OPTIMASI PERANGKAT LUNAK KEUANGAN DI KANTOR EKONOM KEUSKUPAN BANDUNG

Disusun oleh :

Ali Sadiyoko
Gunawan Juanda

LEMBAGA PENELITIAN DAN PENGABDIAN KEPADA MASYARAKAT
UNIVERSITAS KATOLIK PARAHYANGAN
BANDUNG
2009

LAPORAN PENELITIAN

OPTIMASI PERANGKAT LUNAK KEUANGAN DI KANTOR EKONOM KEUSKUPAN BANDUNG

Disusun oleh :

Ali Sadiyoko
Gunawan Juanda

JURUSAN TEKNIK INDUSTRI
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS KATOLIK PARAHYANGAN
2009

Daftar Isi

ABSTRAK

Kantor Ekonom Keuskupan Bandung memiliki peran yang sangat penting dalam mendukung kelancaran kegiatan organisasi dan pelayanan umat di wilayah Keuskupan Bandung. Dalam menjalankan tugasnya, ekonom keuskupan memerlukan alat bantu bantu berupa sebuah perangkat lunak komputer untuk mencatat semua aliran uang keluar dan masuk, baik melalui kas maupun bank. Sebuah perangkat lunak keuangan telah dibuat pada tahun 2002, namun dengan semakin meningkatnya jumlah data yang diolah, maka dirasakan bahwa perangkat lunak tersebut sudah tidak sesuai lagi. Kecepatan pengolahan data, terutama pada saat penyusunan laporan keuangan tahunan, dapat berlangsung lebih dari 45 menit. Waktu ini akan bertambah lama lagi jika data yang diolah semakin banyak, hal ini terjadi misalnya pada unit Kantor Ekonom. Di Kantor Ekonom inilah diolah seluruh data keuangan dari semua paroki, komisi dan unit usaha yang ada di bawah naungan Keuskupan Bandung.

Penelitian yang akan dilakukan ini, bertujuan untuk memenuhi permintaan administrator keuangan di Kantor Ekonom Keuskupan Bandung untuk lebih mengoptimalkan perangkat lunak yang telah ada. Beberapa permintaan yang diajukan oleh pihak Kantor Ekonom antara lain : peningkatan kecepatan proses penghitungan data, penambahan beberapa utilitas yang dapat mempermudah proses pengolahan anggaran, utilitas untuk proses konsolidasi laporan dari beberapa paroki/ unit/ komisi, utilitas untuk memeriksa realisasi anggaran yang telah terpakai sampai pada satu saat tertentu, perbaikan bentuk/ format laporan keuangan dan perbaikan *layout* tampilan program.

Untuk menyelesaikan masalah ini, pada penelitian ini akan dilakukan proses reverse engineering dari perangkat lunak yang sudah ada, dengan melakukan proses ekstraksi cara kerja utamanya dan dianalisis kelemahannya. Proses cara kerja perangkat lunak akan diperbaiki, disesuaikan kembali dengan perubahan tata-cara kerja baru yang diterapkan oleh pihak Kantor Ekonom.

Hasil akhir dari penelitian ini adalah sebuah perangkat lunak baru yang memenuhi kriteria desain seperti yang diinginkan oleh pihak Kantor Ekonom Keuskupan Bandung.

BAB 1

PENDAHULUAN

Kantor Ekonom Keuskupan Bandung adalah sebuah unit kerja di lingkungan Keuskupan Bandung yang mempunyai tugas mengatur semua masalah keuangan di seluruh wilayah kerja Keuskupan Bandung. Keuskupan Bandung sendiri memiliki sekitar 20 unit aktivitas yang memerlukan proses mengaturan keuangan, seperti proses penggajian karyawan, pengeluaran/ pembelanjaan harian/ rutin, baik untuk kebutuhan keagamaan maupun untuk kegiatan harian biasa. Unit aktivitas di wilayah Keuskupan Bandung ini termasuk di antaranya unit paroki, komisi, rumah Uskup dan unit usaha tertentu. Karena banyaknya unit aktivitas ini, maka Kantor Ekonom memiliki peran yang sangat penting dalam mendukung kelancaran kegiatan organisasi dan pelayanan umat di wilayah Keuskupan Bandung.

Dalam menjalankan tugasnya, ekonom keuskupan memerlukan alat bantu bantu berupa sebuah perangkat lunak komputer untuk mencatat semua aliran uang keluar dan masuk, baik melalui kas maupun bank. Sebuah perangkat lunak keuangan telah dibuat pada tahun 2002, namun dengan semakin meningkatnya jumlah data yang diolah, maka dirasakan bahwa perangkat lunak tersebut sudah tidak sesuai lagi. Kecepatan pengolahan data, terutama pada saat penyusunan laporan keuangan tahunan, dapat berlangsung lebih dari 45 menit. Waktu ini akan bertambah lama lagi jika data yang diolah semakin banyak, hal ini terjadi misalnya pada unit Kantor Ekonom. Di Kantor Ekonom inilah diolah seluruh data keuangan dari semua paroki, komisi dan unit usaha yang ada di bawah naungan Keuskupan Bandung.

Penelitian yang akan dilakukan ini, bertujuan untuk memenuhi permintaan administrator keuangan di Kantor Ekonom Keuskupan Bandung untuk lebih mengoptimalkan perangkat lunak yang telah ada. Beberapa permintaan yang diajukan oleh pihak Kantor Ekonom antara lain :

1. Peningkatan kecepatan proses penghitungan data.
2. Penambahan utilitas untuk memasukkan anggaran, yang juga dapat terhubung dengan utilitas pemasukan saldo awal tahun.
3. Penambahan utilitas untuk dapat melakukan proses konsolidasi laporan dari beberapa paroki/ unit/ komisi.
4. Penambahan utilitas untuk memeriksa realisasi anggaran yang telah terpakai sampai pada satu saat tertentu.
5. Perbaikan bentuk/ format laporan.
6. Perbaikan *layout* tampilan program, untuk lebih memudahkan penggunaan perangkat lunak ini, termasuk dalam hal ini adalah penambahan beberapa *short-cut* untuk membantu mempercepat proses pemasukan data.

1.1 Identifikasi Masalah

Penelitian yang akan dilakukan ini, bertujuan untuk memenuhi permintaan administrator keuangan di Kantor Ekonom Keuskupan Bandung untuk lebih mengoptimalkan perangkat lunak yang telah ada. Sebuah perangkat lunak baru perlu dikembangkan, tetapi dengan tetap mengacu pada pola perilaku pada perangkat lunak yang lama. Hal ini perlu dilakukan mengingat para pengguna sudah terbiasa dengan perangkat lunak yang lama. Kebutuhan dari perangkat lunak baru ini adalah :

1. Mempunyai kecepatan proses penghitungan data yang lebih cepat dari perangkat lunak lama.
2. Mempunyai utilitas untuk memasukkan anggaran, yang juga dapat terhubung dengan utilitas pemasukan saldo awal tahun.

3. Mempunyai utilitas untuk dapat melakukan proses konsolidasi laporan dari beberapa paroki/ unit/ komisi.
4. Mempunyai utilitas untuk memeriksa realisasi anggaran yang telah terpakai sampai pada satu saat tertentu.
5. Bentuk/ format laporan yang baru, sesuai dengan kebutuhan saat ini.
6. Mempunyai *layout* tampilan program yang baru, sehingga akan lebih memudahkan pengguna dalam menggunakan perangkat lunak ini, termasuk dalam hal ini adalah penambahan beberapa *short-cut* untuk membantu mempercepat proses pemasukan data.

1.2 Pembatasan Masalah

Agar pengembangan pada penelitian ini tidak menyimpang dari tujuan yang ingin dicapai, maka beberapa pembatasan masalah harus ditetapkan sebagai berikut:

1. Penelitian dilakukan di Kantor Ekonom Keuskupan Bandung, Jl. Jawa no 26 Bandung.
2. Penelitian ini hanya akan membahas konsep perbaikan perangkat lunak keuangan yang lama menjadi yang lebih baik, sesuai dengan desain baru yang diinginkan oleh pihak Kantor Ekonom Keuskupan Bandung.
3. Bahasa pemrograman yang akan digunakan adalah Borland Delphi 7, karena program lama yang ada dibangun menggunakan bahasa pemrograman Delphi 2.0.

1.3 Tujuan Pengembangan

Berdasarkan permasalahan yang dihadapi, maka perumusan tujuan dari pengembangan yang dilakukan dalam penelitian ini adalah :

Mengembangkan suatu perangkat lunak keuangan baru yang dapat memenuhi permintaan pihak Kantor Ekonom Keuskupan Bandung. Perangkat lunak baru ini diharapkan akan dapat bekerja lebih cepat, dan lebih memudahkan penggunanya dalam menggunakan perangkat lunak ini.

1.4 Manfaat Pengembangan

Manfaat dari penelitian yang dilakukan ini adalah membuat sebuah Sistem Informasi Akuntansi baru bagi Kantor Ekonom Keuskupan Bandung dan juga bagi unit-unit aktivitas lain di bawah naungan Keuskupan Bandung, sehingga mereka dapat mencatat dan mengelola keuangan mereka dengan lebih baik (lebih cepat, mudah dan efisien).

1.5 Metodologi Pengembangan

Dalam penelitian ini, dilakukan langkah-langkah yang sistematis untuk dapat melancarkan proses penelitian. Langkah-langkah yang dilakukan penelitian ini adalah sebagai berikut :

1. Identifikasi Masalah.

   Pada tahap pertama ini, dilakukan identifikasi terhadap masalah yang dihadapi oleh Kantor Ekonom Keuskupan Bandung, terutama yang berhubungan performa sistem informasi keuangan yang telah berjalan.

2. Perumusan Masalah.

   Pada tahap ini, dilakukan penguraian semua faktor penyebab yang berhubungan dengan masalah yang ditemukan pada sistem informasi keuangan lama. Pada tahap ini juga dirinci kebutuhan baru dari pihak Kantor Ekonom Keuskupan Bandung. Pada tahap ini juga dilakukan upaya

pembatasan masalah sehingga penelitian yang dilakukan dapat terfokus dan menghindari pembahasan yang terlalu luas.

3. Pengumpulan Data.

   Pada tahap ini akan dilakukan beberapa data teknis dari sistem informasi keuangan lama yang akan diperbaiki. Pada tahap ini, akan dilacak semua kesalahan yang mungkin menimbulkan rangkaian keluhan pada penggunaan sistem informasi ini.

4. Perbaikan Sistem.

   Setelah memperoleh data-data yang diperlukan untuk perbaikan rancangan sistem informasi yang baru, maka mulailah dilakukan tahap-tahap mencari dan mengganti semua kode program yang tidak sesuai dengan serangkaian kode baru yang lebih sesuai. Mengingat jumlah baris program total berjumlah lebih dari 10.000 baris, maka perlu waktu yang agak lama untuk menelaah baris-baris program ini blok demi blok.

5. Analisa dan Pengujian Rancangan

   Pada tahap ini dilakukan serangkaian pengujian pada rancangan sistem informasi yang baru dibuat yang meliputi pengujian tampilan, performa, akurasi perhitungan dan kesesuaian format laporan sesuai dengan panduan dan arahan dari pihak Kantor Ekonom Keuskupan Bandung. Pada tahap ini, dilibatkan juga pengguna dari Kantor Ekonom Keuskupan Bandung untuk memberikan pendapatnya dan juga memeriksa keakuratan hasil akhir dari sistem ini.

6. Kesimpulan dan Saran

   Dari hasil analisis dan pengujian rancangan tersebut, maka dapat dibuat kesimpulan untuk merangkum solusi permasalahan yang ada dan keseluruhan penelitian yang telah dilakukan. Saran dapat diberikan untuk peluang penelitian lebih lanjut guna semakin memperbaiki performansi dan integrasi

sistem informasi keuangan ini dengan sistem yang lain yang ada di Keuskupan Bandung.

1.6 Sistematika Laporan

Laporan Akhir Penelitian ini dibagi menjadi beberapa bab yang saling berkaitan dan ditulis menurut sistematika berikut:

BAB 1 PENDAHULUAN

Bab ini menguraikan gambaran umum pengembangan yang dilakukan terdiri dari latar belakang masalah, identifikasi masalah, pembatasan masalah, perumusan masalah, tujuan pengembangan, manfaat pengembangan dan sistematika penulisan.

BAB 2 LANDASAN TEORI

Bab ini menguraikan teori-teori dan pengertian-pengertian yang ada kaitannya dengan masalah yang ada dan selanjutnya digunakan sebagai dasar pemikiran dalam pemecahan masalah.

BAB 3 PENGEMBANGAN PERANGKAT LUNAK

Bab ini berisi pembahasan tentang pokok-pokok perancangan dan pengembangan perangkat lunak, mulai dari pemahaman permintaan dari pengguna hingga pemrograman perangkat lunak tersebut dengan menggunakan bahasa Borland Delphi.

BAB 4 ANALISIS

Bab ini berisi analisis mengenai perangkat lunak yang telah dikembangkan, apakah sudah sesuai dengan kriteria desain yang diinginkan.

BAB 5 KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan dari pengembangan perangkat lunak yang telah dilakukan dan saran-saran yang perlu diperhatikan agar tujuan dari pengembangan dapat tercapai.

BAB 2

LANDASAN TEORI

Bab kedua ini berisi konsep dasar dan teori-teori yang digunakan dalam penelitian ini. Pembahasan dimulai dengan penjelasan mengenai sistem informasi itu sendiri, perencanaan sistem informasi, analisis sistem dan beberapa metode analisis sistem informasi, metode *Object-Oriented*, *Object-Oriented Analysis*, *Object-Oriented Design, Object-Oriented Programming*, teknologi *client/server*, dan jaringan komputer lokal (*Local Area Network*).

2.1 Sistem Informasi

Ide tentang sistem adalah hal mendasar yang dibicarakan dalam sistem informasi. Sistem informasi juga merupakan suatu sistem yang memiliki karakteristik sistem. Untuk memperjelas bagaimana pemahaman mengenai suatu sistem dapat membantu memahami sistem informasi, berikut ini didefinisikan mengenai sistem, informasi, dan sistem informasi itu sendiri.

2.1.1 Sistem

Steven Alter (1992) mendefinisikan suatu sistem sebagai sekumpulan komponen yang saling berinteraksi dan bekerja sama untuk mencapai tujuan. Sistem menerima input dari lingkungan dan menghasilkan output bagi lingkungannya. Komponen sistem dapat berupa organisasi, orang, atau mesin. Masing-masing komponen dapat diperlakukan sebagai subsistem. Subsistem menerima input, mengolahnya, dan menghasilkan output. Hubungan antar subsistem berupa aliran informasi dan material.

2.1.2 Informasi

Informasi adalah data yang telah diolah sehingga bentuk dan isinya dapat berguna untuk tugas tertentu. Informasi sangat dibutuhkan manajemen sebagai dasar pengambilan keputusan dalam mengelola perusahaan (Alter, 1992).

Karena informasi sangat dibutuhkan dalam proses-proses pengambilan keputusan, maka informasi harus memiliki suatu standar kualitas tertentu. Kualitas sebuah informasi ditentukan oleh tiga hal yaitu keakuratan, ketepatan waktu, dan relevansi. Akurat berarti informasi harus bebas dari kesalahan-kesalahan dan tidak bias atau menyesatkan; akurat juga berarti jelas mencerminkan maksud yang ingin disampaikan. Tepat waktu berarti informasi yang datang pada penerima tidak terlambat. Keterlambatan informasi akan menurunkan nilai informasi tersebut. Relevan berarti informasi tersebut memiliki manfaat bagi penggunanya. (Jogiyanto, 1990)

2.1.3 Sistem Informasi

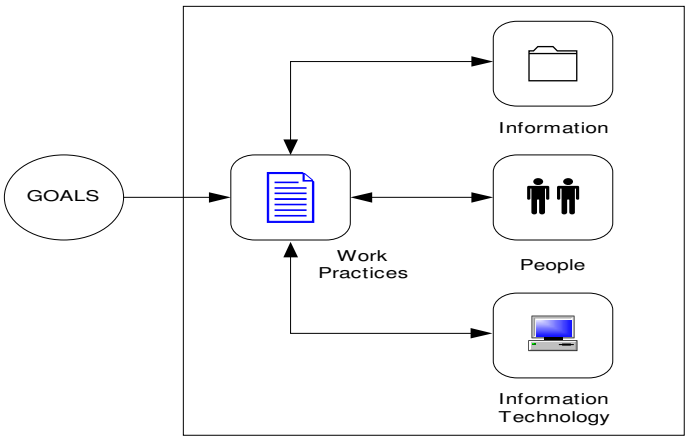
Dari definisi di atas, maka sistem informasi dapat didefinisikan sebagai sebuah kombinasi dari *work practices*, informasi, orang, dan teknologi informasi yang diatur untuk mencapai tujuan suatu organisasi. Tujuan organisasi memiliki peranan vital dalam suatu sistem informasi, yaitu dalam menentukan praktek kerja organisasi tersebut. Tujuan menentukan kriteria untuk memutuskan apakah praktek kerja suatu organisasi harus diubah. (Alter, 1992)

Sistem informasi yang baik bukan hanya berfungsi sebagai alat pencegah terjadinya kesalahan manajemen tetapi juga sebagai alat strategi keunggulan kompetitif perusahaan dalam menjawab tantangan persaingan yang ada. Karena ada syarat kualitas informasi yang harus dipenuhi, maka syarat sistem informasi yang baik adalah juga sistem yang mampu menyediakan informasi yang

berkualitas (akurat, tepat waktu, dan relevan). Sistem informasi yang dibangun sebaiknya sederhana namun tetap lengkap mencakup semua informasi yang dibutuhkan.

Komponen pembentuk suatu sistem informasi terdiri dari :

1. *Work practices*, yaitu metode yang digunakan manusia dan teknologi untuk melakukan suatu pekerjaan. Mencakup prosedur operasi untuk untuk tugas-tugas rutin; lembar kerja dan grafik-grafik untuk koordinasi, komunikasi, membuat keputusan, dan lainnya.
2. Informasi. Suatu sistem informasi dapat terdiri dari data, teks, gambar, dan suara dalam format tertentu. Data adalah sekumpulan fakta, gambar, atau suara yang dapat berguna ataupun tidak untuk suatu tugas tertentu. Sedangkan informasi adalah data yang telah diolah sehingga bentuk dan isinya dapat berguna untuk tugas tertentu. Suatu praktek kerja membutuhkan informasi dan keberadaan informasi menentukan praktek kerja apa saja yang dapat dilakukan.
3. Manusia. Dalam suatu sistem informasi yang tidak terotomasi sepenuhnya, terdapat manusia yang berperan dalam pemasukan, proses, atau penggunaan data. Cara kerja dan karakteristik manusia dalam sistem menentukan pekerjaan apa saja yang dapat dilakukan.
4. Teknologi informasi, mencakup perangkat keras dan lunak yang melakukan tugas-tugas pemrosesan data seperti pengumpulan, pengiriman, penyimpanan, pengambilan, pemrosesan, atau penyajian data. Contohnya adalah komputer, *barcode reader*, perangkat lunak pemrosesan transaksi, dan sebagainya. Teknologi informasi berguna hanya sebagai bagian dari sistem informasi yang terdiri dari *work practices*, orang, dan informasi.



Gambar 2.1 Definisi Sistem Informasi [Alter, 1992]

2.1.4 Tipe Sistem Informasi

Secara umum sistem informasi dapat dibagi menjadi enam tipe. Namun klasifikasi suatu aplikasi sistem informasi seringkali terdiri dari beberapa kategori sistem informasi (*overlap*). Keenam tipe sistem informasi tersebut yaitu :

1. *Transaction Processing System (TPS)*

   Yaitu sistem yang mengumpulkan dan menyimpan data tentang transaksi dan terkadang juga mengontrol keputusan yang dibuat sebagai bagian dari transaksi tersebut. Transaksi adalah kejadian dalam bisnis yang menghasilkan atau mengubah data yang disimpan dalam sistem informasi.

   Terdapat dua tipe TPS, yaitu :

   - *Batch processing*

     Data transaksi dikumpulkan dan disimpan namun tidak dimasukkan ke dalam sistem secara langsung. Pemasukan data dilakukan sesuai jadwal atau pada saat jumlah transaksi telah menumpuk, untuk mengubah *database*.

   - *Real-time processing*

     Transaksi diproses langsung setelah data diperoleh. Tipe pemrosesan ini memerlukan penyimpanan data yang bisa diakses dengan segera.

2. Sistem Informasi Manajemen

   Sistem ini mengubah data dari TPS menjadi informasi bagi pihak manajemen untuk mengatur organisasi, memantau performansi, melakukan koordinasi, dan menyediakan informasi tentang operasional organisasi.

3. *Decision Support System (DSS)*

   Yaitu sistem interaktif yang membantu orang untuk membuat keputusan dan bekerja dalam lingkungan dimana tidak ada orang yang tahu bagaimana tugas harus dilakukan dalam segala kasus. DSS membantu pengambilan keputusan dalam situasi semi terstruktur dan tidak terstruktur, serta menyediakan informasi, model, atau *tool* untuk mengolah data.

4. *Executive Informastion System*

   Yaitu sistem yang menyediakan akses informasi secara fleksibel bagi manajer dan eksekutif untuk memantau hasil operasi dan kondisi umum dari bisnis.

5. *Expert System*

   Sistem ini mendukung pekerjaan intelektual seorang profesional seperti desain, diagnosis, atau evaluasi situasi kompleks yang membutuhkan pengetahuan pakar dalam suatu masalah.

6. *Office Automation System*

   Sistem yang memberikan fasilitas pemrosesan tugas komunikasi dan informasi sehari-hari dalam kantor dan organisasi bisnis. Sistem ini memberikan banyak *tool* seperti pemroses kata, lembar kerja, sistem telepon, dan sebagainya.

2.2 Perencanaan Sistem Informasi

Pembangunan sistem informasi harus direncanakan agar sesuai dengan strategi, tujuan perusahan, dan perencanaan operasionalnya. Perencanaan sistem informasi merupakan bagian dari perencanaan bisnis yang berkaitan dengan pengaturan

sumber daya informasi perusahaan. Perencanaan ini perlu dilakukan untuk memperbaiki ataupun mengganti sistem informasi akibat munculnya permasalahan-permasalahan dalam sistem lama yang tidak dapat beroperasi sesuai dengan yang diharapkan. Pertumbuhan organisasi juga mengakibatkan timbulnya kebutuhan pengembangan teknologi penyediaan informasi yang lebih baik untuk memperoleh kesempatan bisnis. Perencanaan juga dilakukan apabila ada instruksi dari pimpinan ataupun dari luar perusahaan, misalnya peraturan pemerintah. Pendekatan perencanaan sistem informasi yang umum digunakan yaitu *Critical Success Factor* dan *Business System Planning*.

2.2.1 *Critical Success Factor*

Critical Success Factor merupakan suatu pendekatan untuk mengidentifikasi faktor-faktor yang kritis terhadap kesuksesan operasi bisnis. Sistem informasi yang mendukung faktor ini harus mendapatkan prioritas tertinggi dan mampu menyediakan informasi yang dibutuhkan eksekutif. Langkah-langkah pendekatan CSF yaitu :

1. Identifikasi tujuan perusahaan. Tujuan perusahaan digunakan untuk menentukan apakah suatu faktor kritis memang penting bagi perusahaan.
2. Identifikasi CSF dan kebutuhan sistem informasinya. CSF merupakan faktor yang harus dilakukan dengan benar untuk kesuksesan bisnis. CSF dapat diidentifikasikan dari struktur industri, strategi kompetisi perusahaan, posisi perusahaan dalam industri, faktor lingkungan, dan sebagainya. Identifikasi CSF dilakukan untuk perusahaan dan kemudian untuk level dibawahnya.
3. Identifikasi indikator performansi dari faktor kritis. Indikator performansi menunjukkan kemampuan perusahaan dalam mengerjakan faktor kritis.

2.2.2 *Business System Planning*

Business System Planning memberikan gambaran menyeluruh dari informasi dan sistem informasi yang dibutuhkan untuk mendukung operasi bisnis. Gambaran menyeluruh ini disebut juga arsitektur informasi. Dalam pendekatan BSP direncanakan sistem informasi terintegrasi dengan rencana bisnis yang mempu mendukung kebutuhan informasi bisnis jangka pendek dan panjang. BSP memberikan keyakinan tentang arah pengembangan sistem informasi bagi organisasi.

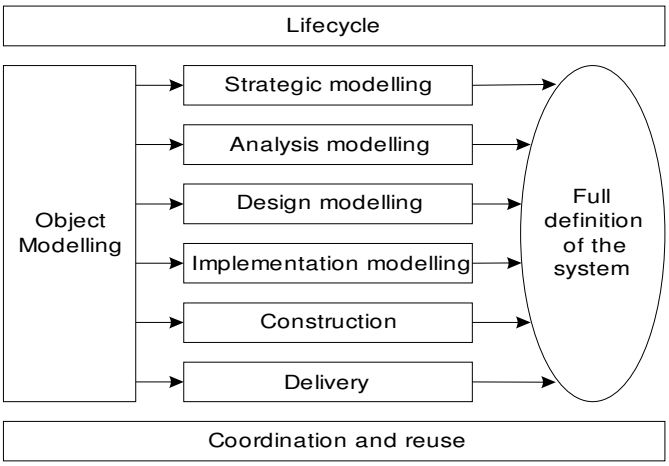
Studi BSP membutuhkan dukungan dan komitmen dari eksekutif. Sistem bisnis secara menyeluruh dan dukungan pengolahan datanya perlu dipahami dengan baik. Proses BSP dibagi menjadi beberapa langkah utama yaitu :

1. Definisikan proses bisnis. Langkah ini membutuhkan pemahaman terhadap fungsi-fungsi dalam perusahaan untuk menentukan proses kuncinya. Proses bisnis didefinisikan dari level perencanaan strategis sampai kontrol kerja.
2. Definisikan data bisnis. Langkah ini mendefinisikan data-data yang digunakan proses bisnis dan mengelompokkannya ke dalam kelas data. Kelas data merupakan kelompok data yang digunakan proses bisnis.
3. Definisikan arsitektur informasi. Langkah ini menggambarkan hubungan antara kelas data dengan proses bisnis dalam matriks proses/kelas data. Kelas data dapat diciptakan (C = created), dibaca (R = read), digunakan (U = usage) atau dihapus (D = delete) oleh suatu proses bisnis.

Untuk memvalidasi dan memperbaiki arsitektur informasi dapat dilakukan wawancara dengan eksekutif. Kemudian ditetapkan prioritas pengembangan sistem informasi berdasarkan kriteria pemilihan tertentu. Sistem informasi yang mendapat prioritas tersebut kemudian dikembangkan.

2.2.3 *Object-Oriented Project Life Cycle*

Untuk suatu pengembangan proyek sistem informasi, *life cycle* diartikan sebagai langkah-langkah atau aktivitas yang harus dilakukan selama proyek berlangsung. Secara sederhana *life cycle* terdiri dari langkah analisis, desain, penulisan program (*coding*), dan pengujian (*testing*). Object Management Group (OMG) memperkenalkan model *OO project life cycle* seperti pada gambar berikut.



Gambar 2.2 *OO Project Life Cycle* [Yourdon, 1994]

- *Object Modelling*

  Keseluruhan aktivitas OO berhubungan dengan objek, baik pada tahap analisis, desain, ataupun pemrograman, dan pada keseluruhan aktivitas *life cycle* ini, konsep abstraksi, enkapsulasi, dan pewarisan memegang peranan besar.

- *Strategic Modelling*

  Tujuan aktivitas ini adalah membangun model unit bisnis yang besar, atau organisasi perusahaan yang terdiri dari beberapa sistem yang berbeda. Pemodelan ini berguna untuk membantu menemukan objek-objek yang relevan dengan sistem; dan dapat membantu menghasilkan sejumlah dokumen dan keputusan pada model perusahaan.

- *Analysis Modelling*

  Konsep *Analysis Modelling* terdiri dari aktivitas mencari dan mendokumentasikan kebutuhan pengguna untuk suatu sistem. Tujuannya adalah untuk membentuk model yang formal dan detail dari kebutuhan yang diharapkan penguna suatu sistem, dan pemahaman yang mendalam dari sistem yang dianalisis.

- *Design Modelling*

  Tujuan dari desain adalah menentukan pandangan eksternal dari sekumpulan tipe objek. Hasil dari aktivitas desain adalah spesifikasi dari tipe objek (khususnya tipe objek yang mengambarkan *user interface* dan komponen database dari aplikasi), spesifikasi operasi dan tatap muka, penggabungan tipe objek ke dalam model desain, dan desain untuk memenuhi permintaan kualitas. Pada tahap ini kemampuan dan konstrain sistem operasi, bahasa pemrograman, *database management system*, dan sebagainya mulai dipertimbangkan.

- *Implementation Modelling*

  Implementation Modelling berhubungan dengan distribusi objek dalam komponen perangkat keras/lunak yang berbeda dalam jaringan *client/server*. Pada tahap ini harus ditentukan strategi untuk menentukan bagaimana objek dan kelas didistribusikan ke berbagai komponen operasional dan juga kapan objek akan saling digabungkan.

2.3 Analisis Sistem

Analisis suatu sistem perlu dilakukan sebelum melakukan perancangan sistem. Analisis dilakukan untuk memperlajari dan memahami karakteristik sistem yang akan dirancang sehingga hasil perancangan akan sesuai dengan kebutuhan pengguna sistem tersebut.

2.3.1 Definisi Analisis Sistem

DeMarco (1978) mendefinisikan analisis sebagai studi suatu permasalahan, sebelum melakukan suatu tindakan. Sedangkan Peter Coad dan Edward Yourdon (1991) mendefinisikan analisis sebagai studi suatu sumber permasalahan, untuk menspesifikasikan perilaku sistem yang dapat diamati; pernyataan yang lengkap dan konsisten tentang apa yang dibutuhkan sistem. Kebutuhan sistem terdiri dari operasi fungsional dan karakteristik operasional, seperti kemudahan penggunaan, kehandalan, dan performansi. Kebutuhan sistem juga mencakup *interface* dari perangkat lunak, lingkungan kerja yang harus diakomodasi oleh perangkat lunak, dan konstrain-konstrain dalam aplikasi desain.

2.3.2 Metode Analisis

Terdapat empat pendekatan untuk melakukan analisis sistem yang digunakan sebagai alat bantu untuk memformulasikan kebutuhan, yaitu :

1. *Functional Decomposition*

   Strategi yang digunakan dalam *Functional Decomposition* terdiri dari penentuan langkah dan sub-langkah pemrosesan dalam suatu sistem. Fokusnya adalah dalam proses apa saja yang diperlukan untuk suatu sistem baru, kemudian menentukan proses dan *interface* fungsional. Dengan metode ini, analis sulit menentukan apakah kebutuhan sistem sudah tepat menunjukkan kebutuhan sistem yang sebenarnya. Permasalahan juga terdapat dalam penentuan fungsi, sub-fungsi, dan pengetahuan terhadap potensi perubahan dalam sistem.

2. *Data Flow Approach*

   Pendekatan ini disebut juga "*structured analysis*". Analis memetakan dunia nyata ke dalam aliran data dan *bubble* (*data flow diagram*). Analis harus

mengidentifikasikan kejadian-kejadian dalam sistem yang harus direspon sistem dan langkah-langkah pemrosesannya, mengikuti aliran data, dan menspesifikasikannya dalam analisis. Setiap kejadian ditandai dengan satu *bubble*. Setiap *bubble* memiliki aliran input dan output. Penyimpanan data ditempatkan diantara *bubble* yang memerlukan komunikasi dengan *database*. Beberapa *bubble* digabungkan menjadi *bubble* dengan level yang lebih tinggi. Level tertinggi dari aliran data dan *bubble* ini disebut konteks diagram. Permasalahan yang timbul antara lain ukuran kamus data yang besar, kurangnya penekanan pada struktur penyimpanan data, dan sulitnya beralih dari analisis ke desain.

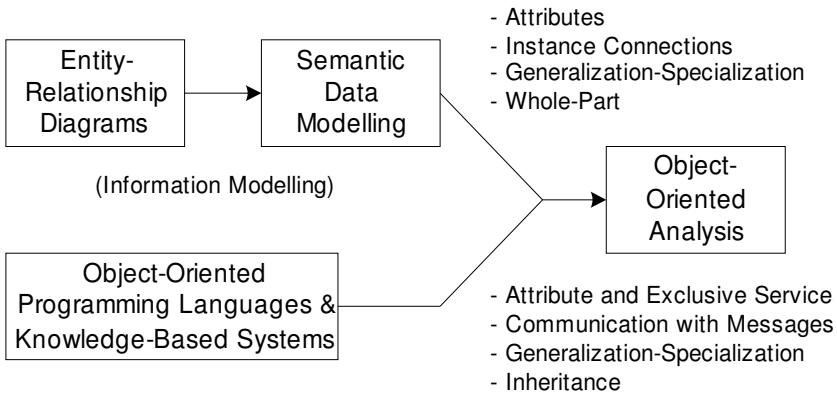
3. *Information Modeling*

Metode ini menggunakan *entity-relationship diagram* dan berkembang menjadi *semantic data model*. Istilah objek sudah mulai digunakan sebagai representasi kejadian entitas dalam dunia nyata. Strategi lama metode ini adalah dengan mendefinisikan atribut-atribut yang dibutuhkan sistem, kemudian satu per satu atribut ditempatkan dalam objek. Tambahkan *relationship* dan definisikan *supertype/sub-type* dan hubungan antar objek. Kemudian normalisasi untuk mengurangi redundansi data dalam implementasi *database*. Strategi yang lebih baru adalah dengan mendefinisikan obbjek dalam dunia nyata dan mendeskripsikannya dengan atribut. Metode ini menggambarkan data dengan baik, namun lemah dalam menggambarkan proses-proses yang terjadi dalam sistem.

4. *Object-oriented*

Metode *Object-oriented* merupakan metode analisis sistem informasi yang dibangun berdasarkan konsep-konsep dari *Information Modelling*, *Object-Oriented Programming*, dan *Knowledges-Based Systems*. Prinsip dasar metode

Object-Oriented adalah memandang sistem sebagai kesatuan atau kumpulan objek.



Gambar 2.3 Penggabungan Disiplin Ilmu [Coad, 1991]

Metode *Object-Oriented* unggul dalam kemampuannya membuat program aplikasi komputer secara modular. Ketika sebuah sistem informasi yang sudah ada akan dikembangkan, modifikasi yang dilakukan lebih mudah dibandingkan dengan metode lainnya, dengan kata lain metode ini memiliki sifat fleksibilitas yang tinggi. Pembahasan lebih lanjut tentang metode ini pada sub-bab berikutnya.

2.4 Metode *Object-Oriented*

Metode *Object-Oriented* merupakan salah satu metode pengembangan sistem informasi yang masih belum banyak digunakan. Metode ini didasarkan pada konsep : objek dan atribut, *whole* dan *part*, kelas dan anggota. Pemrograman *Object-Oriented* muncul pada tahun 1960 dengan bahasa pemrograman SIMULA. Namun metode ini tidak berkembang karena pada tahun 1970-an banyak digunakan metode *functional decomposition*. Baru pada tahun 1990-an metode *Object-Oriented* mulai berkembang. Kebutuhan akan suatu sistem yang lebih

bersifat *user-oriented* dan *data-oriented* menjadi prioritas utama dengan pendekatan *Object-Oriented.*

2.4.1 Definisi *Object-Oriented*

Menurut Edward Yourdon (1994), suatu sistem yang dibangun dengan metode *Object-Oriented* merupakan sistem yang komponennya memiliki sekumpulan data dan fungsi yang tersembunyi, yang dapat mewarisi atribut dan perilaku dari komponen lainnya, dan yang komponennya saling berkomunikasi lewat pesan tertentu. Suatu persamaan yang dapat digunakan untuk mengenali pendekatan *Object-Oriented* adalah :

```
Object-Oriented = Kelas dan Objek
                + Pewarisan
                + Komunikasi dengan Pesan
```

2.4.2 Prinsip-prinsip Metode *Object-Oriented*

Prinsip-prinsip yang digunakan dalam metode *Object-Oriented* untuk mengatasi kompleksitas pembangunan suatu sistem yaitu :

1. **Abstraksi**, adalah mekanisme yang merepresentasikan suatu realitas kompleks dalam model yang disederhanakan; prinsip mengabaikan aspek-aspek dari suatu subjek yang tidak relevan dengan tujuan pembangunan sistem dengan maksud untuk lebih konsentrasi.
2. **Enkapsulasi** (penyembunyian informasi), mekanisme untuk menyembunyikan implementasi suatu objek, sehingga komponen lain dari sistem tidak dapat memperoleh data yang tersimpan dalam objek tersebut.
3. **Pewarisan** (*Inheritance*), mekanisme suatu objek untuk menggunakan seluruh atau sebagian definisi dari objek lain sebagai bagian dari definisi objek itu sendiri. Suatu sistem berorientasi objek akan memiliki hirarki objek, dimana

objek subordinat akan mewarisi atribut dan perilaku fungsional dari level kelas diatasnya.

4. **Polimorfisme**, prinsip perbedaan reaksi yang ditunjukkan antara objek meskipun objek-objek itu menerima pesan yang sama.
5. **Asosiasi**, persatuan atau hubungan antar ide-ide. Prinsip asosiasi digunakan untuk menggabungkan beberapa hal yang terjadi pada saat yang bersamaan atau dalam kondisi yang serupa.
6. **Komunikasi** dengan pesan. Pesan adalah komunikasi lisan maupun tulisan yang dikirim oleh dan kepada seseorang.
7. **Metode organisasi**, digunakan untuk memahami permasalahan dengan lebih mendalam dan terstruktur, yaitu dengan "objek dan atribut", "*whole and parts*", "kelas, anggota, dan perbedaan diantaranya".
8. **Skala**, prinsip yang menggunakan *whole-part* untuk memandu pembaca ke dalam model yang lebih besar.
9. **Kategori perilaku**, sistem memiliki sifat dinamis/aktif dalam menanggapi perubahan, diantaranya sifat respon kejadian secara langsung, perubahan secara kontinu, respon pada kejadian-kejadian serupa.

2.4.3 Keunggulan Teknologi *Object-Oriented*

Teknologi *Object-Oriented* semakin dikenal dan berkembang karena memiliki beberapa keunggulan, antara lain:

1. Produktivitas. Sistem yang dikembangkan dengan teknologi OO dapat meningkatkan produktivitas tim karena kemudahan dalam pemrograman dan penggunaan kembali (*reuse*) kelas dan objek dari perpustakaan komponen.
2. Waktu pengembangan yang lebih cepat. Dengan meningkatnya produktivitas, waktu yang dibutuhkan untuk menyelesaikan suatu proyek akan lebih cepat, terutama dengan prinsip *reuse* dan mekanisme *inheritance*.

3. Kualitas dan kemudahan perawatan. Dengan menggunakan teknologi OO, sistem yang dikembangkan akan lebih terbebas dari cacat (*bug*) sehingga kualitas pengembangan sistem meningkat. Dan juga sistem dibangun berdasarkan jaringan objek yang saling berhubungan, sehingga suatu objek dapat diubah tanpa mengganggu objek lainnya.

2.4.4 Aliran Metode *Object-Oriented*

Metode *Object-Oriented* banyak dikembangkan oleh para pakar sistem informasi sehingga banyak aliran *Object-Oriented* yang muncul. Misalnya metode yang dikembangkan oleh Peter Coad dan Edward Yourdon, Grady Booch, Sally Shlaer dan Stephen J. Mellor, Rumbaugh, Ivar Jacobson, dan lainnya. Masing-masing memiliki notasi dan komponen yang berbeda untuk menggambarkan sistem nyata. Namun demikian konsep dasar dari setiap aliran tersebut tetap sama yaitu *Object-Oriented*.

Metode *Object-Oriented* yang dikembangkan Peter Coad dan Edward Yourdon dibagi menjadi tiga bagian yaitu *Object-Oriented Analysis, Object-Oriented Design,* dan *Object-Oriented Programming*. Masing-masing bagian mewakili fase dalam perancangan sistem informasi.

2.5 *Object-Oriented Analysis* (OOA)

Pada proses analisis dibangun suatu model *Object-Oriented* yang menggambarkan kebutuhan para pengguna dalam sebuah sistem. Model yang dikembangkan Peter Coad dan Edward Yourdon berupa model tunggal berorientasi objek yang ditampilkan dalam sejumlah lapisan. Model ini sesuai dikembangkan untuk sistem dengan karakteristik berorientasi data karena akan menghindari detil yang berlebihan bagi pengguna. Kelebihan lainnya adalah hanya perlu memahami satu

jenis notasi diagram sehingga mempermudah pembacaan diagram. Model ini ditampilkan dalam lima lapisan :

1. Lapisan subjek
2. Lapisan kelas-&-objek
3. Lapisan struktur
4. Lapisan atribut
5. Lapisan layanan

Masing-masing lapisan akan memberikan tingkat analisis yang lebih detil. Keuntungan menggunakan OOA antara lain :

1. Dapat mengatasi berbagai macam permasalahan. OOA memberikan penekanan lebih pada pemahaman terhadap sumber permasalahan.
2. Meningkatkan interaksi analis dengan ahli dalam permasalahan. OOA mengatur analisis dan spesifikasi sistem menggunakan metode organisasi yang menggambarkan cara orang berpikir.
3. Meningkatkan konsistensi internal dari hasil analisis. OOA mengurangi jarak antar aktivitas analisis, dengan memperlakukan atribut dan layanan (*services*) sebagai suatu kesatuan.
4. Menunjukkan sifat umum. OOA menggunakan pewarisan untuk menunjukkan sifat umum dari atribut dan layanan.
5. Membangun spesifikasi yang mudah menyesuaikan dengan perubahan. OOA memberikan stabilitas dalam perubahan kebutuhan sistem.
6. Dapat menggunakan kembali hasil analisis. OOA mengatur hasil analisis berdasarkan permasalahan untuk digunakan kembali saat ini dan masa mendatang.

7. Memberikan representasi yang konsisten untuk analisis dan desain. OOA membangun representasi yang kontinu untuk memperluas hasil analisis ke desain.

Aktivitas utama dalam proses *Object-Oriented Analysis* adalah (Coad dan Yourdon, 1991) :

1. Menemukan kelas dan objek.
2. Mengidentifikasi struktur.
3. Mengidentifikasi subjek.
4. Mengidentifikasi atribut.
5. Mengidentifikasi layanan.

Secara rinci, 5 aktivitas utama OOA dapat dilihat pada pembahasan berikut.

2.5.1 Kelas dan Objek

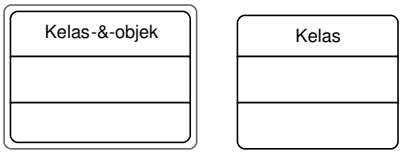
Objek adalah abstraksi sesuatu dalam suatu permasalahan, yang merefleksikan kemampuan sistem untuk menyimpan informasi tentang sistem tersebut, berinteraksi dengannya; tempat penyembunyian nilai atribut dan layanan. Objek biasa disebut juga instansi.

Kelas adalah sebutan untuk satu objek atau lebih yang memiliki sekumpulan atribut dan layanan yang sama, termasuk deskripsi bagaimana untuk menciptakan objek baru dalam kelas tersebut; sekumpulan objek yang dapat dideskripsikan dengan atribut dan layanan yang sama.

Kelas-dan-objek berarti sebuah kelas dan objek dalam kelas itu sendiri.

Tujuan mengidentifikasi kelas-&-objek adalah berusaha agar representasi teknikal dari suatu sistem lebih mendekati pandangan konseptual dari dunia nyata.

Notasi dari kelas-&-objek dan kelas adalah seperti pada gambar berikut.



Gambar 2.4 Notasi Kelas-&-Objek dan Kelas

Suatu objek selalu berada dalam sebuah kelas dan digambarkan dengan notasi kelas-&-objek. Sedangkan notasi kelas digunakan untuk menggambarkan generalisasi dari permasalahan, dimana objeknya berada pada spesialisasi kelas tersebut. Nama kelas berupa sebuah kata benda atau frase kata sifat dan kata benda. Nama kelas-&-objek harus menunjukkan suatu objek dalam kelas. Pilihan nama kelas-&-objek diusahakan menggunakan kosakata yang umum digunakan.

Kelas-&-objek dapat dicari dengan mengamati sistem yang dianalisis, yaitu dengan observasi langsung ke lapangan, mendengarkan dari ahli permasalahan, memeriksa hasil OOA sebelumnya yang serupa (jika ada), memeriksa sistem lain yang memiliki permasalahan serupa, dan banyak membaca dokumen-dokumen dari sistem dan literatur, serta melakukan prototipe. Untuk menemukan kelas-&-objek yang potensial, dapat dicari pada :

- Struktur yang ada dalam sistem.
- Sistem lain yang berinteraksi dengan sistem yang dianalisis.
- Peralatan yang berinteraksi dengan sistem.
- Sesuatu atau kejadian yang harus diingat.
- Peranan yang ada dalam sistem, termasuk yang berinteraksi langsung dan tidak langsung dengan sistem.
- Prosedur operasional dalam sistem yang dilakukan terus menerus.

- Fasilitas fisik dalam sistem.
- Unit organisasi dalam sistem.

Setelah menemukan kelas-&-objek potensial dari langkah diatas, banyak kelas-&-objek yang akan diperoleh, namun mungkin tidak semuanya termasuk dalam model analisis. Perlu dilakukan pertimbangan penentuan kelas-&-objek yang dapat mewakili sistem sehingga sesuai dengan sistem yang dianalisis, yaitu dengan kriteria :

- Kelas-&-objek mengandung informasi untuk diingat.
- Kelas-&-objek melakukan proses dalam sistem.
- Biasanya memiliki beberapa atribut.
- Biasanya lebih dari satu objek dalam suatu kelas.
- Kelas-&-objek mempunyai hubungan dengan sistem yang dianalisis.
- Tidak berasal dari hasil pelaporan akhir.

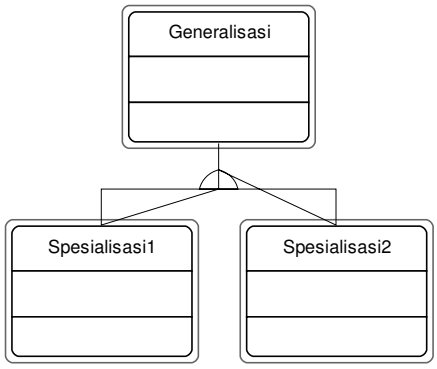
2.5.2 Struktur

Struktur adalah ekspresi dari kompleksitas permasalahan, relevan dengan tanggung jawab sistem. Tujuan identifikasi struktur adalah untuk menganalisis kompleksitas kelas-&-objek, menemukan kelas-&-objek tambahan yang mungkin terlupa, dan mengaplikasikan sifat pewariswan sehingga atribut dan layanan yang umum dapat diidentifikasikan, dispesifikasikan, dan dispesialisasikan dengan tepat.

Terdapat dua jenis struktur yang digunakan OOA sebagai metode organisasi yaitu struktur *Generalization-Specialization* (*Gen-Spec*) dan *Whole-Part*. Suatu struktur kelas-&-objek dapat memiliki kedua struktur ini.

- Struktur *Gen-Spec*

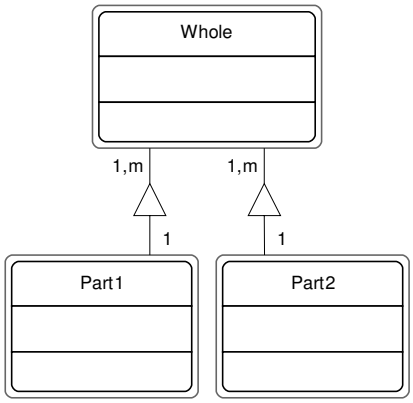
  Struktur *Gen-Spec* menunjukkan hubungan umum (*general*) dan khusus (*special*), atau hubungan struktur "adalah sejenis". Kelas umum diletakkan di bagian atas dari struktur, sedangkan kelas khusus di bagian bawah. Hubungan antar kelas ditunjukkan dengan garis lurus yang saling menghubungkan kelas dalam struktur (bukan objek). Struktur ini menghasilkan suatu hirarki kelas.



Gambar 2.5 Notasi Struktur *Gen-Spec*

- Struktur *Whole-Part*

  Struktur *Whole-Part* menunjukkan hubungan "memiliki". Suatu objek *whole* "memiliki" satu atau beberapa objek *part* (bukan hubungan antar kelas). Ujung garis hubungan struktur ini dilengkapi angka ataupun *range* yang menunjukkan jumlah komponen penyusunan *whole* dengan *part*. Struktur *Whole-Part* dapat dicari dengan mempertimbangkan variasi *assembly-parts, container-contents, collection-members*.

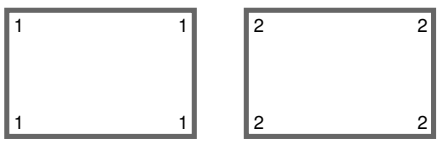


Gambar 2.6 Notasi Struktur *Whole-Part*

Setiap garis notasi struktur *Whole-Part* ditandai dengan suatu range yang menandakan jumlah *part* yang dimiliki oleh suatu *whole.* Dari gambar diatas ditunjukkan bahwa suatu objek *Whole* memiliki paling sedikit satu objek *Part1* dan paling banyak m objek *Part1*. Sedangkan suatu objek *Part1* dimilki oleh satu objek *Whole*.

2.5.3 Subjek

Subjek adalah mekanisme yang membimbing pembaca (analis, manajer, klien) memasuki model yang besar dan kompleks. Subjek berguna untuk mengorganisasi paket-paket kerja pada proyek besar. Tujuan identifikasi subjek adalah untuk memfasilitasi komunikasi, mencegah penggunaan informasi berlebihan, dan membimbing pembaca ke bagian-bagian yang berbeda dalam model. Subjek membagi permasalahan ke dalam beberapa sub-permasalahan. Notasi dari subjek dapat dilihat pada gambar sebagai berikut.



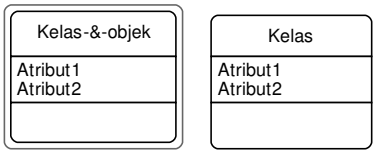
Gambar 2.7 Notasi Subjek

Penentuan subjek dilakukan dengan terlebih dahulu mempromosikan semua kelas teratas dari struktur yang ada dan juga kelas-&-objek yang tidak memiliki struktur, juga memeriksa hasil OOA sebelumnya dari permasalahan yang serupa. Kemudian tentukan subjek dari kandidat subjek sebelumnya yang memiliki ketergantungan minimal dan interaksi minimal diantara subjek. Ketergantungan digambarkan dengan adanya struktur dan *Instance Connection*, sedangkan interaksi digambarkan dengan adanya *Message Connection*.

Notasi subjek digambarkan dengan garis persegi dengan nomor didalamnya. Sebuah kelas-&-objek dapat berada dalam lebih dari satu subjek. Sebuah subjek dapat memiliki subjek didalamnya untuk menunjukkan tingkatan permasalahan dalam model yang besar. Tetapi dalam suatu proyek yang sangat kecil, lapisan subjek ini bisa saja tidak diberikan. Tetapi dalam proyek yang kompleks, subjek dibutuhkan untuk membagi permasalahan ke dalam beberapa sub-permasalahan.

2.5.4 Atribut

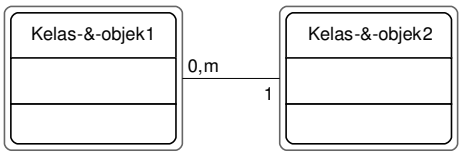
Atribut adalah beberapa data (status informasi) dimana setiap objek dalam sebuah kelas memiliki nilainya masing-masing. Tujuan mendefinisikan atribut adalah untuk menambah detil abstraksi kelas-&-objek dan struktur mode; mendeskripsikan nilai (*state*) yang terdapat dalam objek, yang kemudian akan dimanipulasi oleh layanan dari objek tersebut. Notasi atribut dapat dilihat pada gambar berikut.



Gambar 2.8 Notasi Atribut

Identifikasi objek dilakukan dengan mencari apa saja dari objek yang perlu diketahui, juga memeriksa hasil OOA sebelumnya. Gunakan konsep atomik dimana atribut memiliki nilai tunggal (misalnya nomor KTP) atau beberapa nilai yang digabung menjadi satu kelompok (misalnya atribut alamat yang terbentuk dari jalan, kota, dan kode pos). Tujuannya adalah untuk mendapatkan model yang lebih sederhana dengan jumlah atribut yang minimal. Setiap objek membutuhkan pengenal yang unik dan tidak berubah-ubah. Kemudian tempatkan atribut pada kelas-&-objek yang sesuai. Untuk struktur *Gen-Spec*, tempatkan atribut umum pada kelas teratas dari struktur dimana atribut tersebut masih dapat berlaku untuk semua spesialisasinya.

Selanjutnya melakukan identifikasi *Instance Connection* untuk menggambarkan hubungan antar objek yang belum terjelaskan dengan struktur *Gen-Spec* dan *Whole-part*. ***Instance Connection*** adalah model permasalahan yang menunjukkan keperluan suatu objek dengan objek lainnya, untuk melaksanakan tanggung jawabnya. Hubungan dengan *Instance Connection* ditunjukkan dengan garis yang menghubungkan objek-objek secara individu diantara dua kelas-&-objek. Pada garis *Instance Connection* juga menunjukkan angka atau *range* yang menunjukkan jumlah objek pasangannya. *Instance Connection* digambarkan sebagai berikut.

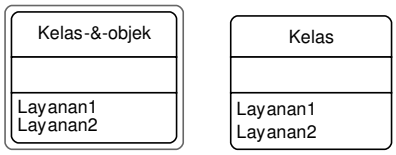


Gambar 2.9 Notasi *Instance Connection*

2.5.5 Layanan

Layanan (*services*) didefinisikan sebagai perilaku spesifik yang harus dilakukan suatu objek. Tujuan mendefinisikan layanan objek adalah menambah detil

abstraksi sesuatu yang dimodelkan, mengindikasikan perilaku apa yang akan dilakukan objek dalam suatu kelas. Notasi atribut dapat dilihat pada gambar berikut.

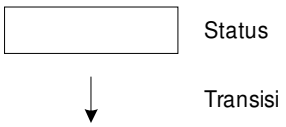


Gambar 2.10 Notasi Layanan

Aktivitas yang dilakukan dalam mendefinisikan layanan adalah :

- Mengidentifikasikan status objek (*Object States*)

  Status suatu objek dinyatakan dengan nilai atributnya. Perubahan nilai atribut menunjukkan perubahan *state*. Untuk mengidentifikasi-kan status objek dilakukan dengan memeriksa semua kemungkinan nilai atribut dan menentukan apakah sistem memiliki perilaku yang berbeda untuk nilai potensial tersebut. Diagram StatusObjek (*ObjectState Diagram*) dapat digunakan untuk menggambarkan seluruh kemungkinan status objek dan transisi suatu status ke status lain.



Gambar 2.11 Notasi Diagram ObjekStatus

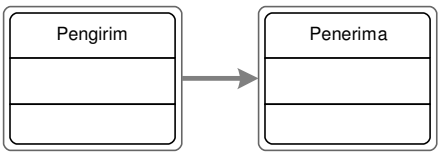
- Mengidentifikasikan layanan yang dibutuhkan

  Layanan pada setiap objek dapat dibagi dua golongan yaitu layanan beralgoritma sederhana dan layanan beralgoritma kompleks. Layanan dengan algoritma sederhana biasanya diperlakukan sebagai layanan implisit dimana layanan tersebut tidak ditampilkan dalam lapisan layanan OOA. Layanan

implisit ditulis dengan menggunakan huruf kecil. Layanan yang termasuk kelompok ini antara lain *create* (menciptakan objek baru dalam kelas), *connect* (menghubungkan suatu objek dengan objek lainnya), *access* (membaca atau mengubah nilai atribut objek), dan *release* (melepas hubungan atau menghapus objek). Sedangkan layanan beralgoritma kompleks dibagi dua kategori yaitu *Calculate* (menghitung hasil dari nilai atribut suatu objek) dan *Monitor* (memantau peralatan atau sistem luar).

- Mengidentifikasikan *Message Connection*

  Message Connection memodelkan ketergantungan antar objek dalam suatu pemrosesan, menandakan adanya kebutuhan layanan untuk memenuhi tanggung jawab objek. *Message Connection* menghubungkan objek dengan objek (atau dengan kelas untuk membentuk objek baru) dimana pengirim mengirimkan pesan, penerima menerima pesan, penerima melakukan pemrosesan dan mengembalikan hasil kepada pengirim. Proses yang diperlukan diberi nama pada spesifikasi layanan pengirim, dan juga didefinisikan pada spesifikasi layanan penerima.

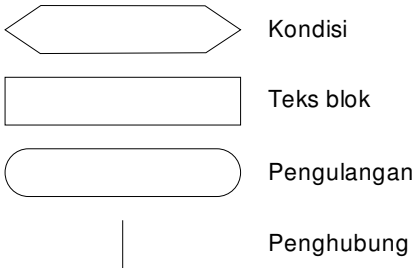


Gambar 2.12 Notasi *Message Connection*

- Menjabarkan layanan

  Setiap layanan dalam kelas-&-objek dijabarkan dengan menggunakan *Service Chart*, terutama untuk layanan beralgoritma kompleks. Notasi yang digunakan dalam *Service Chart* mendukung penerapan prinsip abstraksi prosedural

secara sistematis, dalam lingkup terbatas untuk satu layanan. Notasi *Service Chart* digambarkan sebagai berikut.



Gambar 2.13 Notasi *Service Chart*

2.6 *Object-Oriented Design* (OOD)

Object-Oriented Design adalah tahap lanjutan dari *Object-Oriented Analysis* dimana aktivitas ini memasuki bagian pertengahan dari siklus pembangunan sistem informasi, yaitu aktivitas desain. Tahap desain sistem ini berhubungan dengan bagaimana kebutuhan sistem akan diimplementasikan. Notasi yang digunakan Coad-Yourdon untuk *Object-Oriented Design* sama dengan notasi pada *Object-Oriented Analysis*.

Alasan utama dari identifikasi kelas-&-objek dalam OOD adalah untuk menyesuaikan representasi teknis suatu sistem sedekat mungkin dengan pandangan konseptual dari permasalahan dan implementasinya. Dalam OOD, desainer merancang bagaimana memindahkan hasil analisis ke implementasi perangkat keras/lunak. Sasaran dari OOD adalah meningkatkan produktivitas pembangunan sistem, meningkatkan kualitas sistem, dan meningkatkan kemudahan perawatan. OOD memfokuskan pada aktivitas desain perangkat lunak dengan demikian dapat mengurangi waktu yang dibutuhkan untuk tes dan perbaikan kesalahan. Peningkatan kualitas dilakukan dengan proses analisis dan desain perangkat lunak pada tahap awal pengembangan sehingga dapat mengurangi tingkat *error*.

Model *Object-Oriented Design* terdiri dari empat komponen, yaitu : *Problem Domain Component, Human Interaction Component, Task Management Component,* dan *Data Management Component*. Dengan demikian terdapat empat aktivitas yang dilakukan dalam OOD. Namun aktivitas tersebut bukanlah suatu urutan langkah proses. Aktivitas tersebut adalah :

2.6.1 Mendesain *Problem Domain Component* (PDC)

Hasil analisis merupakan bagian integral dari model multikomponen OOD dimana hasil OOA langsung ditempatkan pada PDC. Peningkatan dan penambahan dalam PDC dilakukan untuk mengorganisasi desain dan pemrograman agar sedekat mungkin dengan permasalahan. Dalam komponen ini kemungkinan besar dilakukan kombinasi dan pembagian kelas, struktur, atribut, dan layanan untuk pertimbangan desain dan kebutuhan implementasi.

Kriteria modifikasi hasil OOA adalah penggunaan kelas dan kode desain, pengelompokan kelas yang memiliki kemiripan, pengakomodasian level pewarisan dari perangkat lunak yang digunakan, peningkatan performansi, pendukungan manajemen penyimpanan, dan penambahan tingkat detil.

2.6.2 Mendesain *Human Interaction Component* (HIC)

Dalam OOA, komponen ini dianalisis dengan menspesifikasikan atribut dan layanan sehingga kebutuhan pengguna dapat dideskripsikan dengan jelas. Sedangkan dalam OOD, hasil analisis ditambahkan HIC sehingga menghasilkan desain *human interaction* dan detail dari interaksi tersebut. Desain HIC juga mencakup desain format untuk tampilan dan laporan. Komponen ini mengidentifikasikan bagaimana manusia akan memerintah sistem dan bagaimana sistem akan menampilkan informasi kepada penggunanya.

Desain *interface* dapat mempengaruhi emosi dari pengguna. Desain yang baik dan sesuai yang diinginkan pengguna dipercayai dapat membantu pengguna menyelesaikan pekerjaannya dan membuat pengguna merasa menikmati pekerjaannya.

Langkah-langkah dalam perancangan HIC adalah :

- Mengklasifikasikan pengguna dan menjabarkan skenario kerjanya.
- Mendesain hirarki perintah.
- Mendesain detil dari interaksi-interaksi yang ada.
- Membuat prototipe.
- Mendesain kelas HIC.

2.6.3 Mendesain *Task Management Component* (TMC)

Task atau proses merupakan serangkaian aktivitas. Pengerjaan sejumlah proses secara bersamaan biasa disebut *multitasking*. Sejumlah *task* akan menambah kompleksitas dalam proses desain. Pengaturan aktivitas perlu dilakukan untuk menghasilkan desain dan kode yang sederhana untuk perilaku konkuren.

Perancangan TMC perlu memperhatikan tingkat kompleksitas yang ditimbulkannya. Untuk bebrapa kondisi, perancangan TMC akan sangat membantu. Tetapi untuk beberapa kondisi lain, perancangan TMC hanya akan menyebabkan sistem terlihat rumit dan semrawut. TMC sangat berguna untuk sistem yang harus melakukan sejumlah prosedur tertentu yang berurutan. Namun untuk sistem yang hanya memiliki prosedur-prosedur singkat dan sederhana, perancangan TMC sebaiknya tidak perlu dilakukan. Beberapa langkah dalam mendesain TMC adalah :

- Mengidentifikasi prosedur yang dikendalikan oleh peristiwa (*event-driven task).*

- Mengidentifikasi prosedur yang dikendalikan oleh waktu (*clock-driven task).*
- Mengidentifikasi prosedur yang memiliki prioritas tinggi dan prosedur yang kritis terhadap jalannya sistem.
- Menguji setiap *task* apakah dibutuhkan atau masih bisa disederhanakan.
- Menjabarkan spesifikasi dari setiap *task* seperti nama, deskripsi, bagaimana pengkoordinasiannya, bagaimana cara komunikasinya.

2.6.4 Mendesain *Data Management Component* (DMC)

Desain DMC menghasilkan infrastruktur untuk penyimpanan dan pengambilan objek dari sistem manajemen data. Komponen ini mengisolasi pengaruh skema manajemen data, baik sistem manajemen data *flat file, relational, object-oriented*, atau yang lainnya. Tiga pendekatan yang sering digunakan dalam manajemen data adalah:

- *Flat file management*

  Pendekatan ini dilakukan dengan penanganan file dan kemampuan mengurutkan yang sederhana, menggunakan tabel umum yang mencakup semua data yang ada. Manajemen data ini memiliki kelemahan dalam penulisan data yang tidak efisien. Seringkali data yang sama dimasukkan berkali-kali. Kelemahan ini semakin terasa pada sistem yang kompleks. Oleh karena itu, pendekatan *flat file management* sudah jarang digunakan.

- *Relational database management system*

  Sistem manajemen data ini mengatur data dalam beberapa tabel. Setiap tabel memiliki nama dan terdiri dari beberapa kolom yang memiliki nama. Setiap baris menunjukkan satu set nilai dalam tabel. Nama tabel, nama kolom, dan batasan setiap kolom didefinisikan dalam skema *database*. Setiap baris harus memiliki pengenal yang unik. Tabel dan kolomnya diatur untuk mengurangi

redundansi data dan mengurangi langkah-langkah untuk memodifikasi data secara konsisten. Hal ini dilakukan dengan normalisasi tabel.

- *Object-oriented database management system* (OO-DBMS)

  Sistem manajemen data ini menganut prinsip-prinsip dalam metode *object-oriented*. Metode ini menyimpan nilai-nilai dalam suatu objek, beserta *internal identifier*-nya. Saat objek diambil dari penyimpanan, objek tersebut sama dengan objek yang ada sebelumnya sebelumnya. Produk komersil OO-DBMS masih terbatas dan harganya relatif tinggi serta belum didukung dengan perkembangan perangkat lunak pembangun aplikasi *database*-nya.

Desain DMC mencakup perancangan layout data dan perancangan layanan interaksi yang berhubungan dengan *database* tersebut. Perancangan layout data dilakukan berdasarkan salah satu pendekatan sistem manajemen data diatas. Sedangkan layanan yang dimaksud adalah bagaimana suatu objek akan disimpan dalam *database*. Layanan tersebut diperlakukan sebagai layanan implisit.

Setelah tahapan OOD selesai, seluruh dokumentasi hasil analisis dan desain telah siap untuk digunakan dalam pembuatan program aplikasi menggunakan *Object-Oriented Programming Language*.

2.7 *Object-Oriented Programming* (OOP)

OOP adalah tahapan pembuatan program aplikasi komputer berbasis *Object-Oriented*. Bahasa pemrograman yang menggunakan metode OOP sebagai prinsip pembangunannya disebut *Object-Oriented Programming Language (OOPL)*. Contoh OOPL yaitu C++, Object Pascal, SmallTalk, Eiffel, Ada, Delphi, Visual Basic, dan sebagainya.

OOPL sangatlah berguna untuk mendukung konstruksi OOA dan OOD. OOPL yang dipilih harus memiliki fitur yang dapat mendukung konsep orientasi objek, yang terdiri dari pembuatan kelas dan objek, struktur *gen-spec*, struktur *whole-part*, atribut, dan layanan. Tingkat dukungan terhadap fitur ini membagi bahasa pemrograman menjadi beberapa kategori, misalnya bahasa *object-based* (mendukung abstraksi data dan kelas), bahasa *object-oriented* (bahasa *object-based* yang juga mendukung sifat pewarisan atau hirarki kelas), dan sebagainya. OOP memberikan beberapa keuntungan dan kemudahan dalam pemrograman, antara lain :

- Representasi dasar yang seragam. Sumber permasalahan dikembangkan dengan metode *object-oriented* menggunakan representasi yang stabil, yaitu kelas. Representasi ini digunakan pada sumber permasalahan, OOA, OOD, dan OOP.
- *Reusability*. Konsep ini membuat usaha untuk pengembangan sistem menjadi lebih mudah. Hasil OOA, OOD, dan OOP dapat digunakan kembali dan dikembangkan dengan lebih cepat.
- Pemeliharaan. Penulisan program yang terstruktur berdasarkan objek membuat kode program mudah untuk dibaca sangat membantu dalam usaha perbaikan sistem.

BAB 3

ANALISIS SISTEM AWAL

Pada bab ketiga ini akan dilakukan telaah terhadap sistem informasi akuntansi awal, yang dikembangkan pada awal tahun 2000. Dari telaah ini akan dianalisis lebih jauh untuk mencari faktor-faktor penyebab tidak optimalnya kinerja dari sistem in*form*asi akuntansi tersebut, dalam hal ini berhubungan langsung dengan perangkat lunak yang digunakan. Melalui identifikasi awal, faktor-faktor yang umumnya menjadi penyebab tidak optimalnya kinerja suatu perangkat lunak dapat dikelompokkankan menjadi :

- Kesalahan/ ketidaktepatan kode pemrograman.
- Fungsionalitas yang tidak dapat memenuhi kebutuhan.

Kedua faktor ini menjadi acuan awal dalam melakukan analisis terhadap sistem perangkat lunak yang saat ini telah digunakan di Keuskupan Bandung (selanjutnya disebut sebagai AccountBdg).

3.1 Kesalahan Kode Pemrograman

Kesalahan kode pemrograman adalah kesalahan-kesalahan yang berhubungan dengan teknis pemrograman. Kesalahan-kesalahan ini tidak terdekteksi secara langsung oleh pengguna program dalam bentuk pesan kesalahan. Oleh karena itu, untuk mendeteksi kesalahan kode pemrograman perlu dilakukan pemeriksaan terhadap kode program yang bersangkutan, bagian demi bagian.

Kesalahan kode pemrograman dapat ditinjau dari dua sisi. Pertama, sisi pengalokasian memory dan yang kedua adalah syntax atau perintah yang digunakan untuk menjalankan suatu fungsi.

3.1.1 Kesalahan Pengalokasian *Memory*

Alokasi memory yang tepat sangat menunjang per*form*ansi suatu perangkat lunak. Faktor ini berhubungan dengan penggunaan dan pembebasan *memory* setelah digunakan untuk menjalankan suatu fungsi tertentu. Jika suatu fungsi berjalan dengan menggunakan memory namun tidak membebaskannya dengan benar saat fungsi itu selesai, maka lokasi di *memory* tersebut tidak dapat digunakan untuk menjalankan fungsi lainnya. Akibatnya, memory yang dapat digunakan untuk menjalankan fungsi lainnya menjadi berkurang. Kurangnya *memory* dapat menyebabkan perangkat lunak menjadi lambat dalam menjalankan fungsinya.

Telaah program (*tracing*) pada *source code* AccountBdg ditemukan banyak *code* yang rentan akan menimbulkan kesalahan pengalokasian memory. Beberapa ketidaksesuaian *code*, terutama ditemukan pada *code* untuk memunculkan suatu *form* pada AccountBdg. Hal ini terlihat pada potongan kode berikut :

```
procedure TMain.About1Click(Sender: TObject);
begin
with TAccountLogo.Create(self) do
try ShowModal;
finally Free;
end;
end;
```

Kode tersebut digunakan untuk memunculkan *form* AccountLogo. Kesalahan pada potongan kode tersebut adalah tidak adanya variabel yang menampung dan menjadi acuan dari obyek TAccountLogo. Apabila terjadi kesalahan pada saat

pengeksekusian *form*, maka alokasi memory untuk *form* tersebut tidak dapat dibebaskan karena program tidak memiliki referensi terhadap lokasi memory.

3.1.2 Kesalahan *Syntax* atau Perintah

Fungsionalitas dari suatu aplikasi dapat diperoleh dengan memanfaatkan satu atau beberapa perintah yang digabungkan. Dalam menggunakan suatu perintah perlu dipertimbangkan kondisi yang tepat untuk menerapkan perintah tersebut. Adanya beberapa pilihan perintah untuk menjalankan suatu fungsi mengharuskan seorang pemrogram untuk dapat memilih perintah yang paling baik untuk diterapkan dalam kondisi yang dihadapinya.

```
procedure TMain.About1Click(Sender: TObject);
begin
with TAccountLogo.Create(self) do
try ShowModal;
finally Free;
end;
end;
```

Program AccountBdg banyak menggunakan perintah **showmodal** untuk menampilkan suatu *form*. Hal ini menyebabkan *form* dimunculkan dengan sifat *modal*, yang menyebabkan pengguna tidak dapat bekerja dengan *form* lain sebelum *form* tersebut ditutup. Hal ini menyebabkan turunnya per*form*ansi sistem, karena suatu modul tidak dapat dimanfaatkan sebelum modul lain belum selesai digunakan. Suatu *form* yang bersifat modal tepat untuk digunakan dalam kondisi dimana terdapat kondisi "pekerjaan lain bergantung pada hasil dari pekerjaan saat ini". Selain itu, *form* ini juga cocok digunakan untuk menampilkan pesan kesalahan dimana sistem harus memastikan bahwa pengguna membaca pesan kesalahan tersebut sebelum melanjutkan pekerjaan.

Modul Neraca dari sistem AccountBdg juga memiliki kesalahan perintah pada saat mengakses basisdata. Kesalahan pertama adalah penggunaan metode **Insert** untuk menambahkan data . Metode Insert adalah metode yang seharusnya digunakan untuk menyisipkan data. Dalam basis data penyisipan data dan penambahan data adalah dua hal yang berbeda dilihat dari sisi proses penambahan datanya. Metode Insert menambahkan data pada posisi dimana *pointer* data berada. Sebelum data ditambahkan, data lain yang berada di posisi setelah *pointer* akan digeser terlebih dahulu baru kemudian data ditambahkan.

Metode yang sesuai untuk penambahan data seharusnya adalah Append. Metode Append selalu menambahkan data di posisi terakhir. Hal ini dapat meningkatkan per*form*ansi karena sistem tidak lagi terbebani dengan proses menggeser data. Kesalahan tersebut terlihat pada potongan program di bawah ini :

```
while not DM1.Table13.EOF do
begin
Table2.Insert;
Table2KODE.Value:=DM1.Table13KODE.Value;
Table2POS.Value:=DM1.Table13POS.Value;
Table2BlnThn.Value:=bulantahun;
Table2UNIT.Value:=NamaUser;
Table2.Post;
ProgressBar1.StepIt;
DM1.Table13.Next;
end;
```

Pada potongan kode di atas juga dapat diamati bahwa kolom-kolom pada basis data diakses dengan metode **Value** yang mengembalikan nilai dengan tipe Varian, sedangkan kolom-kolom tersebut pada kenyataannya memiliki tipe yang spesifik, yaitu String. Tipe Varian adalah suatu tipe variabel yang dapat menampung berbagai jenis tipe variabel di dalamnya. Keistimewaan tipe Varian diimbangi dengan kerugiannya, yaitu kebutuhan memory yang lebih besar.

Perangkat pemrograman yang berorientasi obyek seperti Borland Delphi memungkinkan seorang pemrogram untuk menempatkan fungsi-fungsi program ke dalam *event handler* dari suatu komponen. Alih-alih mendeklarasikan suatu fungsi secara mandiri, beberapa pemrogram malah melakukan praktek seperti diuraikan sebelumnya untuk menghindari penambahan beban kerja dengan mendeklarasikan fungsi.

Praktek seperti ini sebenarnya tidak mempengaruhi program dari sisi fungsi yang mampu dilakukannya, tetapi membebani program dengan perlunya tambahan memory untuk menampung komponen-komponen yang tidak diperlukan (lebih boros *memory*). Berikut adalah potongan kode dari program AccountBdg yang menunjukkan praktek ini :

```
procedure Tfm_BH.BitBtn4Click(Sender: TObject);
begin
hitungbtn.Click;
Panel6.Visible:=True;
end;
```

3.1.2 Kesalahan Penamaan Komponen

Kesalahan penamaan komponen yang dimaksud bukanlah kesalahaan penamaan yang mengakibatkan program tidak dapat berjalan sebagaimana mestinya. Kesalahan penamaan lebih mengacu pada penamaan komponen yang tidak sesuai dengan fungsinya. Secara default pada waktu perancangannya, Borland Delphi sudah memberikan nama standar untuk setiap komponen yang digunakan. Format penamaan yang digunakan umumnya berupa `<Kelas><No.Urut>`, misalnya untuk *instan* pertama dari kelas Bitbtn akan diberi nama BitBtn1.

Seorang pemrogram seharusnya mengubah nama dari komponen yang ditambahkannya ke dalam bentuk yang lebih sesuai. Misalnya untuk Bitbtn yang berisi fungsi penyimpanan data, sebaiknya diberi nama btSimpan. Hal ini tidak

akan mempengaruhi jalannya program. Perbaikan penamaan seperti ini lebih ditujukan untuk mempermudah proses dokumentasi dan proses penelusuran. Berikut adalah potongan kode program pada AccountBdg yang menyalahi aturan penamaan komponen :

```
BitBtn22.Visible:=True;
Panel19.Visible:=False;
Panel16.Visible:=False;
Timer1.Enabled:=False;
Image4.Visible:=False;
BitBtn18.Enabled:=True;
BitBtn29.Enabled:=True;
BitBtn18.Visible:=False;
BitBtn29.Visible:=False;
```

3.2 Fungsionalitas yang Tidak Dapat Memenuhi Kebutuhan

Sebuah sistem memiliki siklus hidup. Seiring berkembangnya waktu, sistem menjadi kuno dan menjadi tidak sesuai untuk diterapkan dalam kondisi terkini. Munculnya kebutuhan-kebutuhan baru menyebabkan program AccountBdg perlu mengalami pemutakhiran. Kebutuhan-kebutuhan baru tersebut antara lain adalah :

1. Tampilan yang lebih berorientasi pada pengguna
2. Validasi input data buku harian secara manual
3. Validasi input data buku harian secara otomatis
4. Kemampuan pembuatan daftar saldo awal secara otomatis
5. Rekapitulasi buku harian dan buku besar
6. Tampilan proses yang lebih informatif
7. Pembuatan ringkasan periodik 3 bulanan, 6 bulanan dan 1 tahunan
8. Penyusunan neraca konsolidasi
9. Penyesuaian kode transaksi.

Munculnya kebutuhan-kebutuhan tersebut menyebabkan tuntutan terhadap pembaharuan AccountBdg meningkat. Penggunaan sistem saat ini tanpa adanya perubahan dapat mengakibatkan kesalahan fatal dalam hal kesesuaian hasil antara program dengan hasil yang seharusnya.

BAB 4

PERANCANGAN SISTEM BARU

Pada bab keempat ini akan dilakukan perancangan sistem baru, sebagai jawaban atas kelemahan sistem lama yang telah dipaparkan pada bab tiga.

4.1 Perbaikan Kesalahan Kode Pemrograman

Pada sistem lama, ditemukan beberapa sub routine program yang mengakibatkan terjadinya pemborosan sumber daya memory (tidak efisien), seperti potongan program berikut ini :

```
procedure TMain.About1Click(Sender: TObject);
begin
with TAccountLogo.Create(self) do
try ShowModal;
finally Free;
end;
end;
```

Perbaikan yang dilakukan adalah mengubah *routine* di atas menjadi :

```
procedure TfMain. About1Click(Sender: TObject);
begin
if Fm_Bh=nil then
   Fm_Bh:= TAccountLogo.Create(self);
try
Fm_Bh.Show;
finally
FreeAndNil(Fm_Bh);
end;
end;
```

Keuntungan menggunakan metode seperti ini adalah :

1. Mencegah 2 atau lebih form yang sama terbuka secara bersama-sama.
2. Menyediakan variabel referensi pada objek yang aktif sehingga memudahkan pengendalian objek yang aktif tersebut.
3. Selain memudahkan pengendalian objek, penggunaan variabel referensi juga dapat mengefisienkan penggunaan memory dengan cara menghapus objek-objek yang tidak diperlukan lagi dari memory, sehingga memory tersebut dapat ditempati oleh objek yang lain yang aktif.
4. Penggunaan perintah **show** untuk menggantikan perintah **showmodal** memungkinkan pengguna untuk tetap dapat bekerja dengan beberapa form yang berbeda.
5. Program baru menggantikan metode **showmodal** dan tanpa variabel referensi ini pada sebanyak 37 form. Diharapkan, hal ini cukup dapat meningkatkan performansi sistem informasi baru ini dibanding dengan yang lama.

4.2 Perbaikan Kesalahan Syntax dan Perintah

Kesalahan yang ditemukan pada sistem informasi lama adalah penggunaan metode **Insert** untuk menambahkan data . Metode Insert adalah metode yang seharusnya digunakan untuk menyisipkan data. Dalam basis data penyisipan data dan penambahan data adalah dua hal yang berbeda dilihat dari sisi proses penambahan datanya.

Metode Insert menambahkan data pada posisi dimana *pointer* data berada. Sebelum data ditambahkan, data lain yang berada di posisi setelah *pointer* akan digeser terlebih dahulu baru kemudian data ditambahkan. Metode insert

memerlukan langkah yang panjang, apalagi jika perintah ini dijalankan pada suatu basis data/ tabel yang sudah memiliki banyak record. Proses ini akan menggeser sejumlah besar *record* ke posisi baru sebelum data disisipkan. Dapat dibayangkan jika terdapat ribuan, puluhan ribu atau bahkan ratusan ribu data (record) yang harus digeser. Hal ini akan memerlukan waktu yang sangat lama.

Metode yang sesuai untuk penambahan data seharusnya adalah Append. Metode Append selalu menambahkan data di posisi terakhir. Hal ini dapat meningkatkan performansi karena sistem tidak lagi terbebani dengan proses menggeser data. Kesalahan tersebut terlihat pada potongan program di bawah ini :

```
while not DM1.Table13.EOF do
begin
Table2.Insert;
Table2KODE.Value:=DM1.Table13KODE.Value;
Table2POS.Value:=DM1.Table13POS.Value;
Table2BlnThn.Value:=bulantahun;
Table2UNIT.Value:=NamaUser;
Table2.Post;
ProgressBar1.StepIt;
DM1.Table13.Next;
end;
```

Sebagai ganti dari penggunaan perintah **Insert** pada *routine* program di atas, maka perintah insert digantikan dengan perintah **Append**, seperti terlihat pada potongan program berikut :

```
while not DM1.Table13.EOF do
begin
Table2.Append;
Table2KODE.Value:=DM1.Table13KODE.Value;
Table2POS.Value:=DM1.Table13POS.Value;
.
.
.
end;
```

Pada potongan kode di atas juga dapat diamati bahwa kolom-kolom pada basis data diakses dengan metode **Value** yang mengembalikan nilai dengan tipe Varian,

sedangkan kolom-kolom tersebut pada kenyataannya memiliki tipe yang spesifik, yaitu String. Tipe Varian adalah suatu tipe variabel yang dapat menampung berbagai jenis tipe variabel di dalamnya. Keistimewaan tipe Varian diimbangi dengan kerugiannya, yaitu kebutuhan memory yang lebih besar.

Sebagai ganti dari penggunaan perintah **Value** pada *routine* program di atas, maka perintah value digantikan dengan tipe variabel yang sesuai dengan nilai yang tersimpan dalam kolom pada tabel yang bersesuaian, seperti terlihat pada potongan program berikut :

```
while not DM1.Table13.EOF do
begin
Table2.Insert;
Table2KODE.Value:=DM1.Table13KODE.AsString;
Table2POS.Value:=DM1.Table13POS. AsString;
Table2BlnThn.AsInteger:=bulantahun;
Table2UNIT. AsString:=NamaUser;
Table2.Post;
ProgressBar1.StepIt;
DM1.Table13.Next;
end;
```

Pada proses optimasi sistem informasi ini, telah dilakukan penggantian tipe variabel agar lebih sesuai, pada lebih dari 10.000 baris kode yang ada.

Kesalahan ketiga yang ditemukan pada sistem informasi lama adalah penggunaan komponen untuk menyimpan suatu fungsi atau prosedur, seperti terlihat pada potongan program berikut :

```
procedure Tfm_BH.BitBtn4Click(Sender: TObject);
begin
hitungbtn.Click;
Panel6.Visible:=True;
end;
```

Penggunaan komponen untuk menyimpan fungsi/ prosedur seharusnya dihindari karena akan mengakibatkan penggunaan memory yang tidak efisien (lebih besar daripada yang seharusnya diperlukan). Kelebihan penggunaan memory ini muncul sebagai akibat dari penggunaan komponen yang sebetulnya tidak dibutuhkan. Oleh karena itu, metode ini digantikan menjadi seperti berikut :

```
procedure Tfm_BH. BitBtn4Click (Sender: TObject);
begin
hitung;
Panel6.Visible:=True;
End;
```

Routine program di atas dapat berjalan dengan syarat prosedur **hitung** telah didefinisikan sebelumnya.

Kesalahan keempat yang ditemukan pada sistem informasi lama adalah penggunaan proses *indexing* setiap kali sebelum dilakukan proses pencarian sebuah data dalam sebuah tabel, seperti terlihat pada potongan program berikut :

```
begin
with DM1 do
begin
Table13.IndexFieldNames:='KODE';
Table13.SetKey;
Table13.FieldByName('KODE').AsString:=ComboBox4.Text;
Table13.GotoNearest;
end;
```

Proses indexing yang dimasukkan ke dalam rangkaian proses pencarian sebuah data memerlukan waktu yang lama, yang akan meningkat seiring dengan penambahan jumlah data/record pada sebuah tabel. Oleh karena itu, proses indexing dihilangkan dari rangkaian proses pencarian seperti terlihat pada potongan program berikut :

```
begin
with DM1 do
Table13.Locate('KODE',ComboBox4.Text,[]);
```

Pada potongan kode di atas, perintah **GotoNearest** yang membutuhkan proses *indexing*, digantikan dengan perintah **Locate** yang lebih sederhana dan tidak memerlukan proses *indexing*.

Penggantian semua proses pencarian data yang menggunakan *indexing* membuat performansi program baru menjadi lebih baik. Jika pada program lama memerlukan waktu lebih dari 10 menit untuk melakukan proses pembuatan buku besar, neraca saldo beserta seluruh laporan keuangan yang diperlukan, maka setelah menggunakan perintah Locate ini, waktu yang dibutuhkan menjadi hanya sekitar 2 – 3 menit saja. Terjadi pengurangan waktu proses yang cukup signifikan, yaitu sebesar 70%.

4.3 Perbaikan Kesalahan Penamaan Komponen

Kesalahan lain yang ditemukan pada sistem informasi lama adalah kesalahan penamaan komponen. Ketidaksesuaian penamaan komponen yang digunakan pada sebuah program, sebenarnya tidak akan mempengaruhi performansi program itu sendiri. Penamaan program yang baik sebenarnya lebih ditujukan uuntuk memudahkan proses dokumentasi dan pengembangan program.

Contoh kesalahan penamaan komponen dapat dilihat pada potongan program berikut :

```
procedure Tfm_BH. BitBtn4Click (Sender: TObject);
begin
hitungbtn.Click;
Panel6.Visible:=True;
End;
```

Pada proses optimasi sistem informasi ini, penamaan komponen di atas telah diubah menjadi seperti berikut :

```
procedure Tfm_BH. btHitungClick (Sender: TObject);
begin
```

```
hitung;
Panel6.Visible:=True;
End;
```

Dengan penggantian nama komponen menjadi seperti di atas, akan lebih memudahkan pemrogram dalam mengidentifikasi fungsi atau perilaku sebuah komponen. Pada contoh di atas, hal ini berarti : komponen BitButton4 yang bertipe bitbutton menjalankan fungsi untuk menghitung. Jika masih menggunakan nama BitButton4, maka pemrogram harus melacak terrlebih dahulu fungsi yang terkandung dalam komponen ini. Dengan mengubah nama BitButton4 menjadi btHitung, pemrogram dapat langsung mengenali fungsi dari komponen ini. Hal ini akan mempercepat pemahaman pemrogram terhadap alur program sistem informasi ini.

Beberapa standar penamaan yang digunakan pada optimasi sistem informasi ini antara lain :

Tabel 4.1 Standar Penamaan Beberapa Komponen

Tipe Komponen	Kode awalan	Contoh
Button	bt	btHitung, btCari
BitButton	bt	btHitung, btCari
EditBox	ed	edNama, edKode
ComboBox	cb	cbKota, cbNegara
Tabel	tb	tbKaryawan, tbPos
RadioGroup	rg	rgJKelamin

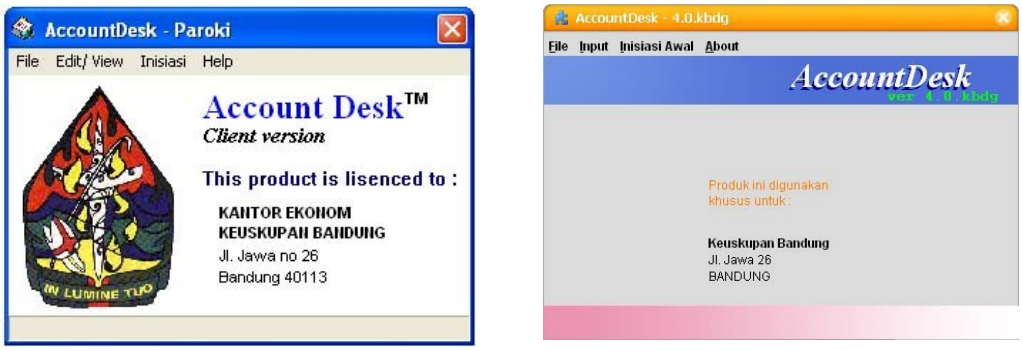
4.4 Perbaikan dan Penambahan Beberapa Fungsi

Pada sub bab ini akan dijabarkan beberapa perbaikan terhadap beberapa kekurangan yang telah disebutkan pada sub bab 3.2 seperti antara lain : tampilan yang lebih berorientasi pada pengguna, validasi input data buku harian secara manual dan otomatis, kemampuan pembuatan daftar saldo awal secara otomatis dan lain sebagainya.

4.4.1 Perbaikan Tampilan

Permintaan perbaikan tampilan program diajukan oleh pihak Unit Keuangan Keuskupan Bandung, terutama karena program lama dinilai bernuansa 'suram' dan diinginkan agar diubah ke suatu suasana tampilan yang lebih 'segar, sederhana & baru'. Oleh karena itu, pada program perbaikan ini, tampilan sengaja dibuat lebih berwarna, dengan dominasi aksen warna oranye, biru muda, merah muda dan hijau muda pada tampilannya.

Beberapa contoh perbaikan tampilan program dapat dilihat pada Gambar 4.1 di bawah ini.

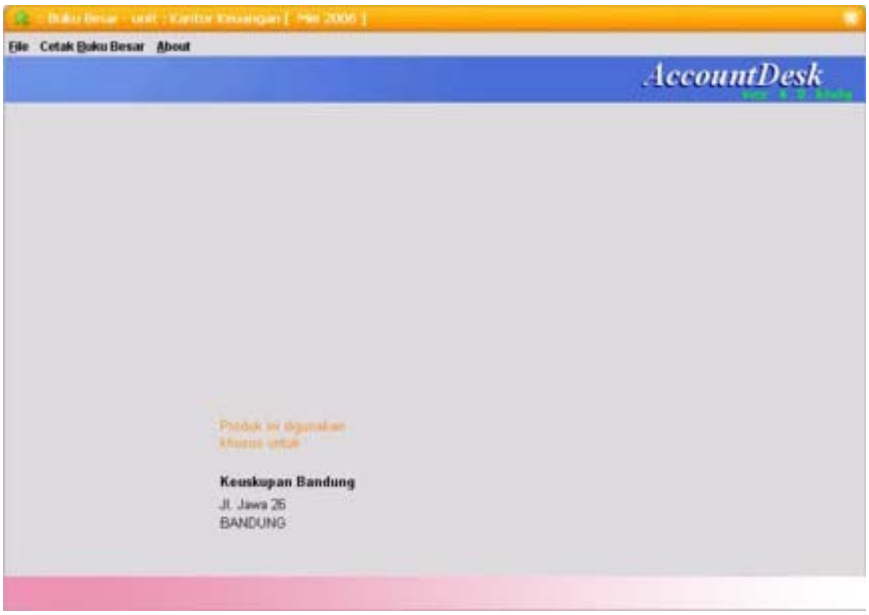


(a) Lama (b) Baru

Gambar 4.1 Perbandingan tampilan awal program lama dan baru

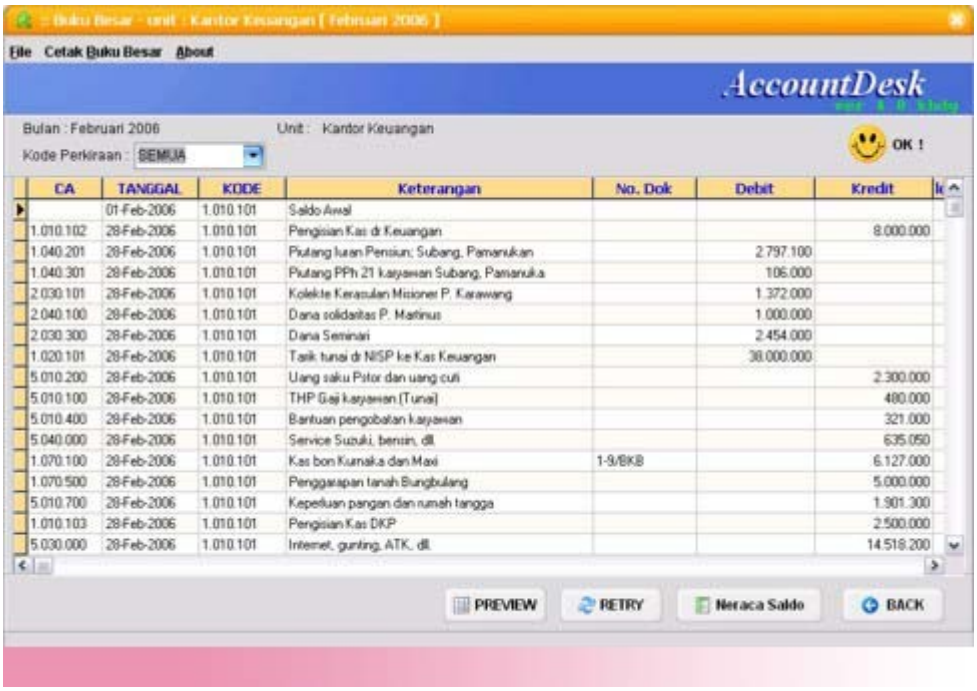
Logo Keuskupan pada program baru ini memang belum dipasang, karena pada saat program ini selesai dibuat, logo baru Keuskupan Bandung belum ada, akibat masih belum adanya Uskup Bandung yang baru.

Beberapa tampilan pada program baru antara lain :

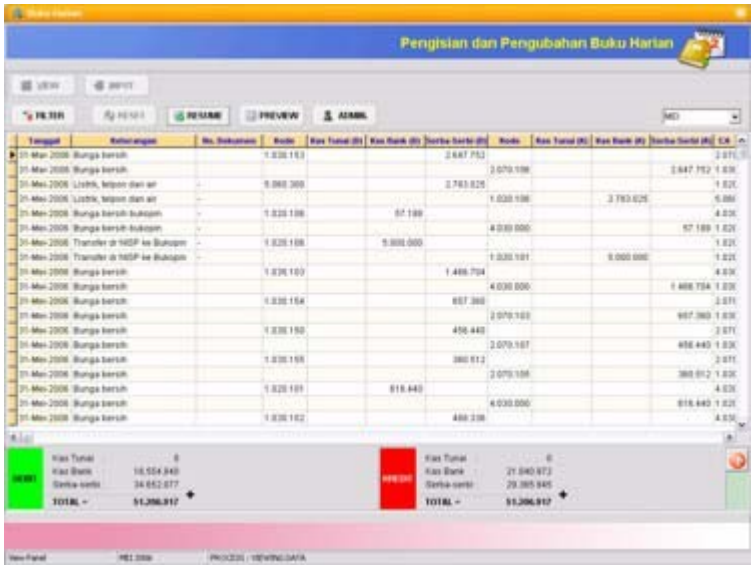


Gambar 4.2 Tampilan awal program untuk membuat laporan Buku Besar

Gambar 4.3 Tampilan program saat menunggu pembuatan laporan Buku Besar



Gambar 4.4 Tampilan hasil laporan Buku Besar



Gambar 4.5 Tampilan form pengisian Buku Harian Keuangan

Pada Gambar 4.2 sampai 4.3, tampak bahwa tampilan program baru memiliki nuansa yang sederhana, bersih dan segar yang diharapkan akan membuat mata operator pengguna program ini tidak cepat lelah. Pada Gambar 4.4 sampai 4.5, juga tampak bahwa tampilan awal program adalah berupa tampilan yang berbentuk tabel. Tampilan seperti ini memang diinginkan oleh pihak unit Keuangan Keuskupan Bandung, karena mereka sudah terbiasa menggunakan program spreadsheet seperti Microsoft Excel.

4.4.2 Validasi Input Data Buku Harian Secara Manual dan Otomatis

Salah satu permintaan pihak unit Keuangan Keuskupan Bandung untuk optimasi program keuangan mereka adalah membuat program pemasukan data keuangan harian dapat divalidasi baik secara manual maupun otomatis. Hal ini disingkapi dengan membuat sebuah fungsi validasi input (function Tfm_BH.TransactionValidation) seperti berikut :

```
function Tfm_BH.TransactionValidation: boolean;
var
  JmlDebet,JmlKredit : real;
begin
    JmlDebet:=0;
    JmlKredit:=0;
  //---memeriksa apakah saldo debet dan kredit yang
dimasukkan sudah seimbang
  with dm1 do begin
    TTmpDebit.DisableControls;
    TTmpKredit.DisableControls;
    TTmpDebit.First;
    while not TTmpDebit.Eof do begin
      JmlDebet := JmlDebet + TTmpDebitJml_Trans.AsFloat;
      TTmpDebit.Next;
    end;
    TTmpKredit.First;
    while not TTmpKredit.Eof do begin
      JmlKredit := JmlKredit +
TTmpKreditJml_Trans.AsFloat;
      TTmpKredit.Next;
    end;
    TTmpKredit.EnableControls;
    TTmpDebit.EnableControls;
  end;
  if JmlDebet = JmlKredit then begin
```

```
      Result:=True;
    end else begin
      Result:=false;
    end;
  end;
```

Fungsi ini bertugas untuk memeriksa apakah pencatatan debit dan kredit pada 1 hari tersebut seimbang. Keluaran dari fungsi ini adalah variabel boolean, sehingga cuma ada 2 nilai saja yang mungkin, yaitu TRUE atau FALSE. Fungsi ini akan mengeluarkan nilai TRUE jika jumlah debit sama dengan jumlah kredit, dan akan bernilai FALSE jika jumlah debit tidak sama dengan jumlah kredit.

4.4.3 Kemampuan Membuat Daftar Saldo Awal Secara Otomatis

Pada program yang lama, daftar saldo awal tidak dapat secara otomatis terintegrasi ke dalam pencatatan buku harian dan seterusnya. Oleh karena itu, pada proses optimasi ini dilakukan penambahan sebuah program baru untuk melakukan pengisian saldo awal dan sekaligus mengintegrasikannya ke dalam sistem informasi keuangan ini. Pengisian saldo awal ini biasanya dilakukan pada setiap awal tahun anggaran baru. Tampilan awal program pencatatan saldo awal dapat dilihat pada Gambar 4.6. Sementara pengisian saldo awalnya dapat dilihat pada Gambar 4.7.

Gambar 4.6 Tampilan awal program pencatatan saldo awal

Gambar 4.7 Tampilan program pengisian saldo awal

Program pengisian saldo awal ini dilengkapi dengan fungsi pengisian otomatis saldo awal tahun, untuk masing-masing pos, berdasar atas saldo akhir tahun sebelumnya. Fungsi pengisian otomatis ini dapat dilihat pada potongan program di bawah ini :

```
//------------ memindahkan saldo ------------------------//
// memindahkan saldo akhir tahun dari folder (tahunbuka-1) ke
tabel1 [TAwalThn]   //
  TAkhirThn.TableName:=TahunSblm + '\AkhirThn.DB';
  TAkhirThn.Open;
  Table1.First;
  while not Table1.Eof do begin
        if TAkhirThn.Locate('Kode',Table1Kode.AsString,
[loCaseInsensitive]) then begin
            Table1.Edit;
            Table1Saldo_Awal.AsFloat:=TAkhirThn.FieldByName('S
aldo_Akhir').AsFloat;
            Table1.Post;
        end;
        Table1.Next;
  end;
  TAkhirThn.Close;
```

```
//--- mengisi saldo untuk pos gabungan ---------------------

    Table1.First;
    QKelompok.Open;
    QKelompok.First;
    while not QKelompok.Eof do begin
        Table1.Filter:='KODE =
'+QuotedStr(Copy(QKelompok.FieldByName('Kode').AsString,1,7)+'
*');
        Table1.Filtered:=True;
        if Table1.RecordCount>1 then begin
            Table1.First;
            Table1.Edit;
            Table1.FieldByName('Saldo_Awal').AsFloat:=0;
            Table1.Post;
        end;
        Table1.Filtered:=false;
        QKelompok.next;
    end;
```

Pada program pengisian saldo awal ini, juga terdapat fungsi pemeriksaan kesetimbangan antara Total Aktiva, Total Kewajiban, Total Aktiva Bersih serta Total Aktiva Bersih dan Kewajiban. Fungsi pemeriksaan kesetimbangan ini dapat dilihat pada potongan program di bawah ini :

```
procedure Tfm_SaldoAwal.BitBtn1Click(Sender: TObject);
var
Bookmark: TBookmark;
TOT_AKTIVA, TOT_WAJIB,TOT_KODE3,KekayaanBersih,TotAktKwjb :
Real;
Kodeawal : string;

begin
// menjumlah TOTAL AKTIVA, TOTAL KEWAJIBAN
// dan TOTAL KEKAYAAN BERSIH

Bookmark:=Table1.GetBookmark;
Table1.DisableControls;
TOT_AKTIVA:=0;
TOT_WAJIB:=0;
TOT_KODE3:=0;

try
Table1.First;
while not Table1.EOF do
begin
Kodeawal:=copy(Table1KODE.AsString,1,1);
if Kodeawal='1' then
```

```
TOT_AKTIVA:=TOT_AKTIVA+Table1Saldo_Awal.AsFloat;
if Kodeawal='2' then
TOT_WAJIB:=TOT_WAJIB+Table1Saldo_Awal.AsFloat;
if (KodeAwal='3') and (Table1Kode.AsString<>'3.010.100') then
TOT_KODE3:=TOT_KODE3 + Table1Saldo_Awal.AsFloat;
Table1.Next;
end;
finally
…
KekayaanBersih:=TOT_AKTIVA-TOT_WAJIB-TOT_KODE3;
Table1.DisableControls;
if Table1.Locate('Kode','3.010.100',[locaseInsensitive]) then
  Table1.Edit;
  Table1Saldo_Awal.AsFloat:=KekayaanBersih;
  Table1.Post;

Table1.EnableControls;
KBersihLabel.Caption:=FormatFloat('#,0',KekayaanBersih+TOT_KOD
E3);
TotAktKwjb := TOT_WAJIB + KekayaanBersih + TOT_KODE3;

…
Table1.First;
while not Table1.EOF do
begin
if Table1KODE.AsString='3.010.100' then
begin
Table1.edit;
Table1Saldo_Awal.AsFloat:=KekayaanBersih;
Table1.post;
end;
Table1.next;
end;
Table1Saldo_Awal.DisplayFormat:='#,#';

with dm1 do begin
   while not TSumSA.IsEmpty do TSumSA.Delete;
   TSumSA.Append;
   TSumSATotAkt.AsFloat:=Tot_Aktiva;
   TSumSATotKwjb.AsFloat:=Tot_Wajib;
   TSumSATotAktBrs.AsFloat:=KekayaanBersih+TOT_KODE3;
   TSumSATotAktKwjb.AsFloat:=TotAktKwjb;
   TSumSA.Post;
   end;
end;
```

BAB 5

KESIMPULAN

Sistem informasi akuntansi di Keuskupan Bandung yang diterapkan sebelumnya pada dasarnya memang telah memasuki tahapan akhir dari siklus hidup suatu produk. Hal ini diindikasikan dengan ketidakmampuan sistem informasi yang lama untuk mengakomodasi kebutuhan pengguna saat ini.

5.1 Kesimpulan

Perbaikan dan pengembangan yang telah berhasil dilakukan terhadap sistem informasi ini meliputi :

1. Peningkatan kecepatan proses penghitungan data yang lebih cepat. Program baru ini berhasil meningkatkan kecepatan pemrosesan data sebesar 300%.
2. Penambahan utilitas untuk memasukkan anggaran, yang juga dapat terhubung dengan utilitas pemasukan saldo awal tahun.
3. Penambahan utilitas untuk dapat melakukan proses konsolidasi laporan dari beberapa paroki/ unit/ komisi.
4. Penambahan utilitas untuk memeriksa realisasi anggaran yang telah terpakai sampai pada satu saat tertentu.
5. Pembuatan bentuk/ format laporan yang baru, sesuai dengan kebutuhan saat ini.
6. Perubahan *layout* tampilan program yang baru, yang bernuansa lebih sederhana, segar tapi modern.

5.2 Saran

Saran yang dapat diberikan untuk pengembangan sistem informasi ini lebih lanjut adalah :

1. Pihak Kantor Ekonom Keuskupan Bandung perlu memikirkan langkah berikutnya untuk mengintegrasikan sistem informasi akuntansi ini dengan sistem-sitem lain yang ada di Keuskupan Bandung, seperti sistem transaksi penjualan di beberapa unit bisnis Keuskupan Bandung.
2. Perlu dipikirkan juga untuk melakukan integrasi data yang terotomasi secara *online* antara unit keuangan Keuskupan Bandung dengan paroki-paroki dan unit-unit yang ada di bawahnya.

DAFTAR PUSTAKA

1. Alter, Steven. 1992. *Information Systems : A Management Perspective*. The Benjamin -Cummings Publishing Company, Inc. USA.
2. Cantu, Marco. 1996. *Mastering Delphi 2 for Windows 95/NT.* Sybex Inc. California. USA.
3. Cantu, Marco. 2001. M*astering Delphi 6.* Sybex Inc. California. USA.
4. Coad, P. Dan Yourdon, E. 1991. *Object-Oriented Analysis, 2nd ed*. Prentice-Hall International, Inc. New Jersey. USA.
5. Coad, P. Dan Yourdon, E. 1991. *Object-Oriented Design*. Prentice-Hall International, Inc. New Jersey. USA.
6. Jogiyanto, HM. 1990. *Analisis dan Desain Sistem Informasi : Pendekatan Terstruktur*. Andi Offset. Jogyakarta. Indonesia.