



REFACTORING UNTUK MENINGKATKAN KUALITAS *MAINTAINABILITY* PERANGKAT LUNAK PADA SISTEM INFORMASI KEUANGAN

Ariq Miftah Fauzan^{1*}, Puspita Nurul Sabrina², Herdi Ashaury³

^{1,2,3} Program Studi Informatika, Fakultas Sains dan Informatika,
Universitas Jenderal Achmad Yani

E-mail: ariqfauzanmiftah@gmail.com*, puspita.sabrina@lecture.unjani.ac.id,
herdi.ashaury@lecture.unjani.ac.id

ABSTRACT

This study aimed to apply refactoring to improve software quality and maintainability in the Financial Information System (SIK). The research focused on identifying code smells that could affect code readability, maintenance, and software development. The research method consisted of identifying code smells, applying appropriate refactoring techniques, conducting Black-Box Testing, and measuring software metrics before and after refactoring. The results identified six types of code smells, namely Long Method, Duplicate Code, Lack of Validation, Hardcoded Logic, Data Clumps, and Incomplete Abstraction. The refactoring techniques applied included Extract Method, Introduce Parameter Validation, Group Data in Variable, Introduce Missing Methods or Properties, and Query Abstraction. Black-Box Testing was conducted on 22 test cases, with all cases producing results as expected, resulting in a 100% success rate. The Maintainability Index increased after refactoring, indicating improved code maintainability. Therefore, refactoring was effective in improving the structure, readability, and maintainability of the SIK.

Keywords: *refactoring, code smells, Maintainability Index, software metrics, Financial Information System*

ABSTRAK

Penelitian ini bertujuan menerapkan *refactoring* untuk meningkatkan kualitas dan *maintainability* pada Sistem Informasi Keuangan (SIK). Permasalahan penelitian berfokus pada keberadaan *code smells* yang dapat memengaruhi keterbacaan, pemeliharaan, dan pengembangan perangkat lunak. Metode penelitian dilakukan melalui identifikasi *code smells*, penerapan teknik *refactoring*, pengujian menggunakan *Black-Box Testing*, serta pengukuran *software metrics* sebelum dan sesudah *refactoring*. Hasil identifikasi menunjukkan enam jenis *code smells*, yaitu *Long Method*, *Duplicate Code*, *Lack of Validation*, *Hardcoded Logic*, *Data Clumps*, dan *Incomplete Abstraction*. Teknik *refactoring* yang diterapkan meliputi *Extract Method*, *Introduce Parameter Validation*, *Group Data in Variable*, *Introduce Missing Methods or Properties*, dan

Article History

Received: September 2026

Reviewed: September 2026

Published: September 2026

Plagiarism Checker No
237.u873247.fh8374

Copyright : Author
Publish by : Kohesi



This work is licensed under
a [Creative Commons
Attribution-NonCommercial
4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/)



abstraksi *Query*. Hasil *Black-Box Testing* terhadap 22 kasus pengujian menunjukkan seluruh fungsi berjalan sesuai dengan yang diharapkan dengan tingkat keberhasilan 100%. Pengukuran *Maintainability Index* menunjukkan peningkatan setelah *refactoring*, sehingga penerapan *refactoring* dapat meningkatkan keterpeliharaan dan kualitas struktur kode pada SIK.

Kata kunci: *refactoring*, *code smells*, *Maintainability Index*, *software metrics*, Sistem Informasi Keuangan.

1. PENDAHULUAN

Sistem Informasi Keuangan (SIK) memiliki peran penting dalam mendukung pengelolaan keuangan organisasi, mulai dari pendapatan, pengeluaran, pencatatan transaksi, hingga penyusunan laporan keuangan. Kualitas perangkat lunak menjadi aspek penting agar sistem dapat digunakan secara efektif dan berkelanjutan. Salah satu pendekatan untuk meningkatkan kualitas struktur kode adalah *refactoring*, yaitu proses perubahan struktur internal kode sumber tanpa mengubah perilaku eksternal perangkat lunak. Penerapan *refactoring* dapat menghasilkan kode yang lebih mudah dibaca, dipahami, dipelihara, dan dikembangkan. Almogahed dan Omar (2021) menunjukkan bahwa pemilihan teknik *refactoring* disesuaikan dengan kondisi kode dan kebutuhan pemeliharaan, sedangkan Alsarraj dan Altaie (2021) menjelaskan bahwa penerapan *refactoring* dapat meningkatkan kualitas perangkat lunak, termasuk kemudahan pemeliharaan dan pemahaman kode.

Permasalahan struktur kode dapat diketahui melalui *code smells*, yaitu indikasi kelemahan dalam kode yang dapat meningkatkan kesulitan pemeliharaan dan pengembangan. Agnihotri dan Chug (2020) menjelaskan keterkaitan *code smells*, *software metrics*, dan teknik *refactoring* dalam peningkatan kualitas kode, sementara Kusuma (2024) menunjukkan bahwa deteksi *code smells* secara sistematis dapat membantu menemukan permasalahan struktur kode dan menentukan tindakan perbaikan. Salah satu jenis yang umum ditemukan adalah *long method*, yaitu metode dengan ukuran atau kompleksitas yang terlalu besar, yang dapat ditangani menggunakan *Extract Method*. Shahidi et al. (2022) menunjukkan bahwa teknik tersebut dapat digunakan untuk memperbaiki *long method*. Selain itu, *duplicate code* berupa pengulangan kode atau logika dapat meningkatkan beban pemeliharaan. Sasaki et al. menunjukkan pentingnya identifikasi duplikasi kode melalui studi empiris menggunakan berbagai metode investigasi dan alat deteksi.

Selain *long method* dan *duplicate code*, permasalahan struktur kode dapat berupa *data clumps*, *lack of validation*, *hardcoded logic*, dan *incomplete abstraction*. *Data clumps* terjadi ketika kelompok data yang sama muncul berulang dan digunakan secara bersamaan pada beberapa bagian kode. Baumgartner et al. (2023) menunjukkan bahwa deteksi *data clumps* dapat membantu pengembang menemukan bagian kode yang memerlukan perbaikan. Sementara itu, *lack of validation* berkaitan dengan tidak optimalnya validasi masukan pengguna yang dapat memengaruhi ketepatan dan keandalan sistem. Berbagai *code smells* tersebut perlu ditangani agar kualitas kode tetap terjaga. Dalam evaluasi kualitas perangkat lunak, model McCall dapat digunakan untuk menilai berbagai faktor kualitas, termasuk *maintainability*. Dirjen et al. (2017) dan Syaifullah et al. (2022) menunjukkan penerapan model McCall sebagai pendekatan untuk mengevaluasi kualitas sistem informasi.

Penerapan *refactoring* telah digunakan sebagai strategi untuk meningkatkan kualitas dan pemeliharaan perangkat lunak. Fredianto (2021) menunjukkan bahwa *refactoring* dapat digunakan untuk mengoptimalkan perangkat lunak melalui perbaikan struktur kode, sedangkan



Andriani et al. menunjukkan penerapan metode *refactoring* dalam meningkatkan kualitas *maintainability* pada sistem informasi. Selain itu, Sa et al. (2022) mengembangkan alat *refactoring* otomatis untuk menangani *code smell* tertentu dengan memanfaatkan pendekatan *software metrics*. Temuan tersebut menunjukkan bahwa *refactoring* dapat menjadi strategi dalam meningkatkan kualitas kode selama proses pemeliharaan perangkat lunak. Dalam SIK, kebutuhan pemeliharaan menjadi semakin penting seiring bertambahnya data, perubahan kebutuhan pengguna, dan pengembangan sistem. Oleh karena itu, *code review*, identifikasi *code smells*, dan penerapan teknik *refactoring* diperlukan untuk memperbaiki struktur kode secara terarah.

Berdasarkan permasalahan tersebut, penelitian ini berfokus pada peningkatan kualitas kode sumber Sistem Informasi Keuangan melalui penerapan *refactoring* dengan fokus pada enam jenis *code smells*, yaitu *long method*, *duplicate code*, *data clumps*, *lack of validation*, *hardcoded logic*, dan *incomplete abstraction*. Teknik *refactoring* dipilih berdasarkan karakteristik permasalahan yang ditemukan dengan tetap mempertahankan fungsionalitas sistem. Penelitian ini bertujuan meningkatkan kualitas perangkat lunak pada faktor *maintainability*, mengetahui perubahan kualitas kode setelah penerapan *refactoring*, serta memastikan bahwa perubahan struktur kode tidak mengganggu fungsionalitas sistem. Dengan demikian, penelitian diharapkan menghasilkan struktur kode SIK yang lebih terorganisasi, mudah dipahami, dipelihara, dan dikembangkan.

2. METODE PENELITIAN

Penelitian ini menggunakan pendekatan pengembangan dan perbaikan perangkat lunak dengan menerapkan *refactoring* pada kode sumber Sistem Informasi Keuangan (SIK) untuk meningkatkan faktor *maintainability*. Tahapan penelitian meliputi *code review* dan identifikasi *code smells*, pengukuran *software metrics* awal menggunakan PHP Depend, perancangan dan implementasi *refactoring*, pengujian fungsional, serta pengukuran dan evaluasi setelah *refactoring*. Identifikasi dilakukan secara manual pada sembilan *class controller*, yaitu Catering, Gaji, Guru, Lainnya, Pendaftaran, Pengeluaran, Snack, Transaksi, dan Ujian, dengan fokus pada *long method*, *duplicate code*, *data clumps*, *incomplete abstraction*, *lack of input validation*, dan *hardcoded logic*. Metrik yang digunakan meliputi *Line of Code* (LOC), *Cyclomatic Complexity* (CC), *Halstead Volume*, dan *Maintainability Index* (MI) untuk membandingkan kondisi kode sebelum dan sesudah *refactoring* (Agnihotri & Chug, 2020; Syaifullah et al., 2022). Teknik *refactoring* dipilih berdasarkan jenis permasalahan, meliputi *Extract Method*, *Group Data in Variables*, *Introduce Parameter Validation*, *Introduce Missing Methods or Properties*, dan abstraksi *Query*. *Extract Method* digunakan untuk memisahkan tanggung jawab metode dan menangani *long method* serta *duplicate code*, sebagaimana diterapkan dalam penelitian Shahidi et al. (2022), sedangkan penerapan *refactoring* untuk meningkatkan *maintainability* sistem informasi juga didukung oleh Andriani et al. (n.d.). Implementasi dilakukan tanpa menambah fitur atau mengubah fungsi utama sistem, kemudian diuji menggunakan *Black-Box Testing* untuk memastikan seluruh fungsi tetap berjalan sesuai kebutuhan. Selanjutnya, nilai LOC, CC, *Halstead Volume*, dan MI diukur kembali menggunakan PHP Depend dan dianalisis secara deskriptif, dengan MI sebagai indikator utama untuk mengevaluasi perubahan *maintainability* perangkat lunak.

3. HASIL DAN PEMBAHASAN

Penerapan *Refactoring*

Hasil analisis terhadap kode sumber Sistem Informasi Keuangan (SIK) menunjukkan adanya enam jenis *code smells*, yaitu *long method*, *duplicate code*, *data clumps*, *incomplete*



abstraction, *lack of input validation*, dan *hardcoded logic*. *Code smells* tersebut ditemukan pada sembilan *class controller*, yaitu *Catering*, *Gaji*, *Guru*, *Lainnya*, *Pendaftaran*, *Pengeluaran*, *Snack*, *Transaksi*, dan *Ujian*. Penerapan *refactoring* dilakukan dengan menyesuaikan teknik perbaikan terhadap jenis permasalahan yang ditemukan. Penggunaan *refactoring* untuk memperbaiki struktur kode dan meningkatkan kualitas perangkat lunak sejalan dengan Agnihotri dan Chug (2020) serta Almogahed dan Omar (2021).

Class Catering

Pada *class Catering* ditemukan *long method* pada metode *Simpan* dan *UpdateData*. Metode *Simpan* sebelumnya menggabungkan proses perhitungan tanggal, pemeriksaan hari, validasi kondisi siswa, pembentukan data tanggal, penyimpanan data, dan keluaran JSON. Sementara itu, metode *UpdateData* menggabungkan pemeriksaan data, *query*, pembaruan data catering, pengambilan nominal, pembaruan kas keluar, dan respons JSON.

Perbaikan dilakukan menggunakan teknik *Extract Method*. Pada metode *Simpan* dibuat metode *generateTanggalCatering*, *isTanggalValid*, dan *outputJSON*. Pada metode *UpdateData* dibuat metode *isCateringExists*, *updateCateringData*, *getNominalPembayaran*, *updateKasKeluar*, dan *outputJSON*.

Tabel 1 Hasil Refactoring Class Catering (Simpan)

```
function Simpan() {
    $tanggalAwal = $this->input->post('tanggal', TRUE);
    $harga = $this->input->post('harga');
    $totalHari = $this->input->post('total', TRUE);
    $idSiswa = $this->input->post('id_siswa');
    $totalPembayaran = $this->input->post('seluruh');

    $tanggalList = $this->generateTanggalCatering($tanggalAwal,
    $totalHari, $idSiswa, $harga);

    if (!empty($tanggalList)) {
        $this->db->insert_batch('catering', $tanggalList);
        $this->M_General->update_kas('kas_masuk',
    $totalPembayaran);
    }

    $this->outputJSON($tanggalList);
}
```

Tabel 2 Hasil Refactoring Class Catering (generateTanggalCatering)

```
private function generateTanggalCatering($tanggalAwal, $totalHari,
    $idSiswa, $harga) {
    $tanggalList = [];
    $hariTerproses = 0;
    $offset = 0;

    while ($hariTerproses < $totalHari) {
        $tanggal = date('Y-m-d', strtotime("+ $offset days",
    strtotime($tanggalAwal)));
    }
```



```

        if ($this->isTanggalValid($tanggal, $idSiswa)) {
            $tanggalList[] = [
                'waktu' => $tanggal,
                'nominal' => $harga,
                'time' => waktu(),
                'id_siswa' => $idSiswa,
            ];
            $hariTerproses++;
        }

        $offset++;
    }

    return $tanggalList;
}

```

Tabel 3 Hasil Refactoring Class Catering (isTanggalValid)

```

private function isTanggalValid($tanggal, $idSiswa) {
    $isTanggalTersedia = $this->db->query("SELECT id FROM tanggal
    WHERE tgl = '$tanggal'")->num_rows() == 0;
    $isSiswaBelumBayar = $this->db->query("SELECT id FROM catering
    WHERE waktu = '$tanggal' AND id_siswa = '$idSiswa'")->num_rows() ==
    0;
    $isHariKerja = !in_array(date("D", strtotime($tanggal)), ["Sun",
    "Fri", "Sat"]);

    return $isTanggalTersedia && $isSiswaBelumBayar &&
    $isHariKerja;
}

```

Tabel. 4 Hasil Refactoring Class Catering (UpdateData)

```

function UpdateData() {
    $idSiswa = $this->input->post('id');
    $tanggal = $this->input->post('tanggal');

    $queryExists = $this->isCateringExists($idSiswa, $tanggal);

    if ($queryExists) {
        $this->updateCateringData($idSiswa, $tanggal);
        $nominal = $this->getNominalPembayaran();
        $this->updateKasKeluar($nominal, $idSiswa);
        $response['status'] = TRUE;
    } else {
        $response['status'] = FALSE;
    }

    $this->outputJSON($response);
}

```

Tabel 5 Hasil Refactoring Class Catering (isCateringExists)

```
private function isCateringExists($idSiswa, $tanggal) {
    return $this->db->query("SELECT id FROM catering WHERE
    id_siswa = '$idSiswa' AND waktu = '$tanggal'")->num_rows() > 0;
}
```

Tabel. 6 Hasil Refactoring Class Catering (updateCateringData)

```
private function updateCateringData($idSiswa, $tanggal) {
    $this->db->query("UPDATE catering SET nominal = '0' WHERE
    id_siswa = '$idSiswa' AND waktu = '$tanggal'");
}
```

Tabel. 7 Hasil Refactoring Class Catering (getNominalPembayaran)

```
private function getNominalPembayaran() {
    return $this->db->query("SELECT nominal FROM pembayaran
    WHERE id = 4")->row_array()['nominal'];
}
```

Pada permasalahan *data clumps*, *data title*, *judul*, dan *icon* ditemukan digunakan secara bersamaan dalam konfigurasi tampilan. Perbaikan dilakukan menggunakan teknik *Group Data in Variables* dengan membuat metode *prepareViewData*.

Tabel 8 Hasil Refactoring Class Catering (Index)

```
public function index() {
    $this->setBreadcrumb(['SI Keuangan ' => 'Beranda', 'Uang ' . $this-
    >parents => $this->parents]);
    $data = $this->prepareViewData('Pembayaran Uang Catering');
    $this->template->views('Backend/' . $this->parents . '/v_' . $this-
    >parents, $data);
}
```

Tabel. 9 Hasil Refactoring Class Catering (Detail)

```
function Detail($id) {
    $this->setBreadcrumb([
        'SI Keuangan ' => base_url(),
        $this->parents => base_url('Catering'),
        'Detail Pembayaran Uang Catering' => $this->parents,
    ]);
    $data = $this->prepareViewData('Pembayaran Uang Catering');
    $this->template->views('Backend/' . $this->parents . '/v_Detail',
    $data);
}
```

Tabel. 10 Method Tambahan Untuk Data Clumps

```
private function prepareViewData($pageTitle) {
    return [
        'title' => $pageTitle . ' | SI Keuangan',
    ];
}
```

```
'judul' => $pageTitle,
'icon' => $this->icon,
];
}
```

Pemisahan metode tersebut membuat setiap fungsi memiliki tanggung jawab yang lebih spesifik sehingga struktur kode menjadi lebih terorganisasi. Penggunaan *Extract Method* untuk menangani *long method* juga didukung oleh Shahidi et al. (2022).

Class Gaji

Pada *class Gaji* ditemukan *long method* pada metode *Simpan*. Metode tersebut menggabungkan proses validasi, pemeriksaan data ganda, perhitungan gaji, persiapan data, penyimpanan, pembaruan kas, dan keluaran JSON. Perbaikan dilakukan menggunakan *Extract Method* dengan membuat metode *isGajiExist*, *calculateTotalGaji*, *saveGaji*, dan *outputJson*.

Tabel. 11 Hasil Refactoring Class Gaji (Simpan)

```
function Simpan() {
    $id_guru = $this->input->post('guru');
    $bulan = filter_string($this->input->post('bulan', TRUE));
    $jam = filter_string($this->input->post('jam', TRUE));
    $nominal = filter_string($this->input->post('gaji', TRUE));

    // Validasi apakah gaji sudah ada
    if ($this->isGajiExist($id_guru, $bulan)) {
        $this->outputJson(['status' => FALSE]);
        return;
    }

    // Hitung total gaji
    $total = $this->calculateTotalGaji($jam, $nominal);

    // Simpan data ke database
    $this->saveGaji($id_guru, $bulan, $jam, $nominal, $total);

    // Berikan respon sukses
    $this->outputJson(['status' => TRUE]);
}
```

Tabel 12 Hasil Refactoring Class Gaji (isGajiExist)

```
private function isGajiExist($id_guru, $bulan) {
    return $this->db->query("SELECT id FROM gaji WHERE id_guru =
'$id_guru' AND periode = '$bulan'")->num_rows() > 0;
}
```

Tabel 13 Hasil Refactoring Class Gaji (calculateTotalGaji)

```
private function calculateTotalGaji($jam, $nominal) {
    return $jam * $nominal;
}
```

Tabel 14 Hasil Refactoring Class Gaji (saveGaji)

```
private function saveGaji($id_guru, $bulan, $jam, $nominal, $total) {
    $data = [
        'id_guru' => $id_guru,
        'periode' => $bulan,
        'time' => waktu(),
        'jam' => $jam,
        'nominal' => $nominal
    ];
    $this->M_General->insert($this->table, $data);
    $this->M_General->update_kas('kas_keluar', $total);
}
```

Tabel 15 Hasil Refactoring Class Gaji (outputJson)

```
private function outputJson($data) {
    $this->output->set_content_type('application/json')-
    >set_output(json_encode($data));
}
```

Selain *long method*, ditemukan *lack of input validation*. Perbaikan dilakukan menggunakan teknik *Introduce Parameter Validation* melalui *form_validation*. Validasi diterapkan pada data guru, bulan, jam, dan gaji.

Tabel. 16 Hasil Refactoring Class Gaji (Simpan)

```
public function Simpan() {
    $this->form_validation->set_rules('guru', 'Guru', 'required');
    $this->form_validation->set_rules('bulan', 'Bulan', 'required');
    $this->form_validation->set_rules('jam', 'Jam',
    'required|numeric');
    $this->form_validation->set_rules('gaji', 'Gaji',
    'required|numeric');

    if ($this->form_validation->run() == FALSE) {
        $data = [
            'status' => FALSE,
            'errors' => validation_errors()
        ];
    } else {
        $id_gur = $this->input->post('guru');
        $bln = filter_string($this->input->post('bulan', TRUE));
        $cek = $this->db->query("SELECT id FROM gaji WHERE id_guru
        = '$id_gur' AND periode = '$bln'")->num_rows();
        $jam = filter_string($this->input->post('jam', TRUE));
        $nominal = filter_string($this->input->post('gaji', TRUE));
        $total = $jam * $nominal;

        if ($cek > 0) {
            $data['status'] = FALSE;
        } else {
            $insert = [
```

```

        'id_guru' => $id_gur,
        'periode' => $bln,
        'time'    => waktu(),
        'jam'     => $jam,
        'nominal' => $nominal
    ];

    $this->M_General->insert($this->table, $insert);
    $this->M_General->update_kas('kas_keluar', $total);
    $data['status'] = TRUE;
    }
}

$this->output
->set_content_type('application/json')
->set_output(json_encode($data));
}

```

Penambahan validasi bertujuan memastikan data masukan memenuhi aturan sebelum diteruskan ke proses penyimpanan. Validasi input merupakan bagian penting dalam menjaga kualitas pemrosesan data pada perangkat lunak (Bosse Brinhosa et al., 2012).

Class Guru

Pada *class Guru* ditemukan *duplicate code* pada metode *Simpan* dan *Ubah*. Kedua metode memiliki proses pengolahan data yang sama sehingga dilakukan *Extract Method* melalui metode *prepareData*.

Tabel 17 Hasil refactoring Duplicate Code Simpan Class Guru

```

public function simpan() {
    // Validasi input
    $this->form_validation->set_rules('nama', 'Nama', 'required');
    $this->form_validation->set_rules('gender', 'Gender', 'required');
    $this->form_validation->set_rules('nip', 'NIP', 'required');
    $this->form_validation->set_rules('bidang', 'Bidang', 'required');
    $this->form_validation->set_rules('alamat', 'Alamat', 'required');
    $this->form_validation->set_rules('status', 'Status', 'required');
    $this->form_validation->set_rules('telepon', 'Telepon', 'required');

    if ($this->form_validation->run() == FALSE) {
        // Jika validasi gagal
        $this->sendJsonResponse(FALSE, validation_errors());
        return;
    }

    // Siapkan data
    $data = $this->prepareData();

    // Simpan data ke database
    if ($this->M_General->insert(self::TABLE, $data)) {
        $this->sendJsonResponse(TRUE);
    }
}

```

```

    } else {
        $this->sendJsonResponse(FALSE, 'Gagal menyimpan data.');
```

Tabel 18 Hasil Refactoring Duplicate Code Ubah Class Guru

```

public function ubah() {
    // Validasi input
    $this->form_validation->set_rules('id', 'ID', 'required');
    $this->form_validation->set_rules('nama', 'Nama', 'required');
    $this->form_validation->set_rules('gender', 'Gender', 'required');
    $this->form_validation->set_rules('nip', 'NIP', 'required');
    $this->form_validation->set_rules('bidang', 'Bidang', 'required');
    $this->form_validation->set_rules('alamat', 'Alamat', 'required');
    $this->form_validation->set_rules('status', 'Status', 'required');
    $this->form_validation->set_rules('telepon', 'Telepon', 'required');

    if ($this->form_validation->run() == FALSE) {
        // Jika validasi gagal
        $this->sendJsonResponse(FALSE, validation_errors());
        return;
    }

    // Siapkan data
    $data = $this->prepareData();

    // Update data ke database
    if ($this->M_General->update(self::TABLE, $data, 'id', $this->
    >input->post('id')) {
        $this->sendJsonResponse(TRUE);
    } else {
        $this->sendJsonResponse(FALSE, 'Gagal mengubah data.');
```

Selain itu, ditemukan *lack of input validation*. Validasi ditambahkan menggunakan *form_validation* untuk memastikan data yang dimasukkan memenuhi aturan sebelum diproses.

Tabel. 19 Hasil Refactoring Class Guru (Simpan)

```

public function simpan() {
    // Validasi input
    $this->form_validation->set_rules('nama', 'Nama', 'required');
    $this->form_validation->set_rules('gender', 'Gender',
    'required');
    $this->form_validation->set_rules('nip', 'NIP', 'required');
    $this->form_validation->set_rules('bidang', 'Bidang',
    'required');
    $this->form_validation->set_rules('alamat', 'Alamat',
    'required');
```

```

$this->form_validation->set_rules('status', 'Status', 'required');
$this->form_validation->set_rules('telepon', 'Telepon',
'required');

if ($this->form_validation->run() == FALSE) {
    // Jika validasi gagal
    $this->sendJsonResponse(FALSE, validation_errors());
    return;
}

// Siapkan data
$data = $this->prepareData();

// Simpan data ke database
if ($this->M_General->insert(self::TABLE, $data)) {
    $this->sendJsonResponse(TRUE);
} else {
    $this->sendJsonResponse(FALSE, 'Gagal menyimpan data.');
```

Tabel 20 Hasil Refactoring Class Guru (Ubah)

```

public function ubah() {
    // Validasi input
    $this->form_validation->set_rules('id', 'ID', 'required');
    $this->form_validation->set_rules('nama', 'Nama', 'required');
    $this->form_validation->set_rules('gender', 'Gender',
'required');
    $this->form_validation->set_rules('nip', 'NIP', 'required');
    $this->form_validation->set_rules('bidang', 'Bidang',
'required');
    $this->form_validation->set_rules('alamat', 'Alamat',
'required');
    $this->form_validation->set_rules('status', 'Status', 'required');
    $this->form_validation->set_rules('telepon', 'Telepon',
'required');

    if ($this->form_validation->run() == FALSE) {
        // Jika validasi gagal
        $this->sendJsonResponse(FALSE, validation_errors());
        return;
    }

    // Siapkan data
    $data = $this->prepareData();

    // Update data ke database
    if ($this->M_General->update(self::TABLE, $data, 'id', $this-
>input->post('id')) {
        $this->sendJsonResponse(TRUE);
```

```

    } else {
        $this->sendJsonResponse(FALSE, 'Gagal mengubah data.');
```

Metode *prepareData* digunakan untuk menyiapkan data nama, jenis kelamin, NIP, bidang, alamat, status, dan nomor sehingga proses pengolahan data tidak ditulis berulang.

Tabel 21 Menambahkan *prepareData*

```

private function prepareData(): array {
    return [
        'name' => filter_string(ucwords($this->input->post('nama',
TRUE))),
        'sex'  => $this->input->post('gender', TRUE),
        'nip'  => $this->input->post('nip', TRUE),
        'bidang' => filter_string($this->input->post('bidang', TRUE)),
        'alamat' => filter_string($this->input->post('alamat', TRUE)),
        'status' => filter_string($this->input->post('status', TRUE)),
        'number' => filter_string($this->input->post('telepon', TRUE))
    ];
}
```

Class Lainnya

Pada *class Lainnya* ditemukan *lack of input validation* pada metode *Simpan*. Sebelumnya data dapat diproses tanpa validasi yang memadai. Perbaikan dilakukan menggunakan *Introduce Parameter Validation* melalui *form_validation*.

Tabel. 22 Hasil Refactoring Class Lainnya (Simpan)

```

public function Simpan(){
    $this->form_validation->set_rules('nominal', 'Nominal',
'required|numeric');
    $this->form_validation->set_rules('jenis', 'Jenis', 'required');
    $this->form_validation->set_rules('keterangan', 'Keterangan',
'required');

    if ($this->form_validation->run() === FALSE) {
        $response = ['status' => FALSE, 'errors' =>
validation_errors()];
    } else {
        $total = filter_string($this->input->post('nominal', TRUE));

        $data = [
            'nominal'  => $total,
            'sekarang' => sekarang(),
            'time'     => waktu(),
            'jenis'    => filter_string($this->input->post('jenis', TRUE)),
            'keterangan' => filter_string($this->input->post('keterangan',
TRUE)),
        ];
    }
}
```

```

        $this->M_General->insert($this->table, $data);
        $this->M_General->update_kas('kas_masuk', $total);

        $response = ['status' => TRUE];
    }

    $this->output->set_content_type('application/json')
        ->set_output(json_encode($response));
}

```

Validasi dilakukan terhadap data nominal, jenis, dan keterangan sebelum data diproses lebih lanjut.

Class Pendaftaran

Pada *class Pendaftaran* ditemukan *duplicate code* pada metode *Simpan*, *Kelas*, dan *Ubah*. Ketiga metode memiliki proses pengambilan dan pengolahan data yang berulang. Perbaikan dilakukan menggunakan *Extract Method* dengan membuat metode *getInputData*.

Tabel. 23 Hasil Refactoring Class Pendaftaran (Simpan)

```

public function Simpan() {
    $data = $this->getInputData();
    $this->insertTempData($data);
    $this->session->set_flashdata('success', 'Berhasil menambahkan
    Data Baru!');

    redirect($this->uri->segment(1), 'refresh');
}

```

Tabel 24 Hasil Refactoring Class Pendaftaran (Kelas)

```

public function Kelas() {
    $data = $this->getInputData();
    $data['status'] = 'Aktif';
    $data['kelas'] = $this->input->post('kelas');

    $this->insertSiswaData($data);
    $this->deleteTempData($this->input->post('id'));

    $this->session->set_flashdata('success', 'Siswa Baru Berhasil
    ditambahkan!');
    redirect($this->uri->segment(1), 'refresh');
}

```

Tabel 25 Hasil Refactoring Class Pendaftaran (Ubah)

```

public function Ubah() {
    $data = $this->getInputData();
    $id = $this->input->post('id');

    $this->updateTempData($data, $id);

    $this->session->set_flashdata('success', 'Data berhasil diubah.');
```

```

        redirect($this->uri->segment(1), 'refresh');
    }

```

Tabel 26 Method getInputData

```

private function getInputData() {
    return [
        'name' => filter_string(ucwords($this->input->post('nama',
TRUE))),
        'sex' => $this->input->post('gender', TRUE) ?? 'Tidak Diketahui',
        'nis' => $this->input->post('nis', TRUE),
        'tempat' => filter_string($this->input->post('tempat', TRUE)),
        'tanggal' => filter_string($this->input->post('tanggal', TRUE)),
        'alamat' => filter_string($this->input->post('alamat', TRUE)),
        'wali' => filter_string($this->input->post('wali', TRUE)),
    ];
}

```

Tabel 27 Method insertTempData

```

private function insertTempData($data) {
    $this->db->insert($this->table, $data);
}

```

Tabel 28 Method insertSiswaData

```

private function insertSiswaData($data) {
    $this->db->insert('siswa', $data);
}

```

Tabel 29 Method deleteTempData

```

private function deleteTempData($id) {
    $this->db->delete($this->table, ['id' => $id]);
}

```

Tabel 30 Method updateTempData

```

private function updateTempData($data, $id) {
    $this->db->update($this->table, $data, ['id' => $id]);
}

```

Metode *getInputData* digunakan untuk mengambil data nama, jenis kelamin, NIS, tempat, tanggal, alamat, dan wali. Dengan demikian, proses pengambilan data yang sama dapat digunakan oleh beberapa metode.

Pada *incomplete abstraction*, ditemukan bahwa setelah proses perubahan data sistem tidak mengarahkan pengguna kembali ke halaman pendaftaran. Perbaikan dilakukan dengan menambahkan *redirect* setelah proses pembaruan data selesai.

Tabel 31 Hasil Refactoring Class Pendaftaran

```

public function Ubah() {
    $data = $this->getInputData();
    $id = $this->input->post('id');
}

```

```

$this->updateTempData($data, $id);

$this->session->set_flashdata('success', 'Data berhasil diubah.');
```

redirect(\$this->uri->segment(1), 'refresh');

```

}
```

Pada *hardcoded logic*, ditemukan *query* yang ditulis secara langsung pada metode *Index*, *Bayar*, dan *Kelas*. Perbaikan dilakukan dengan memisahkan proses akses data ke metode khusus.

Tabel 32 Hasil Refactoring Class Pendaftaran (Index)

```

public function index() {
    $this->set_breadcrumb();

    $data = [
        'title' => "{$this->parents} | SI Keuangan",
        'judul' => $this->parents,
        'icon' => $this->icon,
        'isi' => $this->getTempData(),
        'bayar' => $this->getBayarData(5),
    ];

    $this->template->views("Backend/{$this->parents}/v_{$this->parents}", $data);
}
```

Tabel 33 Hasil Refactoring Class Pendaftaran (Bayar)

```

public function Bayar() {
    $id = $this->input->post('id');
    $this->updateTempBayar($id);

    $paymentData = [
        'siswa' => $this->input->post('nama'),
        'nominal' => $this->input->post('harga'),
        'time' => waktu(),
    ];
    $this->insertPendaftaranData($paymentData);
    $this->updateKas('kas_masuk', $this->input->post('harga'));

    $this->session->set_flashdata('success', 'Pembayaran Uang
Pendaftaran Berhasil!');
    redirect($this->uri->segment(1), 'refresh');
}
```

Tabel 34 Hasil Refactoring Class Pendaftaran (Kelas)

```

public function Kelas() {
    $data = $this->getInputData();
    $data['status'] = 'Aktif';
    $data['kelas'] = $this->input->post('kelas');

    $this->insertSiswaData($data);
}
```

```

$this->deleteTempData($this->input->post('id'));

$this->session->set_flashdata('success', 'Siswa Baru Berhasil
ditambahkan!');
redirect($this->uri->segment(1), 'refresh');
}

```

Tabel 35 Method getTempData

```

private function getTempData() {
    return $this->db->get($this->table)->result();
}

```

Tabel 36 Method getBayarData

```

private function getBayarData($id) {
    return $this->db->get_where('pembayaran', ['id' => $id])-
>row_array();
}

```

Tabel 37 Method insertPendaftaranData

```

private function insertPendaftaranData($data) {
    $this->db->insert('pendaftaran', $data);
}

```

Tabel 38 Method updateKas

```

private function updateKas($type, $amount) {
    $this->M_General->update_kas($type, $amount);
}

```

Tabel 39 Method insertSiswaData

```

private function insertSiswaData($data) {
    $this->db->insert('siswa', $data);
}

```

Tabel 40 Method deleteTempData

```

private function deleteTempData($id) {
    $this->db->delete($this->table, ['id' => $id]);
}

```

Abstraksi tersebut membuat proses utama *controller* lebih terstruktur karena proses akses basis data dipisahkan dari alur utama metode.

Class Pengeluaran

Pada *class Pengeluaran* ditemukan *duplicate code* pada metode *Index* dan *Detail*, terutama pada pengaturan *breadcrumb* dan data tampilan. Perbaikan dilakukan menggunakan teknik *Extract Method* dengan membuat metode *setBreadcrumb* dan *setViewData*.

Tabel. 41 Hasil Refactoring Class Pengeluaran (Index)

```

public function index() {
    // Menggunakan fungsi setViewData untuk mempersiapkan data
    halaman
}

```

```

$data = $this->setViewData(
    $this->parents . ' Lainnya',
    $this->parents . ' Lainnya'
);

// Menyiapkan breadcrumb
$this->setBreadcrumb([
    'SI Keuangan ' => 'Beranda',
    $this->parents . ' Lainnya' => $this->parents
]);

// Menampilkan view
$this->template->views('Backend/' . $this->parents . '/v_' . $this->parents, $data);
}

```

Tabel. 42 Hasil Refactoring Class Pengeluaran (Detail)

```

public function Detail() {
    // Menggunakan fungsi setViewData untuk mempersiapkan data
    halaman
    $data = $this->setViewData(
        'Detail ' . $this->parents . ' Lainnya',
        'Detail ' . $this->parents . ' Lainnya'
    );

    // Menyiapkan breadcrumb
    $this->setBreadcrumb([
        'SI Keuangan ' => base_url(),
        $this->parents => base_url('Pengeluaran'),
        'Detail Pengeluaran Lainnya' => $this->parents
    ]);

    // Menampilkan view
    $this->template->views('Backend/' . $this->parents . '/v_Detail',
    $data);
}

```

Pemisahan tersebut mengurangi pengulangan kode dan membuat pengaturan data tampilan dapat digunakan secara lebih terstruktur.

Class Snack

Pada *class Snack* ditemukan *long method* pada metode *Simpan* dan *UpdateData*. Metode *Simpan* sebelumnya menggabungkan pengambilan input, pengolahan tanggal, validasi, penyimpanan data, pembaruan kas, dan keluaran JSON. Perbaikan dilakukan menggunakan *Extract Method*.

Tabel 43 Hasil Refactoring Class Snack (Simpan)

```

public function simpan() {
    $tanggal = filter_string($this->input->post('tanggal', TRUE));
}

```



```

    $harga = $this->input->post('harga');
    $total_hari = filter_string($this->input->post('total', TRUE));
    $id_siswa = $this->input->post('id_siswa');

    $data = $this->generate_snack_data($tanggal, $harga,
    $total_hari, $id_siswa);
    $this->db->insert_batch($this->table, $data);

    $this->M_General->update_kas('kas_masuk', $this->input-
    >post('seluruh'));
    $this->output_json($data);
}

```

Tabel 44 Method generate_snack_data

```

private function generate_snack_data($start_date, $harga,
$total_days, $id_siswa) {
    $data = [];
    $days_added = 0;
    $i = 0;

    while ($days_added < $total_days) {
        $current_date = date('Y-m-d', strtotime("+ $i days",
        strtotime($start_date)));
        $is_weekend = date('D', strtotime($current_date)) === "Sun";
        $date_exists = $this->db->where('tgl', $current_date)-
        >count_all_results('tanggal') > 0;
        $snack_exists = $this->db->where(['waktu' => $current_date,
        'id_siswa' => $id_siswa])
        ->count_all_results($this->table) > 0;

        if (!$is_weekend && !$date_exists && !$snack_exists) {
            $data[] = [
                'waktu' => $current_date,
                'nominal' => $harga,
                'time' => waktu(),
                'id_siswa' => $id_siswa
            ];
            $days_added++;
        }

        $i++;
    }

    return $data;
}

```

Tabel 45 Method OutputJSON

```
private function output_json($data) {
    $this->output->set_content_type('application/json')-
    >set_output(json_encode($data));
}
```

Tabel 46 Hasil Refactoring Class Snack (UpdateData)

```
public function updateData() {
    $id_siswa = $this->input->post('id');
    $tanggal = $this->input->post('tanggal');
    $siswa = get_siswa($id_siswa);

    $snack_exists = $this->db->where(['id_siswa' => $id_siswa, 'waktu'
=> $tanggal])
        ->count_all_results($this->table) > 0;

    if ($snack_exists) {
        $this->mod->update_snack($id_siswa, $tanggal, $siswa);
        $this->output_json(['status' => true]);
    } else {
        $this->output_json(['status' => false]);
    }
}
```

Metode *generate_snack_data* digunakan untuk memproses tanggal, memeriksa hari Minggu, mengecek keberadaan tanggal dan snack, serta membentuk data. Sementara itu, *output_json* digunakan untuk menangani respons JSON.

Class Transaksi

Pada *class Transaksi* ditemukan *duplicate code* pada metode *Simpan* dan *Ubah*, terutama pada proses menghasilkan respons JSON. Perbaikan dilakukan menggunakan *Extract Method* dengan membuat metode *sendJsonResponse*.

Tabel 47 Hasil Refactoring Class Transaksi (Simpan)

```
function Simpan(){
    $data = $this->prepareDataForInsert();
    $this->insertTransaksi($data);
    $this->sendJsonResponse(['status' => TRUE]);
}
```

Tabel 48 Hasil Refactoring Class Transaksi (Ubah)

```
function Ubah(){
    $data = $this->prepareDataForUpdate();
    $this->updateTransaksi($data);
    $this->sendJsonResponse(['status' => TRUE]);
}
```

Tabel 49 Method sendJsonRespon

```
private function sendJsonResponse($data){
    $this->output->set_content_type('application/json')-
    >set_output(json_encode($data));
}
```



```
}

```

Pemisahan tersebut mengurangi pengulangan kode karena respons JSON yang sebelumnya terdapat pada kedua metode dipusatkan pada *sendJsonResponse*.

Class Ujian

Pada *class Ujian* ditemukan *hardcoded logic* pada metode *getUjian* dan *Simpan* karena terdapat *query* yang ditulis secara langsung. Perbaikan dilakukan melalui abstraksi *query* dengan membuat metode *getNominalUjian* dan *isDuplicateEntry*.

Tabel 50 Hasil Refactoring Class Ujian (getUjian)

```
public function getUjian() {
    $this->outputJson($this->getNominalUjian());
}
```

Tabel 51 Hasil Refactoring Class Ujian (Simpan)

```
public function Simpan() {
    $id = $this->input->post('id', TRUE);
    $bln = filter_string($this->input->post('bulan', TRUE));
    $total = filter_string($this->input->post('harga', TRUE));

    // Validasi input
    if ($this->isDuplicateEntry($id, $bln)) {
        $this->outputJson(['status' => FALSE]);
        return;
    }

    // Siapkan data untuk disimpan
    $dataInsert = $this->prepareInsertData($id, $bln, $total);

    // Simpan data
    $this->M_General->insert($this->table, $dataInsert);
    $this->M_General->update_kas('kas_masuk', $total);

    $this->outputJson(['status' => TRUE]);
}
```

Tabel 52 Method getNominalUjian

```
private function getNominalUjian() {
    $result = $this->db->query("SELECT nominal FROM pembayaran
    WHERE id = 2")->row_array();
    return $result['nominal'] ?? null;
}
```

Tabel 53 Method isDuplicateEntry

```
private function isDuplicateEntry($id, $bln) {
    return $this->db->query("SELECT id FROM ujian WHERE id_siswa =
    '$id' AND periode = '$bln'")->num_rows() > 0;
}
```

Secara keseluruhan, penerapan *refactoring* dilakukan pada sembilan *class controller* dengan enam jenis *code smells*. Teknik yang digunakan meliputi *Extract Method*, *Group Data in Variables*, *Introduce Parameter Validation*, *Introduce Missing Methods or Properties*, dan abstraksi *query*. Teknik *Extract Method* menjadi teknik yang paling banyak digunakan karena permasalahan yang ditemukan banyak berkaitan dengan metode yang terlalu panjang dan pengulangan kode (Charalampidou et al., 2017; Shahidi et al., 2022).

Hasil *Black-Box Testing*

Setelah penerapan *refactoring*, dilakukan *Black-Box Testing* untuk memastikan bahwa perubahan struktur kode tidak menyebabkan perubahan terhadap fungsionalitas Sistem Informasi Keuangan. Pengujian dilakukan terhadap fungsi yang mengalami perubahan dengan membandingkan hasil yang diharapkan dengan hasil aktual sistem.

Berdasarkan hasil pengujian, terdapat 22 kasus uji yang dilakukan pada fungsi-fungsi Sistem Informasi Keuangan. Seluruh kasus uji menghasilkan status sesuai dan tidak terdapat kasus uji yang tidak sesuai.

Jumlah kasus uji = 22

Sesuai = 22

Tidak sesuai = 0

Persentase keberhasilan pengujian dihitung dengan persamaan:

$$\begin{aligned}\text{Persentase keberhasilan} &= (\text{Jumlah kasus sesuai} / \text{Jumlah seluruh kasus uji}) \times 100\% \\ &= (22/22) \times 100\% = 100\%\end{aligned}$$

Hasil tersebut menunjukkan bahwa seluruh fungsi yang diuji tetap berjalan sesuai dengan keluaran yang diharapkan setelah dilakukan *refactoring*. Dengan demikian, perubahan struktur kode tidak menyebabkan kegagalan pada fungsi Sistem Informasi Keuangan yang diuji.

Hasil Pengukuran *Software Metrics*

Evaluasi kualitas kode dilakukan dengan membandingkan nilai *software metrics* sebelum dan sesudah *refactoring*. Metrik yang digunakan terdiri atas *Lines of Code* (LOC), *Cyclomatic Complexity* (CC), *Halstead Volume* (V), dan *Maintainability Index* (MI). Penggunaan metrik tersebut dapat memberikan gambaran kuantitatif mengenai karakteristik dan kualitas kode (Agnihotri & Chug, 2020; Syaifullah et al., 2022).

Sebelum *Refactoring*

Tabel 54 Nilai Metrics sebelum di Refactoring

No	Class	LOC (PerNode)	CC	HALSTEAD VOLUME (V)	MI
1	Catering	11	14	16	16
2	Gaji	9	12	14	14
3	Guru	9	12	14	14
4	Lainnya	9	12	14	14
5	Pendaftaran	11	15	19	19
6	Pengeluaran	9	12	14	14
7	Snack	11	14	16	16
8	Transaksi	10	13	15	15



9	Ujian	9	12	14	14
---	-------	---	----	----	----

Sebelum *refactoring*, nilai LOC berada pada rentang 9-11, nilai CC berada pada rentang 12-15, *Halstead Volume* berada pada rentang 14-19, dan MI berada pada rentang 14-19. *Class Pendaftaran* memiliki nilai CC dan MI tertinggi, sedangkan beberapa *class*, yaitu *Gaji*, *Guru*, *Lainnya*, *Pengeluaran*, dan *Ujian*, memiliki nilai MI terendah.

Tabel 55 Nilai Metrics Seluruh Controller Sebelum Refactoring

Class	LOC (PerNode)	CC	HALSTEAD VOLUME (V)	MI
Seluruh Controller	148	199	205	205

Nilai keseluruhan sebelum *refactoring* menunjukkan LOC sebesar 148, CC sebesar 199, *Halstead Volume* sebesar 205, dan MI sebesar 205. Nilai tersebut digunakan sebagai kondisi awal untuk dibandingkan dengan hasil setelah penerapan *refactoring*.

Sesudah Refactoring

Tabel 56 Nilai Metrics Sesudah di Refactoring

No	Class	LOC (PerNode)	CC	HALSTEAD VOLUME (V)	MI
1	Catering	20	23	25	25
2	Gaji	14	17	19	19
3	Guru	12	12	15	15
4	Lainnya	11	13	15	15
5	Pendaftaran	25	29	33	33
6	Pengeluaran	13	16	18	18
7	Snack	14	17	19	19
8	Transaksi	17	20	22	22
9	Ujian	15	18	20	20

Setelah *refactoring*, nilai LOC meningkat pada seluruh *class*. Peningkatan terbesar terdapat pada *Pendaftaran*, dari 11 menjadi 25, sedangkan *Catering* meningkat dari 11 menjadi 20. Peningkatan LOC dapat terjadi karena kode yang sebelumnya terkonsentrasi dalam satu metode dipisahkan menjadi beberapa metode yang memiliki tanggung jawab lebih spesifik.

Nilai CC juga mengalami peningkatan pada sebagian besar *class*. *Catering* meningkat dari 14 menjadi 23 dan *Pendaftaran* dari 15 menjadi 29. Sementara itu, *Halstead Volume* juga mengalami peningkatan pada seluruh *class*.

Hal yang menjadi fokus utama penelitian adalah nilai MI. Setelah *refactoring*, MI meningkat pada seluruh *class*. *Catering* meningkat dari 16 menjadi 25, *Gaji* dari 14 menjadi 19, *Guru* dari 14 menjadi 15, *Lainnya* dari 14 menjadi 15, *Pendaftaran* dari 19 menjadi 33, *Pengeluaran* dari 14 menjadi 18, *Snack* dari 16 menjadi 19, *Transaksi* dari 15 menjadi 22, dan *Ujian* dari 14 menjadi 20.

Tabel 57 Hasil Refactoring Keseluruhan Class

Class	LOC (PerNode)	CC	HALSTEAD VOLUME (V)	MI
Seluruh Controller	206	256	262	262



Nilai keseluruhan setelah *refactoring* menunjukkan LOC sebesar 206, CC sebesar 256, *Halstead Volume* sebesar 262, dan MI sebesar 262.

[Masukkan Tabel Perbandingan Metrics Sebelum dan Sesudah di sini]

Berdasarkan perbandingan tersebut, LOC meningkat dari 148 menjadi 206, CC meningkat dari 199 menjadi 256, *Halstead Volume* meningkat dari 205 menjadi 262, sedangkan MI meningkat dari 205 menjadi 262.

Peningkatan LOC terjadi karena proses *refactoring* memisahkan kode yang sebelumnya memiliki beberapa tanggung jawab ke dalam metode yang lebih spesifik. Oleh karena itu, peningkatan jumlah baris kode tidak secara langsung menunjukkan penurunan kualitas *maintainability*. Penilaian kualitas kode perlu mempertimbangkan tujuan perubahan struktur dan beberapa metrik yang digunakan (Agnihotri & Chug, 2020).

Peningkatan CC dan *Halstead Volume* menunjukkan adanya perubahan karakteristik kode berdasarkan hasil pengukuran *PHP Depend*. Namun, indikator utama yang digunakan dalam penelitian ini adalah MI. Peningkatan MI dari 205 menjadi 262 menunjukkan adanya peningkatan nilai *maintainability* berdasarkan indikator yang digunakan.

Secara keseluruhan, penerapan *refactoring* berhasil memperbaiki *code smells* pada sembilan *class controller*, mempertahankan fungsionalitas sistem dengan tingkat keberhasilan *Black-Box Testing* sebesar 100%, serta meningkatkan nilai MI keseluruhan dari 205 menjadi 262. Hasil tersebut menunjukkan bahwa penerapan *refactoring* memberikan dampak positif terhadap *maintainability* Sistem Informasi Keuangan berdasarkan nilai MI.

4. KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa penerapan *refactoring* berhasil meningkatkan kualitas *maintainability* pada Sistem Informasi Keuangan (SIK). Proses *refactoring* mampu menangani enam jenis *code smells*, yaitu *Long Method*, *Duplicate Code*, *Lack of Validation*, *Hardcoded Logic*, *Data Clumps*, dan *Incomplete Abstraction*. Teknik yang digunakan disesuaikan dengan karakteristik masing-masing *code smells*, yaitu *Extract Method* untuk *Long Method* dan *Duplicate Code*, *Introduce Parameter Validation* untuk *Lack of Validation*, *Group Data in Variable* untuk *Data Clumps*, *Introduce Missing Methods or Properties* untuk *Incomplete Abstraction*, serta abstraksi *Query* untuk *Hardcoded Logic*.

Hasil pengujian menggunakan *Black-Box Testing* terhadap 22 kasus menunjukkan bahwa seluruh fungsi sistem berjalan sesuai dengan hasil yang diharapkan, dengan tingkat keberhasilan sebesar 100%. Selain itu, hasil pengukuran *software metrics* menunjukkan adanya peningkatan nilai *Maintainability Index* setelah penerapan *refactoring*. Peningkatan tersebut menunjukkan bahwa struktur kode menjadi lebih terorganisasi sehingga lebih mudah dibaca, dipelihara, dan dikembangkan. Dengan demikian, penerapan *refactoring* terbukti efektif dalam meningkatkan *maintainability* perangkat lunak pada Sistem Informasi Keuangan.

DAFTAR PUSTAKA

- Agnihotri, M., & Chug, A. (2020). A systematic literature survey of software metrics, code smells and refactoring techniques. *Journal of Information Processing Systems*, 16(4), 915-934. <https://doi.org/10.3745/JIPS.04.0184>
- Almogahed, A., & Omar, M. (2021). Refactoring techniques for improving software quality: Practitioners' perspectives. *Journal of Information and Communication Technology*, 20(4), 511-539. <https://doi.org/10.32890/jict2021.20.4.3>



- Andriani, R., Sabrina, P. N., & Ashaury, H. (2025). Metode refactoring untuk meningkatkan kualitas maintainability pada sistem informasi puskesmas. *Jurnal Tekno Kompak*, 19(1), 227-239. <https://doi.org/10.33365/jtk.v19i1.4686> (E-Jurnal Teknokrat)
- Baumgartner, N., Adleh, F., & Pulvermüller, E. (2023). Live code smell detection of data clumps in an integrated development environment. In *International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)* (pp. 64-76). Science and Technology Publications. <https://doi.org/10.5220/0011727500003464>
- Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., Gkortzis, A., & Avgeriou, P. (2017). Identifying Extract Method refactoring opportunities based on functional relevance. *IEEE Transactions on Software Engineering*, 43(10), 954-974. <https://doi.org/10.1109/TSE.2016.2645572>
- Dirjen, S. K., et al. (2017). Metode McCall's untuk pengujian kualitas sistem informasi administrasi tugas akhir (SIATA). [Nama jurnal tidak tercantum pada data sumber], 1(3), 488-495.
- Fredianto, H. (2021). Optimalisasi perangkat lunak menggunakan metode refactoring. *Jurnal Syntax Admiration*, 2(10), 1885-1902. <https://doi.org/10.46799/jsa.v2i10.326>
- Kusuma, E. (2024). Deteksi code smell dengan pendekatan machine learning: Analisis bibliometrik. *bit-Tech*, 7(2), 366-376. <https://doi.org/10.32877/bt.v7i2.1822>
- Sa'adah, U., Permatasari, D. I., Hardiansyah, F. F., Yunanto, A. A., Hasim, J. A. N., Wulandari, I., Pahlevi, M. R., & Shihab, D. Q. (2022). Tool refactoring otomatis untuk menangani lazy class code smell dengan pendekatan software metrics. *Jurnal Teknologi Informasi dan Ilmu Komputer*, 9(4), 743-750. <https://doi.org/10.25126/jtiik.202294646> (ResearchGate)
- Sasaki, Y., Hotta, K., Higo, Y., & Kusumoto, S. (2011). Is duplicate code good or bad? An empirical study with multiple investigation methods and multiple detection tools. In *Proceedings of the 22nd International Symposium on Software Reliability Engineering (ISSRE 2011)*. (Google Sites)
- Shahidi, M., Ashtiani, M., & Zakeri-Nasrabadi, M. (2022). An automated extract method refactoring approach to correct the long method code smell. *Journal of Systems and Software*, 187, 111221. <https://doi.org/10.1016/j.jss.2022.111221>
- Syaifullah, A. M., Rosyid, H., Aisyiyah, P., & Devi, R. (2022). Pengukuran kualitas sistem informasi perangkat lunak pada website E-Procurement PT. Wakabe Indonesia dengan metode McCall. *ILKOMNIKA: Journal of Computer Science and Applied Informatics*, 4(3), 274-283. <https://doi.org/10.28926/ilkomnika.v4i3.523>