untuk aturan ini adalah melihat variabel yang digunakan, jika variabel yang digunakan memiliki satu tujuan maka akan bernilai 1 dan 0 jika memiliki berbagai tujuan. Berdasarkan pengujian dari semua modul, diketahui bahwa variabel pada seluruh modul memiliki satu tujuan penggunaan. Maka dari itu penilaian pada aturan ini bernilai 1.

j. Tidak ada mode ekspresi campuran

Pengujian yang dilakukan untuk aturan ini adalah dengan melihat adanya ekspresi campuran yang digunakan. Aturan ini melihat penggunaan data pada sebuah variabel, jika

terdapat data *float* yang diproses dengan data *integer*. Berdasarkan pengujian tidak didapatkan mode ekspresi campuran, sehingga penilaian pada aturan ini bernilai 1.

k. *Nesting Level*

Pengujian ini dilakukan dengan melihat banyaknya *loop* yang digunakan dalam kode program. Aturan ini tidak didapatkan kode program yang menggunakan *nesting level*. Maka dari itu, penilaian yang didapatkan pada aturan ini adalah 0.

l. Jumlah cabang

Pengujian ini mencari banyaknya jumlah cabang yang ada pada kode program. kode cabang yang digunakan contohnya adalah if. selanjutnya mencari *executable statement* dari percabangan tersebut. Berdasarkan hasil pengujian diketahui bahwa modul *admin* memiliki 2 cabang dan 2 *executable statement*. Selanjutnya modul *auth* memiliki 4 cabang dan 4 *executable statements*. Kemudian, modul *controller* memiliki 20 cabang dan 46 *executable statements*. Selanjutnya pada modul *middleware* memiliki 3 cabang dan 3 *executable statements*. Berdasarkan pengujian yang dilakukan maka didapatkan perhitungan menggunakan rumus Persamaan (4.15) dan didapatkan hasil 0,47 sebagai berikut.

$$1 - \frac{\#\ Branches}{\#executable\ statements} \tag{4.15}$$

$$1 - \frac{29}{55} = 0{,}47$$

m. Jumlah GOTO

Pengujian ini mencari penggunaan GOTO pada kode program. Namun pada kode program tidak ditemukan fungsi GOTO, sehingga pengujian ini memiliki hasil nilai 1.

n. Tidak ada bahasa asing yang digunakan pada kode

Pengujian ini akan melihat kode program dan mencari kode yang tidak berfungsi. Dalam kode program terdapat 3 modul yang memiliki kode yang tidak berfungsi yaitu pada modul *admin*, modul *controller*, modul *middleware*. Modul *admin* memiliki fungsi yang tidak memiliki kode didalamnya, selanjutnya pada modul *controller* juga terdapat fungsi yang tidak memiliki kode. Setelah itu, terdapat modul *middleware* terdapat variabel yang tidak berfungsi, sehingga ketiga modul tersebut akan bernilai 0, dan modul *auth, app, http, migrations,* dan *seeds* akan bernilai 1.

o. Campuran variabel dalam modul

Aturan ini akan melihat variabel lokal dan global yang ada pada kode program, perhitungan yang dilakukan pada aturan ini menggunakan rumus Persamaan (4.16) sebagai berikut.

$$\frac{\#\ Internal\ Variables}{Total\ \#\ Variables} \tag{4.16}$$

Namun pada aturan ini setiap modul akan bernilai 1 dikarenakan kode program yang digunakan pada kode program tidak memiliki variabel global.

p. Kepadatan Variabel

Aturan ini akan mencari variabel dan *executable statements*, dari hasil pengujian didapatkan bahwa modul *admin* terdapat 5 variabel dan 9 *executable statements.* Selanjutnya modul *auth* terdapat 15 variabel dan 23 *executable statements*, untuk modul *controller* memiliki 307 variabel dan 362 *executable statements.* Kemudian, modul *middleware* memiliki 6 variabel dan 7 *executable statements.* Selanjutnya modul *HTTP* memiliki 4 variabel dan 5 *executable statements.* Selanjutnya pada modul *app* memiliki 28 variabel dan 30 *executable statements.* Modul *migrations* tidak memiliki variabel tetapi memiliki 110 *executable statements.* Setelah itu modul *seeds*, memiliki 12 variabel dan 10 *executable statements.* Berdasarkan modul yang telah diketahui tersebut dapat dilakukan perhitungan menngunakan rumus Persamaan (4.17) sebagai berikut.

$$1 - (\frac{\#\ Variables}{\#\ Executable\ Statements}) \tag{4.17}$$

$$1 - \frac{376}{555} = 0{,}32$$

Dari nilai yang telah didapatkan untuk aturan-aturan, maka nilai-nilai tersebut akan di rata-rata setiap aturan dari aturan *measure of coding simplicity* menggunakan rumus Persamaan (4.4) seperti berikut.

$$\frac{1 + 1 + 1 + 1 + 1 + 0 + 0{,}07 + 1 + 1 + 1 + 0 + 0{,}47 + 1 + 0 + 1 + 0 + 0 + 1 + 1 + 1 + 1 + 1 + 0{,}32}{16}$$

$$= 0{,}99$$

Kemudian nilai yang didapatkan tersebut akan dibagi dengan modul yang digunakan pada fase implementasi yang berjumlah 8, maka perhitungannya sebagai berikut.

$$\frac{0{,}99}{8} = 0{,}12$$

4.2.5 *Consistency*

Karakteristik ini akan memiliki dua aturan yang diuji melalui fase desain dan fase implementasi. Penilaian aturan-aturan pada karakteristik ini dapat dilihat sebagai berikut.

Procedure consistency measure

Aturan ini diuji dengan melihat fase desain dan implementasi, aturan ini terdapat empat aturan yang harus diukur. Aturan tersebut dapat dilihat sebagai berikut.

a. *Standard Design Representation*

Aturan ini akan melihat fase desain dari dokumentasi, apakah modul memiliki *flowchart* dan *use case diagram* atau tidak. Perhitungan yang dilakukan menggunakan rumus Persamaan (4.5), dari dokumentasi semua modul telah memiliki *flowchart* dan didapatkan hasil nilai 1.

$$1 - \frac{0}{3} = 1$$

b. *Calling sequence conventions*

Aturan ini menghitung dengan melihat fase desain dan implementasi, fase desain melihat adanya standar modul yang ada pada dokumentasi. Dalam fase desain modul *admin* tidak terdapat *flowchart* yang dapat menjadi standar acuan. Hanya modul *admin* saja yang tidak mengikuti aturan ini, sehingga perhitungannya sebagai berikut.

$$1 - \frac{1}{3} = 0{,}67$$

Selanjutnya untuk fase implementasi, aturan ini akan menghitung berdasarkan fase desain. Dalam fase ini hanya modul *admin* saja yang tidak mengikuti aturan, dikarenakan hanya modul *admin* yang tidak memiliki standar acuan dan perhitungannya sebagai berikut.

$$1 - \frac{1}{3} = 0{,}67$$

c. *Input/Output Conventions*

Aturan ini mengatur *input* dan *output* dalam kode program. *I/O* tersebut dijelaskan dalam fase desain dan diikuti dalam fase implementasi. Namun, dari fase desain ini tidak ada dokumen yang menjelaskan I/O tersebut. Sehingga penilaian untuk fase desain dan implementasi adalah *Not Available*.

d. *Error Handling Conventions*

Aturan ini menghitung dengan melihat fase desain dari dokumentasi. Fase desain melihat dokumentasi terkait metode yang digunakan jika terjadi *error* yang akan diikuti pada saat implementasi. Namun, dari pengujian ini tidak ditemukan cara mengatasi *error* pada dokumentasi sehingga fase desain dan implementasi akan bernilai *Not Available.* Kemudian nilai yang sudah didapatkan tersebut akan dirata rata pada fase desain sebagai berikut.

$$\frac{1 + 0{,}67}{2} = 0{,}84$$

Selanjutnya fase implementasi dirata-rata dan mendapatkan nilai 0,67 seperti berikut.

$$\frac{0{,}67}{1} = 0{,}67$$

Procedure consistency measure

Aturan ini akan diuji dengan melihat fase desain dan implementasi. Aturan ini memiliki 5 aturan yang harus diuji sebagai berikut.

a. *Standard Data Usage Representation*

Aturan ini dilihat melalui fase desain, fase desain untuk aturan ini akan melihat dokumentasi apakah terdapat standar untuk penggunaan data atau tidak. Dari fase desain hanya terdapat modul *app* saja yang memiliki informasi mengenai penggunaan data. maka dari itu perhitungan pada fase desain ini adalah sebagai berikut.

$$1 - \frac{2}{3} = 0{,}33$$

b. *Naming Convention*

Penggunaan variabel dalam desain telah disepakati dan diikuti pada fase implementasi. Dari dokumentasi hanya modul *app* saja yang terdapat variabelnya, sedangkan modul lain tidak ada. Oleh karena itu, perhitungan yang dilakukan hanya untuk modul *app* saja seperti berikut.

$$1 - \frac{2}{3} = 0{,}33$$

Kemudian untuk fase implementasi, karena hanya modul *app* saja yang terdapat variabel, maka perhitungan dapat dilakukan sebagai berikut.

$$1 - \frac{7}{8} = 0{,}125$$

c. *Unit Consistency*

Aturan ini diuji dengan melihat fase desain dan fase implementasi, dari fase desain yang dilihat dalam dokumentasi apakah satuan yang digunakan pada nilai sama semua atau tidak. Untuk fase desain didapatkan nilai *Not Available* dikarenakan tidak ada informasi yang menjelaskan aturan tersebut. Selanjutnya, untuk fase implementasi mencari dalam kode program data yang digunakan apakah sama semua atau tidak. Dari fase implementasi dapat diketahui untuk seluruh modul telah mengikuti aturan ini sehingga dapat dihitung seperti berikut.

$$1 - \frac{0}{8} = 1$$

d. *Consistent Global Definitions*

Aturan ini dihitung dengan melihat fase desain dan implementasi, fase desain mencari dari dokumentasi mengenai pengolahan data global. Namun, dari dokumentasi tidak terdapat penjelasan mengenai hal tersebut. Oleh karena itu, nilai yang didapatkan untuk fase desain bernilai *Not Available*. Selanjutnya, fase implementasi semua data telah mengikuti aturan ini dikarenakan aturan dalam penulisan kode telah sesuai dengan aturan ini, maka perhitungannya dapat dilihat sebagai berikut.

$$1 - \frac{0}{8} = 1$$

e. *Data Type Consistency*

Aturan ini dihitung dengan melihat fase desain dan implementasi, fase desain mencari dari dokumentasi mengenai pengolahan tipe data. Namun, dari dokumentasi tidak terdapat penjelasan mengenai hal tersebut. Oleh karena itu, nilai yang didapatkan untuk fase desain bernilai *Not Available*. Selanjutnya, untuk fase implementasi semua data telah mengikuti aturan ini dikarenakan aturan dalam penulisan kode telah sesuai dengan aturan ini, karena penulisan tidak ada yang menggabungkan dua tipe data. Maka dari itu, perhitungannya dapat dilihat sebagai berikut.

$$1 - \frac{0}{8} = 1$$

4.2.6 *Generality*

Karakteristik ini memiliki dua aturan untuk diuji melalui fase desain dan fase implementasi sebagai berikut.

Extent to which modules are referenced by other modules

Aturan ini akan mencari modul yang sering digunakan dengan modul lainnya. Aturan ini dihitung dengan melihat fase desain dan implementasi. Berdasarkan fase desain tidak ada modul yang digunakan oleh modul lainnya, sehingga nilai yang diperoleh untuk fase desain adalah 0. Selanjutnya, untuk fase implementasi terdapat dua modul yang sering digunakan oleh modul lainnya, yaitu *app* dan *controller*. Oleh karena itu, perhitungan ini didapatkan nilai 0,25 menggunakan rumus Persamaan (4.18) seperti berikut.

$$\frac{Common\ Modules}{Total\ Modules}) \tag{4.18}$$

$$\frac{2}{8} = 0{,}25$$

Implementation for Generality Measure

Aturan ini akan dibagi menjadi lima aturan lagi yang perlu diuji untuk mendapatkan nilai pada aturan ini. Perhitungan untuk aturan tersebut dapat dilihat sebagai berikut.

a. *Input, processing, output function are not mixed in a single function*

Aturan ini akan dihitung melalui fase desain dan implementasi, ketika fase desain nilai yang diukur adalah adanya diagram alir yang memiliki *input*, *output*, dan proses pada satu alur. Dari *activity diagram* diketahui bahwa terdapat lima diagram alir dan dua diagram alir yang memiliki *input, output,* dan proses. Berdasarkan pengujian tersebut maka dapat dihitung dengan rumus Persamaan (4.5) seperti berikut.

$$1 - \frac{2}{8} = 0{,}6$$

Selanjutnya untuk fase implementasi, seluruh modul terdapat sebuah fungsi yang memiliki proses, *input*, dan *output*. Oleh karena itu, perhitungan pada fase implementasi adalah sebagai berikut.

$$1 - \frac{8}{8} = 0$$

b. *Application and machine dependent functions are not mixed in a single module*

Aturan ini melakukan pengujian hanya melalui fase implementasi. Kemudian, dengan melihat kode program jika terdapat bahasa pemrograman yang menggunakan *machine dependent* seperti bahasa *assembly*. Dari pengujian tidak ditemukan bahasa *machine dependent* karena bahasa yang digunakan adalah bahasa tingkat tinggi, sehingga semua mesin dapat menerjemahkan bahasa ini. Penilaian yang didapatkan pada nilai ini adalah *Not Available*.

c. *Processing not data volume limited*

Aturan ini melakukan pengujian melalui fase desain dan implementasi. Fase desain pada aturan ini mencari dari dokumentasi mengenai penjelasan bahwa sebuah data memiliki jumlah *input* yang terbatas atau tidak. Dari hasil pengujian didapatkan bahwa terdapat gambar yang menunjukkan bahwa data tersebut terbatas, namun tidak ada penjelasan lebih mengenai pembatasan *input* tersebut. Maka dari itu, perhitungan fase desain untuk aturan ini bernilai *Not Available*. Selanjutnya fase implementasi, terdapat empat modul yang memiliki sebuah fungsi yang membatasi inputan modul. Modul tersebut adalah *admin, auth, middleware,* dan *migrations*. Sehingga perhitungan yang dilakukan pada pengujian ini adalah sebagai berikut.

$$1 - \frac{4}{8} = 0{,}5$$

d. *Processing not data value limited*

Aturan ini melakukan pengujian melalui fase desain dan implementasi., fase desain untuk aturan ini mencari dari dokumentasi mengenai penjelasan batasan pada sebuah modul. Namun, untuk aturan ini tidak terdapat sebuah penjelasan mengenai batasan sebuah fitur. Oleh karena itu, penilaian untuk aturan ini bernilai *Not Available*. Selanjutnya, fase implementasi terdapat beberapa fungsi dan variabel yang membatasi penggunaanya seperti menggunakan perintah *protected function*, yaitu modul *auth* dan *middleware,* sehingga perhitungan untuk fase ini dapat dilihat sebagai berikut.

$$1 - \frac{2}{8} = 0{,}75$$

e. *All constants should be defined once*

Aturan ini melakukan pengujian hanya melalui fase implementasi, fase implementasi yang dicari pada aturan ini adalah mencari nilai konstanta yang didefinisikan sekali dalam kode

program, setelah itu nilai konstanta itu tidak diubah-ubah kembali untuk kemudahan pemrosesan pada modul lainnya. Dari hasil pengujian pada kode program tidak ada konstanta yang digunakan dalam kode, maka penilaian pada aturan ini bernilai *Not Available*.

Kemudian nilai tersebut akan dirata-rata untuk fase desain dan implementasi dengan rumus Persamaan (4.4). Nilai yang didapatkan untuk fase desain adalah sebagai berikut.

$$\frac{0,6}{1} = 0,6$$

Nilai yang didapatkan untuk fase implementasi adalah sebagai berikut.

$$\frac{0 + 0,5 + 0,75}{3} = 0,42$$

4.2.7 *Expandability*

Karakteristik ini memiliki dua aturan yang diuji melalui fase desain dan implementasi perhitungan aturan tersebut dapat dilihat sebagai berikut.

Data Storage Expansion Measure

Aturan ini akan memiliki dua aturan yang akan diuji melalui fase desain dan implementasi. Perhitungan yang dilakukan pada aturan ini dapat dilihat sebagai berikut.

a. *Logical processing independent of storage specification*

Aturan ini akan membahas batasan yang ada pada sebuah variabel, aturan ini diuji melalui fase desain dan implementasi. Dari hasil pengujian diketahui bahwa tidak terdapat dokumentasi yang menjelaskan bahwa adanya batasan pada sebuah modul. Oleh karena itu, penilaian untuk aturan ini bernilai *Not Available*. Selanjutnya, fase implementasi terdapat pembatasan pada sebuah variabel seperti panjang *text*. Oleh karena itu, perhitungan yang dilakukan adalah sebagai berikut.

$$1 - \frac{0}{8} = 1$$

b. *Percent of memory capacity uncommitted*

Aturan ini akan diuji melalui fase implementasi yang mana melihat persentase alokasi yang dibutuhkan *database*. Dari hasil pengujian semua modul tidak terdapat alokasi persentase alokasi yang dibutuhkan untuk *database*. Oleh karena itu, aturan ini dapat dihitung sebagai berikut.

$$1 - \frac{8}{8} = 0$$

Setelah mendapatkan nilai-nilai tersebut maka nilai tersebut akan dirata-rata berdasarkan fase desain dan implementasi. Nilai rata-rata pada fase desain berupa *Not Available*, sedangkan untuk fase implementasi dapat dihitung menggunakan rumus Persamaan (4.4). seperti berikut.

$$\frac{1 + 0}{2} = 0{,}5$$

Data Storage Expansion Measure

Aturan ini akan terdiri dari tiga aturan yang akan diuji melalui fase desain dan implementasi, perhitungannya dapat dilihat sebagai berikut.

a. *Accuracy, convergence, timing attributes which control processing are parametric*

Aturan ini membahas waktu *processing* yang dilakukan sebuah aplikasi dan akurasinya ketika dijalankan, fase desain melihat dokumentasi terkait hal-hal tersebut. Dari dokumentasi tidak didapatkan penjelasan mengenai akurasi dan waktu *processing* yang didapatkan. Maka dari itu, aturan ini akan bernilai *Not Available* untuk fase desain. Fase implementasi melakukan pengujian pada aplikasi dan melihat apakah akan mempengaruhi data yang ada di *database.* Pengujian dilakukan dengan membuat pesanan dan melakukan transaksi pada pesanan tersebut. Namun, diketahui bahwa ketika transaksi sudah dilakukan data pada *database* tidak berkurang sama sekali. Oleh karena itu, penilaian pada aturan ini akan bernilai 0 dengan perhitungan sebagai berikut.

$$1 - \frac{8}{8} = 0$$

b. *Modules table driven*

Aturan ini menguji apakah sebuah data pada *database* dapat diekspor menjadi data lain seperti *pdf* atau *excel*. Aturan ini akan diuji melalui fase desain dan implementasi, fase desain diketahui bahwa tidak ada penjelasan mengenai ekspor data tersebut. Maka perhitungan ini tidak dapat dilakukan dan bernilai *Not Available*. Selanjutnya, fase implementasi diketahui bahwa berdasarkan *database* sistem dapat melakukan ekspor data. Maka dari iu, perhitungan yang dilakukan pada fase ini adalah sebagai berikut.

$$1 - \frac{0}{8} = 1$$

c. *Percent of speed capacity uncommitted*

Aturan ini dihitung hanya melalui fase implementasi dengan melihat apakah *developer* telah melakukan pengujian kecepatan pada sebuah aplikasi. Namun, untuk aturan ini *developer* belum melakukan pengujian tersebut, sehingga penilaian bernilai *Not Available*. Setelah mendapatkan nilai-nilai tersebut maka selanjutnya akan melakukan rata-rata untuk setiap aturan dari fase desain dan implementasi. Untuk fase desain nilai rata-rata yang didapatkan adalah *Not Available*, sedangkan fase implementasi dapat dilihat sebagai berikut.

$$\frac{0 + 1}{2} = 0{,}5$$

4.2.8 *Instrumentation*

Karakteristik ini memiliki tiga aturan yang akan diuji melalui fase desain dan implementasi. Namun, ketiga aturan tersebut tidak ditemukan pada dokumentasi untuk fase desain, sehingga seluruh fase desain akan bernilai *Not Available*. Perhitungan yang akan dilakukan dapat dilihat sebagai berikut.

Module Testing Measure

Aturan ini terdapat dua aturan yang akan diuji melalui fase desain dan implementasi, perhitungan yang dilakukan dapat dilihat sebagai berikut.

a. *Path Coverage*

Aturan ini akan mencari rencana pengujian dari fase desain dan dilakukan pada fase implementasi. Namun, dalam fase desain tidak ditemukan rencana pengujian tersebut sehingga penilaian akan bernilai *Not Available*. Sedangkan bagian fase implementasi terdapat pengujian yang dilakukan melalui dua jalur untuk setiap modul, tetapi tidak dijelaskan pada fase desain sebelumnya. Maka dari itu, fase implementasi ini bernilai 1 karena semua modul diuji berdasarkan jalur yang berbeda tidak hanya satu jalur saja.

b. *Input Parameter Boundary Tested*

Aturan ini melihat melalui fase desain dan implementasi, fase desain melihat dari dokumentasi jika terdapat rencana pengujian *input* pada sebuah fungsi. Namun, tidak ada data tersebut dalam dokumentasi sehingga nilainya adalah *Not Available*. Sedangkan untuk

fase implementasi terdapat sebuah data pengujian yang dilakukan, tetapi tidak dijelaskan dalam dokumentasi.

Setelah mendapatkan nilai tersebut maka nilai itu akan dirata-rata dari fase desain dan implementasi. Namun, seluruh fase desain karena tidak ada rancangan data dari dokumentasi sehingga nilainya adalah *Not Available*. Sedangkan pada fase implementasi rata-rata dapat dihitung sebagai berikut.

$$\frac{0+1}{2} = 0{,}5$$

Integration Testing Measure

Aturan ini akan menguji *interface* dan *performance* yang direncanakan dari fase desain dan dilakukan pada fase implementasi. Namun, aturan ini bernilai *Not Available* karena tidak ada rencana pengujian serta eksekusi pengujian yang dilakukan pada aturan ini. Oleh karena itu, aturan ini bernilai *Not Available*.

System Testing Measure

Aturan ini akan menguji kualitas modul dan pengujian *input* serta *output* dalam bentuk ringkasan yang direncanakan dari fase desain dan dilakukan pada fase implementasi. Namun, aturan ini juga tidak didapatkan data mengenai perencanaan dan eksekusi pengujian tersebut. Oleh karena itu, aturan ini akan bernilai *Not Available* (*NA*) seperti diperlihatkan Tabel 4.1.

Tabel 4.1 merupakan hasil perhitungan eksekusi pengujian dari aturan-aturan setiap karakteristik. Nilai-nilai tersebut dihitung berdasarkan rumus metrik masing-masing aturan dari setiap karakteristik. Setelah mendapatkan nilai perhitungan rata-rata menggunakan *system matrix value* langkah selanjutnya, yaitu menghitung kualitas faktor menggunakan Persamaan (2. 1).

Tabel 4.1 Hasil Nilai Matrik

Karakteristik	Metric	Bobot	Design		Implementasi	
			1/0	Value	1/0	Value
Conciseness	1. Halstead Measure $1 - \frac{\lvert calculated - observed \rvert}{Module}$	0.125	-	-	-	0.49, 0.53, 0.26, 0.64, 0.63, 0.50, 0.23, 0.54

	System Matrix Value $\frac{Sum\ Halstead\ measure\ each\ module}{Module}$		-	-	-	0.48
Self Descriptive	1. Quantitiy Of Comments $\frac{\#\ Of\ Comments}{Total\ \#\ lines}$	0.125	-	-	-	0.27, 0.45, 0.32, 0.33, 0.27, 0.11, 0.27, 0.24
	System Matrix Value $\frac{Sum\ Quantity\ of\ Comments}{Total\ \#\ modules}$		-	-	-	0.28
	2. Effectiveness of Comments Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$		-	-	-	0, 1, 0, 0.13, 0
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	-	-	0.23
	3. Descriptiveness Of Implementation Language $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$		-	-	-	1, 0.75, 0.75, 1, 0.84, 1
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	-	-	0.89
Modularity	1. Stability Measure $\frac{Expected\ \#\ modules}{Total\ \#\ modules}$	0.125	-	NA	-	-
	System Matrix Value $\frac{Expected\ \#\ modules}{Total\ \#\ modules}$		-	NA	-	-
	2. Modular Implementation Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$		-	NA	-	1, 0.88, 1, 0, 1,

						1, 1,
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	NA	-	0.74
Simplicity	1. Design Structure Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$	0.125	-	1, 1, 0.33, 0, 0.67, 0.67, NA	-	1 0, 0, 0, 1 0
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	0.61	-	0.33
	2. Use Of Structured Language Or Preprocessor 1 jika menggunakan dan 0 jika tidak		-	-	1, 1, 1, 1, 0, 0, 0, 0	-
	System Matrix Value $\frac{Sum\ Of\ Modules\ Scores}{\#\ modules\ scores}$		-	-	-	0.5
	3. Complexity Measure		-	0.2, 0.8	-	0.4, 0.2, 0.14, 0.17, 0, 0.11, 0, 0.25
	System Matrix Value $\frac{Total\ Scores\ complexity\ measure}{Total\ \#\ modules}$		-	0.5	-	0.16
	4. Measure Of Coding Simplicity $1 - \frac{\#\ of\ elements}{executable\ statements}$		-	-	1, 1,	 1 1, 1,

					0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,	0.07, 0, 0.47, 1, 1, 0.32
	Module Matrix value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	-	-	0.99
	System Matrix Value $\frac{Scores\ Coding\ Simplicity\ measure}{\#\ modules}$		-	-	-	0.12
Consistency	1. Procedure Consistency Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$	0.125	-	1, 0.67 NA, NA	-	0.67 NA, NA.
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	0.84	-	0.67

	2. Data Consistency Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$		-	0.33, 0.33, NA, NA, NA	-	0.125 1, 1, 1
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	0.33	-	0.78
Generality	1. Extent To Which Module Is Referenced By Other Modules $\frac{\#\ Common\ Modules}{Total\ \#\ modules}$	0.125	-	0	-	0.25
	System Matrix Value $\frac{\#Common\ Modules}{Total\ \#\ modules}$		-	0	-	0.25
	2. Implementation For Generality $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$		-	0.6, NA, NA	-	0, NA, 0.5, 0.75, NA
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	0.6	-	0.42
Expandability	1. Data Storage Expansion Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$	0.125	-	NA	-	1, 0
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	NA	-	0.5
	2. Extensibility Measure $1 - \frac{\#modules\ violate\ rule}{Total\#modules}$		-	NA, NA	-	0, 1, NA
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		-	NA	-	0.5
Instrumentation	1. Module Testing Measure $\frac{Path\ to\ be\ tested}{Total\ \#\ Path}$	0.125	NA	NA	-	1, 0

	Modul Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		NA	NA	-	0.5
	System Matrix Value $\frac{Total\ Scores\ modules\ testing\ measu}{Total\ \#\ modules}$		NA	NA	-	0.06
	2. Integration Testing Measure $\frac{\#\ to\ be\ tested}{Total\ \#\ interface}$		NA	NA	-	NA
	System Matrix Value $\frac{Total\ Scores\ applicable\ elements}{Total\ \#\ modules}$		NA	NA	-	NA
	3. System Testing Measure $\frac{\#modules\ to\ be\ executed}{Total\ \#\ of\ modules}$		NA	NA	NA	NA
	System Matrix Value $\frac{Total\ Scores\ modules\ testing\ measu}{Total\ \#\ modules}$		NA	NA	-	NA

4.3 Penutupan Siklus Uji (*Test Cycle Closure)*

Tabel 4.1 Hasil Nilai Matrik merupakan nilai matrik yang didapatkan dari pengujian setiap karakteristik, setelah mendapatkan perhitungan matrik tersebut langkah selanjutnya adalah menghitung kualitas faktor menggunakan Persamaan (2. 1) menggunakan nilai rata-rata yang didapatkan dari setiap karakteristik.

a. Hasil Analisis Faktor Kualitas Implementasi *Maintainability*

$$\text{Conciseness} = 0.125 \,.\, 0.48 = 0.06$$

$$\text{Self Descriptive} = 0.125 \,.\, 0.28 + 0.125 \,.\, 0.23 + 0.125 \,.\, 0.89 = 0.18$$

$$\text{Modularity} = 0.125 \,.\, 1 + 0.125 \,.\, 0.88 + 0.125 \,.\, 1 + 0.125 \,.\, 0 + 0.125 \,.\, 1 + 0.125 \,.\, 1 + 0.125 \,.\, 1 + 0.125 \,.\, 0 = 0.74$$

$$\text{Simplicity} = 0.125 \,.\, 0.33 + 0.125 \,.\, 0.5 + 0.125 \,.\, 0.16 + 0.125 \,.\, 0.12 = 0.14$$

$$\text{Consistency} = 0.125 \,.\, 0.67 + 0.125 \,.\, 0.78 = 0.18$$

$$\text{Fq} = 0.06 + 0.175 + 0.74 + 0.14 + 0.18 = 1.295$$

Setelah didapatkan nilai faktor kualitas dari nilai *maintainability* bernilai 1.295. Nilai tersebut akan diubah menggunakan persentase seperti Persamaan (2. 2).

$$\text{Rerata Faktor Software Quality} = \frac{1.295}{5} = 0.26$$

Nilai maksimum pada satu faktor adalah 1, sehingga perhitungan dapat dilakukan sebagai berikut.

$$\% = \frac{0.26}{1} . 100\% = 26\%$$

Dari perhitungan di atas, dapat diketahui bahwa aplikasi Ivent berdasarkan faktor *maintainability* pada fase implementasi memiliki kualitas bernilai 26%.

b. Hasil Analisis Faktor Kualitas Implementasi *Flexibility*

$$\text{Self Descriptive} = 0.125 . 0.28 + 0.125 . 0.23 + 0.125 . 0.89 = 0.18$$

$$\text{Generality} = 0.125 . 0.25 + 0.125 . 0.42 = 0.08$$

$$\text{Expandability} = 0.125 . 0.5 + 0.125 . 0.5 = 0.13$$

$$\text{Fq} = 0.18 + 0.08 + 0.13 = 0.39$$

$$\text{Rerata Faktor Software Quality} = \frac{0.39}{3} = 0.13$$

$$\% = \frac{0.13}{1} . 100\% = 13\%$$

Dari perhitungan di atas, dapat diketahui bahwa aplikasi Ivent ini berdasarkan faktor *flexibility* pada fase implementasi memiliki kualitas sebesar 13%.

c. Hasil Analisis Faktor Kualitas Implementasi *Testability*

$$\text{Self Descriptive} = 0.125 . 0.28 + 0.125 . 0.23 + 0.125 . 0.89 = 0.18$$

$$\text{Modularity} = 0.125 . 1 + 0.125 . 0.88 + 0.125 . 1 + 0.125 . 0 + 0.125 . 1 + 0.125 . 1 + 0.125 . 1 + 0.125 . 0 = 0.74$$

$$\text{Simplicity} = 0.125 . 0.33 + 0.125 . 0.5 + 0.125 . 0.16 + 0.125 . 0.12 = 0.14$$

$$\text{Instrumentation} = 0.125 * 0.06 = 0.01$$

$$\text{Fq} = 0.18 + 0.74 + 0.14 + 0.01 = 1.07$$

$$\text{Rerata Faktor Software Quality} = \frac{1.07}{4} = 0.27$$

$$\% = \frac{0.27}{1} . 100\% = 27\%$$

Setelah mengetahui kualitas faktor pada fase implementasi, maka selanjutnya mencari faktor kualitas pada fase desain sebagai berikut.

a. Hasil Analisis Faktor Kualitas Desain *Maintainability*

$$\text{Simplicity} = 0.125 . 0.61 + 0.125 . 0.5 = 0.14$$

$$\text{Consistency} = 0.125 . 0.84 + 0.125 . 0.33 = 0.15$$

$$\text{Fq} = 0.14 + 0.15 = 0.29$$

$$\text{Rerata Faktor Software Quality } = \frac{0.29}{2} = 0.15$$

$$\% = \frac{0.14}{1} . 100\% = 15\%$$

b. Hasil Analisis Faktor Kualitas Desain *Flexibility*

$$\text{Generality } = 0.125 . 0 + 0.125 . 0.6 = 0.075$$

$$\% = \frac{0.075}{1} . 100\% = 7.5\%$$

c. Hasil Analisis Faktor Kualitas Desain *Testability*

$$\text{Simplicity } = 0.125 . 0.61 + 0.125 . 0.5 = 0.14$$

$$\% = \frac{0.14}{1} . 100\% = 14\%$$

Setelah melakukan perhitungan seperti yang ada diatas, maka didapatkan tabel dibawah ini.

Berdasarkan dari perhitungan yang telah dilakukan dengan menggunakan rumus *Factor Software Quality*, didapatkan nilai akhir seperti pada Tabel 4. 2 Nilai Akhir Karakteristik.

Tabel 4. 2 Nilai Akhir Karakteristik

Faktor / Karakteristik	Maintainability		Flexibility		Testability	
	Desain	**Implementasi**	**Desain**	**Implementasi**	**Desain**	**Implementasi**
Conciseness		0.06				
Self Descriptive		0.18		0.18		0.18
Modularity	NA	0.74			NA	0.74
Simplicity	0.14	0.14			0.14	0.14
Consistency	0.15	0.18				
Generality			0.075	0.08		
Expandability			NA	0.13		
Instrumentation					NA	0.01
Rata-Rata	**15%**	**26%**	**7.5%**	**13%**	**14%**	**27%**
	20.5%		**10.3%**		**20.5%**	

Dari Tabel 3. 3 Rentang Kategori Kualitas yang menjelaskan kategori dari kualitas yang didapatkan masing-masing karakteristik, aplikasi web Ivent masuk dalam kategori sangat tidak baik. Diketahui faktor *Maintainability* memiliki nilai akhir sebesar 20.5% nilai tersebut termasuk dalam kategori sangat tidak baik. Hal ini dikarenakan karakteristik *conciseness* dan *simplicity* memiliki nilai yang kecil, karakteristik *conciseness* memiliki nilai yang kecil disebabkan oleh penulisan kode yang digunakan dalam pembuatan aplikasi web Ivent terbilang tidak singkat. Karakteristik *simplicity* memiliki nilai yang kecil karena kode yang digunakan

terbilang cukup kompleks, sehingga banyak aturan yang dilanggar pada karakteristik ini khususnya pada aturan *complexity measure* terdapat banyak modul yang memiliki variabel yang sering digunakan dalam sebuah kode maupun dokumentasi sehingga melanggar aturan ini. Faktor *Flexibility* memiliki nilai akhir sebesar 10.3% nilai tersebut termasuk dalam kategori sangat tidak baik. Hal ini dikarenakan karakteristik *generality*, karakteristik *generality* memiliki nilai yang rendah khususnya dalam aturan *extent to which modules are referenced by other modules* dalam kode program hanya memiliki dua modul yang sering digunakan oleh modul lainnya. Oleh karena itu, penggunaan modul lainnya dalam karakteristik ini dinilai tidak *general*, dalam dokumentasi tidak dijelaskan penggunaan modul, sehingga tidak dapat dilakukan pengujian dalam fase desain. Faktor *Testability* memiliki nilai akhir sebesar 20.5% nilai tersebut termasuk dalam kategori sangat tidak baik. Hal ini dikarenakan karakteristik *instrumentation*, karakteristik *instrumentation* memiliki nilai yang rendah khususnya pada aturan *module testing measure* dikarenakan tidak adanya penjelasan pengujian yang digunakan pada dokumentasi dan tidak adanya pengujian yang dilakukan mengenai rentang *input* pada sebuah modul.

Dari penjelasan nilai setiap faktor sebelumnya, untuk fase desain dokumentasi yang dibuat oleh *developer* aplikasi web Ivent kurang lengkap. Beberapa karakteristik melihat dokumnetasi desain dalam melakukan perhitungan, namun banyak aturan yang tidak ditemukan dalam dokumentasi seperti karakteristik *Modularity, Expandability,* dan *Instrumentation* yang bernilai *Not Available* (NA). Fase implementasi dilihat dari penulisan kode oleh *developer,* namun banyak aturan dalam karakteristik yang tidak terpenuhi dari penulisan kode. Contohnya seperti karakteristik *Self Descriptive* yang mengatur penulisan komentar pada kode, sebuah komentar harus menjelaskan isi dari kode yang ditulis agar pembaca dapat mengerti fungsi dan tujuan dari kode. Tetapi, banyak ditemukan komentar yang tidak menjelaskan isi dari kode yang ditulis, bahkan ditemukan kode tanpa penulisan komentar. Oleh karena itu, aplikasi web Ivent masih tidak mampu untuk menghadapi perubahan berdasarkan nilai yang didapatkan dari pengujian nonfungsional pada perspektif *product revision* dengan menggunakan metode pengujian *McCall's Factor*.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian ini difokuskan pada pengujian kualitas perangkat lunak nonfungsional. Berdasarkan hasil penelitian, pengujian perangkat lunak Ivent dengan menggunakan pendekatan *McCall's Factor* pada aspek *product revision* melalui tahapan awal dengan menentukan jenis tes yang dibutuhkan. Pengujian dilanjutkan dengan menentukan metrik pengujian dan perumusan pada setiap karakteristik yang telah ditentukan. Kemudian, penelitian melakukan pengaturan lingkungan uji untuk menentukan syarat *software* telah benar-benar siap untuk diuji yang dilanjutkan dengan melakukan eksekusi pengujian.

Dari perhitungan implementasi, didapatkan nilai *Software Quality* untuk faktor *Maintainability* sebesar 26%, faktor *Flexibility* sebesar 13%, dan *Testability* sebesar 14%. Sedangkan nilai dari hasil pengujian perangkat lunak pada fase desain untuk faktor *Maintainability* sebesar 15%, faktor *Flexibility* sebesar 7.5%, dan faktor terakhir yaitu *Testability* sebesar 14%. Jika nilai fase desain dan implementasi digabung, maka faktor *maintainability* mendapatkan nilai sebesar 20.5%, *flexibility* sebesar 10.3%, dan *testability* sebesar 20.5%. Perolehan nilai *Software Quality* tersebut dapat menjadi acuan bagi *developer* dalam melakukan perbaikan dan pengembangan kualitas perangkat lunak. Perbaikan khususnya pada faktor kualitas *flexibility* yang hanya mendapatkan faktor kualitas sebesar 10.3%. Perbaikan yang perlu diperbaiki terdapat dalam karakteristik *expandability* yaitu aturan batasan pada sebuah data lebih baik dijelaskan di dokumentasi, dan penggunaan data serta persentase alokasi yang dibutuhkan *database* agar dijelaskan di dalam dokumentasi. Selain itu dalam dokumentasi sebaiknya diberikan penjelasan terkait akurasi yang tertampil di aplikasi dan yang ada pada *database*, apakah selaras atau tidak. Kemudian untuk faktor kualitas *testability*, aplikasi Ivent ini masih kurang di bagian karakteristik *instrumentation* dalam menguji sistem terkait *input* dan *output* dari sebuah modul. Kemudian aturan pengujian *interface* yang dilakukan tidak ditulis dalam dokumentasi.

5.2 Saran

Penelitian ini masih memiliki kekurangan, karena penelitian ini hanya berfokus pada faktor nonfungsional aspek *product revision*. Penelitian selanjutnya dapat diteruskan dengan menguji pada aspek *product transition* maupun *product operations* menggunakan metode

McCall's Factor, dan diharapkan dapat membuat *tools* untuk melakukan pengujian nonfungsional menggunakan metode *McCall's Factor* secara otomatis.

DAFTAR PUSTAKA

Aikal, M. (2021). *Pengembangan E-commerce vendor dan Event Organizer Berbasis Website dengan Metode Waterfall*. https://dspace.uii.ac.id/handle/123456789/34251

Auliyaa, T. (2020). *Software Testing Life Cycle*. https://sis.binus.ac.id/2020/07/06/software-testing-life-cycle/#:~:text=Requirement%20Analysis&text=Aktivitas%20%E2%80%93%20aktivitas%20yang%20dilakukan%20berupa,hingga%20melakukan%20identifikasi%20lingkungan%20pengujian.

Chung, L., & do Prado Leite, J. C. S. (2009). On Non-Functional Requirements in Software Engineering. In A. T. Borgida, V. K. Chaudhri, P. Giorgini, & E. S. Yu (Eds.), *Conceptual Modeling: Foundations and Applications* (Vol. 5600, pp. 363–379). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-02463-4_19

Hamilton, T. (2020). *What is Non Functional Testing? (Types)*. https://www.guru99.com/non-functional-testing.html

Khasanah, F. N. (2018). Pengujian Fungsional Dan Non Fungsional Aplikasi Informasi Telepon Darurat Berbasis Android. *INFORMATION SYSTEM FOR EDUCATORS AND PROFESSIONALS*, *3*(1), 79–90.

McCall, J., Richards, P., & Walters, G. (1977). Factors in Software Quality. *Tehnical Report RADC-TR-77-369*, *1*.

Mona, J. (2022). *Software Testability (Its Benefits, Limitations, and Facilitation)*. *445*, 287–298. https://doi.org/https://doi.org/10.1007/978-981-19-1412-6_23

Musa, K., & Alkhateeb, J. (2013). Quality Model Based on Cots Quality Attributes. *International Journal of Software Engineering & Applications*, *4*(1), 1–8. https://doi.org/10.5121/ijsea.2013.4101

Naik, K., & Tripathy, P. (2011). *Software Testing and Quality Assurance : Theory and Practice*. https://books.google.co.id/books?id=neWaoJKSkvgC

Nelson, K. M., Nelson, H. J., & Ghods, M. (1997). Technology flexibility: Conceptualization, validation, and measurement. *Proceedings of the Hawaii International Conference on System Sciences*, *3*, 76–87. https://doi.org/10.1109/HICSS.1997.661572

Pohan, A. B., & Kom, M. (2018). *Pengujian dan Implementasi Sistem*. https://scholar.google.co.id/citations?view_op=view_citation&hl=id&user=Peq20YgAAAAJ&citation_for_view=Peq20YgAAAAJ:8k81kl-MbHgC

Rizky, D. (2019). *Mengenal STLC — Software Testing Life Cycle*. https://medium.com/dot-intern/mengenal-stlc-software-testing-life-cycle-d1bc5a938b72

Sharma, V., & Baliyan, P. (2011). Maintainability Analysis of Component Based Systems. *International Journal of Software Engineering and Its Applications*, *5*(3).

Siregar, M. U., & Arif, A. H. (2018). A Usage of McCall's Software Quality Analysis on the Bonus System of PT Surya Pratama Alam. *JISKA (Jurnal Informatika Sunan Kalijaga)*, *3*(1), 63. https://doi.org/10.14421/jiska.2018.31-07

Tanoto, U. (2020). Activity Diagram: Pengertian, Fungsi, Contoh serta Cara Membuatnya. In *Jojonomic Aplikasi HRIS, Human Capital & Expense Management*. https://www.jojonomic.com/blog/activity-diagram/

Weerasinghe, T. (2022). Software Quality Factors. In *Medium*. https://thilinaweerasinghe.medium.com/software-quality-factors-c0f57bbfc3d2