MODUL PEMBELAJARAN

PENGUJIAN DAN IMPLEMENTASI SISTEM

Disusun Oleh Dosen

Achmad Baroqah Pohan , M.Kom

UNIVERSITAS BINA SARANA INFORMATIKA
Jakarta
2018

DAFTAR ISI

PERTEMUAN I

PENGUJIAN DAN IMPLEMENTASI SISTEM

PENGHANTAR

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

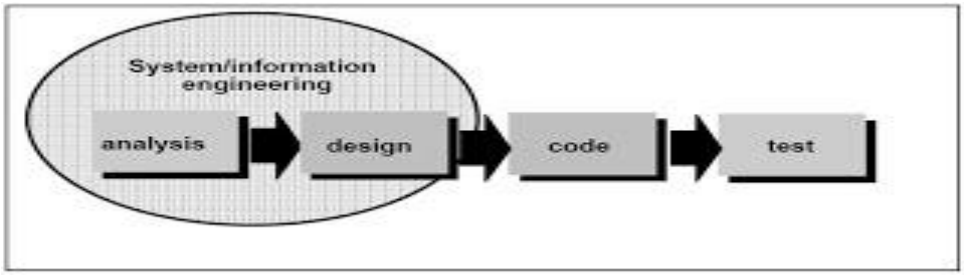
Pertemuan : 1 (Satu)	Waktu :

Modul	1 (Satu)
Topik	Penghantar
Sub Topik	-
Materi	• Siklus hidup pengembangan sistem • Pengkodean • Pengujian • Dokumentasi
Tujuan	Mahasiswa akan dapat menerangkan peranan tahap pengujian dan tahap implementasi pada rangkaian siklus hidup pengembangan sistem perangkat lunak

Siklus Hidup Pengembangan Sistem

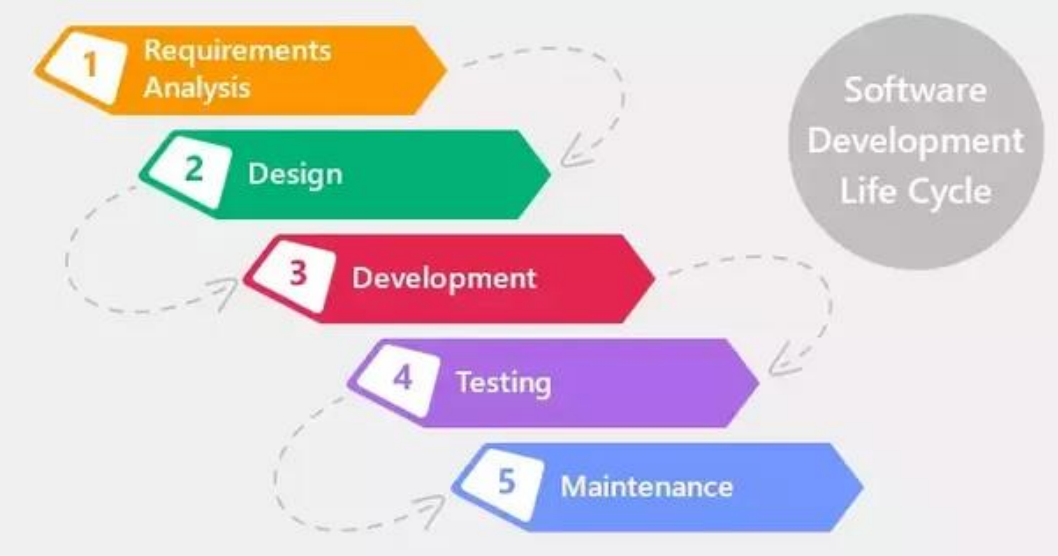
Proses bisnis yang berjalan di dalam organisasi semakin lama semakin berkembang. Proses transaksi yang ada juga semakin rumit. Dalam hal ini, suatu organisasi tidak bisa hanya mengandalkan pemrosesan transaksi secara tradisional. Oleh karena itu, pengembangan sistem informasi merupakan suatu keharusan bagi sebuah organisasi dalam menjalankan aktivitas bisnisnya.

Pengembangan sistem informasi merupakan proses atau prosedur yang harus diikuti untuk melaksanakan seluruh langkah dalam menganalisis, merancang, mengimplementasikan dan memelihara sistem informasi. Proses proses pengembangan sistem atau SDLC (system development life cycle). SDLC yang terkenal adalah SDLC Model klasik yang biasa disebut dengan waterfall. Menurut Pressman (2001), Tahapan tahapan waterfall terlihat pada gambar di bawah ini :



Gambar 1. SDLC Waterfall menurut Roger Pressman

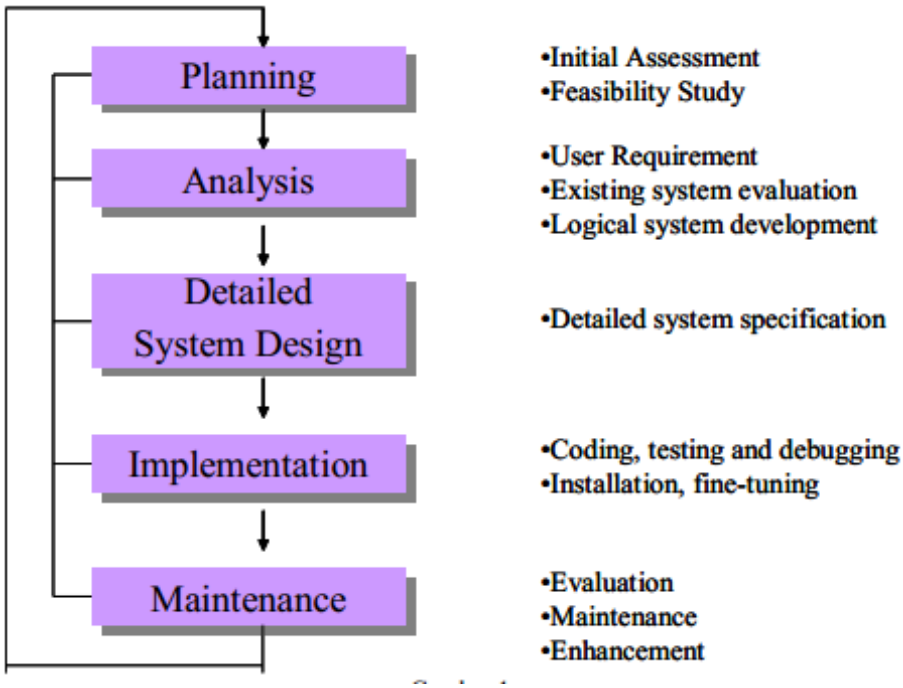
Sedangkan menurut sommerville (2001) fase fase SDLC model klasik terlihat pada gambar di bawah ini :



Gambar 2. SDLC Waterfall menurut Sommerville

Meskipun dalam kedua gambar tersebut menggunakan istilah yang berbeda, namun pada dasarnya sama.

Pada saat ini perkembangan proyek sistem informasi dilihat dari segi kompleksitas dan kapasitasnya, saat ini banyak methodology atau pendekatan yang dikembangkan dari tahapan siklus secara umum. Sebagai gambaran dapat dilihat pada bagan dibawah ini :



Gambar 3. Tahapan Pengembangan Sistem

(Dewanto, Jurnal Fasilkom)

1. Perencanaan

   Tahapan ini biasanya dilaksanakan untuk memberikan jawaban dari pertanyaan apakah proyek tersebut layak untuk dikerjakan dari sisi teknis, operasional dan juga biaya. Untuk itu biasanya dikerjakan beberapa aktifitas yang harus dikerjakan, sebagai berikut :

   - Mengenali dan memastikan masalah
   - Menentukan objektif
   - Mengidentifikasikan ruang lingkup sistem
   - Mengatur dan menjalankan studi kelayakan system
   - Menyiapkan proposal system yang akan dikerjakan
   - Membangun mekanisme pengawasan

2. Analisa

   Tahapan analisa mencoba menganalisa secara makro analisa

   - Menganalisa kebutuhan informasi dari end-user, lingkungan perusahaan dan juga jika ada sistem yang sedang digunakan.
   - Mengorganisir tim proyek

- Menentukan jenis studi sistem yang akan digunakan
- Menentukan kriteria performa sistem
- Menentukan kriteria keamanan sistem
- Menentukan aturan integritas data
- Menyiapkan proposal rancangan sistem

3. Perancangan
   - Identifikasi sistem alternative
   - Evaluasi konfigurasi sistem alternative
   - Menyiapkan bagan organisasi
   - Menyiapkan diagram data
   - Menyiapkan kamus data
   - Menyiapkan bentuk formulir tercetak
   - Menyiapkan diagram struktur, menyiapkan model objek rasional.
   - Menyiapkan spesifikasi kelas.

4. Implementasi
   - Rancangan diterjemahkan ke dalam kode program.
   - Memenuhi sumber daya perangkat lunak
   - Menyiapkan database
   - Menyiapkan fasilitas fisik
   - Melatih user
   - Menyiapkan proposal impelementasi

5. Perawatan

   Penggunaan adalah tahap terakhir dalam SDLC. Disini sistem baru akan digunakan dalam lingkungan sebenarnya dan dilakukan pemeliharaan agar system selalu mampu menyesuaikan diri dengan perubahan perubahan atau pengembangan dalam lingkungan, aktifitas yang dikerjakan adalah :

   - Menggunakan sistem baru
   - Evaluasi sistem baru

- Memelihara sistem seperti memperbaiki kesalahan kesalahan , menjaga sistem terupdate, juga mengembangkan sistem.

Implementasi sistem merupakan pengembangan dari tahap desain sistem. Tahap ini merupakan tahap yang menentukan pengembangan sistem, karena sebagus apapun desain yang telah dibuat, tetapi tidak ada implementasi akan tidak ada gunanya. Tahap implementasi mencakup pengkodean atau pemrograman, pengujian dan dokumentasi (Mulyanto,2009)

A. Pengkodean

Pengkodean atau yang lebih dkenal dengan pemrograman merupakan kegiatan analisis kebutuhan yang diterjemahkan ke dalam bahasa yang dimegerti oleh komputer dengan menggunakan bahasa pemrograman pemrograman merupakan metodologi pemecahan masalah, kemudian menuangkanya dalam suatu notasi tertentu yang mudah dibaca dan dipahami.

Dalam melakukan pemrograman, seorang programmer sering melakukan pengujian terhadap kode kode program untuk memastikan kebenaran program tersebut. Pengujian ini dilakukan untuk mencari kesalahan yang ditimbulkan karena salah tulis atau kesalahan pemrograman. Kegiatan ini disebut dengan istilah debugging. Kegiatan ini digunakan untuk mencari posisi kesalahan (error) dari kode kode program.

B. Pengujian

Pengujian atau testing merupakan proses pengeksekusian untuk menemukan kesalahan kesalahan yang terdapat di dalam sistem, kemudian dilakukan pembenahan. Tahap ini merupakan tahap yang penting dalam pengembangan sistem karena pada tahap ini merupakan tahapan untuk memastikan bahwa suatu sistem terbebas dari kesalahan. Penguji juga dilakukan dengan memerhatikan konsep pengembangan menurut sommerville (2001), tahap tahap pengujian antara lain :

1. Pengujian Unit

   Komponen individual diuji untuk menjamin operasi yang benar. Setiap komponen diuji secara independen , tanpa komponen sistem yang lain.
2. Pengujian Modul

Modul merupakan kumpulan kompone yang berhubungan seperti kelas, objek, tipe data abstrak, atau sekumpulan prosedur dan fungsi dengan hubungan yang luas. Sebuah modul merangkum komponen komponen yang berhubungan, sehingga dapat diuji tanpa modul sistem yang lain.

3. Pengujian Subsistem

   Fase ini melibatkan pengujian sekumpulan modul yang telah diintegrasikan menjadi subsistem. Masalah umum yang sering muncul pada sistem perangkat lunak yang besar adalah ketidaksesuaian interface. Proses pengujian subsistem seperti itu harus dikonsentrasikan pada deteksi kesalahan interface modul dengan menjalankan interface berkali-kali.

4. Pengujian Sistem

   Subsistem diintegrasikan untuk membentuk sistem. Proses ini berkenaan dengan penemuan kesalahan yang diakibatkan dari interaksi yang tidak diharapkan antara subsistem dengan interface subsistem. Proses ini juga berhubungan dengan validasi bahwa sistem telah memenuhi persyaratan fungsional dan nonfungsionalnya, dan sistem baru.

   Pada fase sebelumnya yaitu fase analisis yang mempunyai tujuan untuk memahami dengan sebenar-benarnya kebutuhan dari sistem baru dan mengembangkan sebuah sistem yang mewadahi requirement tersebut atau memutuskan bahwa sebenarnya pengembangan sistem baru tidak dibutuhkan. pada fase ini, seorang system analis menentukan keseluruhan requirement secara lengkap dan membagi kebutuhan sistem ke dalam 2 jenis, yaitu **Kebutuhan Fungsional (functional Requirement)**. Kebutuhan fungsional adalah jenis kebutuhan yang berisi proses-proses apa saja yang nantinya dilakukan oleh system. Kebutuhan fungsional juga berisi informasi-informasi apa saja yang harus ada dan dihasilkan oleh sistem. Jenis kedua adalah **Kebutuhan Non fungsional (Nonfunctional Requirements).** Requirement jenis ini adalah tipe requirement yang berisi properti perilaku yang dimiliki oleh sistem, meliputi:

   a. Operasional

Pada bagian ini harus dijelaskan teknis bagaimana system baru akan beroperasi. Platform sistem yang dipakai didefinisikan, apakah menggunakan windows atau Linux misalnya. Software untuk mengembangkan sistem juga ditentukan. Hardware spesifik yang diperlukan juga ditentukan. Terakhir arsitektur sistem juga dijelaskan apakah 2-tier, 3 –tier atau yang lainnya.

Contoh :

Digunakan pada system operasi Microsoft Windows XP®, Microsoft Windows® NT, Microsoft Windows®2000, Spesifikasi computer minimum Pentium III, Kebutuhan memori 128 MB – 256 MB RAM, Bisa dilengkapi barcode reader, Printer untuk mencetak kartu anggota dan laporan keuangan maupun yang lain-lain.

b. Performannce

Pada bagian ini dijelaskan seberapa bagus kinerja dari software yang dikembangkan dalam mengolah data, menampilkan informasi dan secara keseluruhan menyelesaikan proses bisnis yang ditanganinya. Efisiensi dari perangkat lunak juga dicantumkan.

Contoh :

Waktu untuk transaksi peminjaman buku dibatasi 2 menit, Waktu untuk transaksi pengembalian buku di batasi 1 menit.

c. Keamanan

Kebutuhan keamanan berisi pernyataan tentang mekanisme pengamanan aplikasi, data maupun transaksi yang akan diimplementasikan pada sistem. Sistem password yang digunakan akan seperti apa dan hardware spesifik untuk pengamanan sistem juga dideskripsikan.

Contoh :

Dilengkapi password untuk sistem aplikasinya maupun databasenya, Dilengkapi dengan kamera untuk mengawasi anggota yang membaca di ruang baca dan ruang penyimpanan tas yang tersambung ke komputer

d. Politik dan Budaya

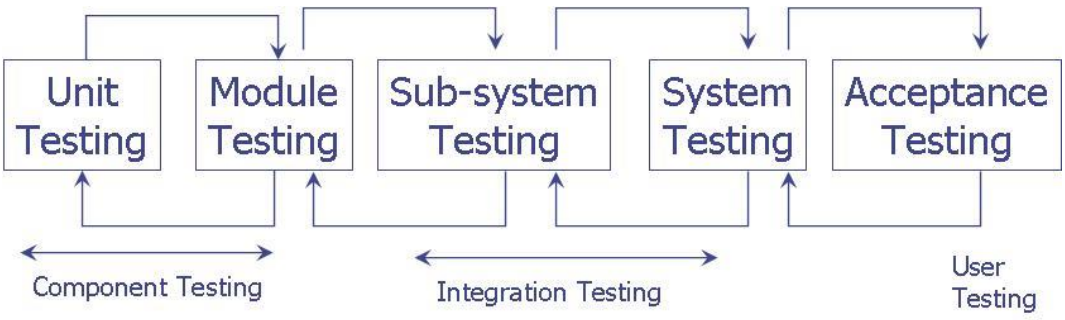
Requirement yang isinya menyangkut atau berhubungan dengan isu politik dan budaya ditentukan disini. Isi yang secara politik dan budaya harus dijamin tidak menimbulkan persepsi negatif terhadap sistem.

Contoh :

Sistem Informasi Perpustakaan harus bersifat tertutup dimana yang menginput data merupakan petugas perpustakaan sesuai kebijakan kampus.

5. Pengujian Penerimaan

   Ini merupakan tahap akhir proses pengujian sebelum sistem diterima untuk penggunaan operasional. Sistem diuji dengan data yang dipasok oleh pemakai sistem dan bukan data uji simulasi. Pengujian penerimaan bisa menunjukkan kesalahan dan mengharuskan definisi persyaratan sistem karena data riil menjalankan sistem dengan cara yang berbeda dari data uji. Pengujian penerimaan juga dapat mengungkap masalah persyaratan di mana fasilitas sistem tidak memenuhi keperluan pemakai atau kinerja sistem tidak dapat diterima.



Gambar 3 . Proses Pengujian (Sommerville, 2001)

C. Dokumentasi

Dokumentasi penting untuk dilaksanakan, karena dapat digunakan untuk penelusuran jika terjadi kesalahan. Dokumentasi merupakan kegiatan melakukan pencatatan terhadap setiap langkah pekerjan dari awal pembuatan program sampai akhir pembuatan program.

Berdasarkan pemakainya, dokumentasi dapat dibedakan menjadi tiga jenis, yaitu dokumentasi untuk programmer, dokumentasi untuk operator dan dokumentasi untuk pemakai.

Dokumentasi untuk programmer dapat dibedakan lagi menjadi dua jenis Internal dan Eksternal. Dokumentasi internal digunakan untuk menuntun pembuat program tersebut

untuk berinteraksi dengan bahasa pemrograman. Sedangkan dokumentasi eksternal merupakan penjelasan mengenai program tersebut, biasanya berupa manual dan catatan tentang program.

Dokumentasi operator berisi hal hal yang mendasar tentang bagaimana cara mengoperasikan program atau sistem tersebut. Informasi yang ada pada dokumentasi ini berbeda dengan informasi yang ada pada dokumentasi untuk programmer, mengingat seorang operator sistem tidak perlu tahu bagaimana sistem tersebut dibuat.

Yang lebih umim lagi adalah dokumentas untuk pemakai. Dokumentasi untuk pemakai biasanya berisi tentang informasi timbal balik antara pemakai dengan sistem atau interaksi pemakai dengan sistem tersebut.

PERTEMUAN II

PENGUJIAN DAN IMPLEMENTASI SISTEM

KUALITAS PERANGKAT LUNAK

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 2 (Dua)	Waktu :

Modul	2 (Dua)
Topik	Kualitas Perangkat Lunak
Sub Topik	-
Materi	• Konsep Kualitas • Jaminan Kualitas Perangkat Lunak • Kajian Perangkat Lunak • Kajian Teknik Formal • Realibilitas Perangkat Lunak • Standar Kualitas ISO
Tujuan	Mahasiswa akan dapat menerangkan kualitas dan resiko kualitas perangkat lunak

Jaminan kualitas perangkat lunak (Software Quality Assurance / SQA) adalah aktifitas pelindung yang diaplikasikan pada seluruh perangkat lunak.

SQA Meliputi :

1. Pendekatan Manajemen Kualitas

2. Teknologi rekayasa perangkat lunak yang efektif (metode dan piranti)
3. Kajian teknik formal yang diaplikasikan pada keseluruhan proses perangkat lunak
4. Strategi pengujian multitiered (deret bertingkat)
5. Kontrol dokumentasi perangkat lunak dan perubahan yang dibuat untuknya.
6. Prosedur untuk menjamin kesesuain dengan standar pengembangan perangkat lunak (bila dapat diaplikasikan)
7. Mekanisme pengukuran dan pelaporan

A. Konsep Kualitas

1. Kualitas

   American Heritage Dictionary mendefinisikan kata kualitas sebagai “Sebuah Karakteristik atau atribut dari sesuatu”. Sebagai atribut dari sesuatu, kualitas mengacu pada karakteristik yang dapat diukur, sesuatu yang dapat kita bandingkan dengan standar yang sudah diketahui, seperti panjang, warna, sifat kelistrikan, kelunakan dsb. Tetapi perangkat lunak, yang sebagian besar merupakan entitas intelektual, lebih menantang untuk dikarakterisasi daripada objek fisik.

   Pengukuran karakteristik program benar benar ada. Properti tersebut mencakup kompleksitas siklomatik (metrik perangkat lunak menunjukkan kompleksitas dari program), kohesi (kerapatan antar objek), jumlah function point, baris kode dan lain lain.

   Bila kita mengamati sebuah item dengan didasarkan pada sifat pengukurannya, ada dua jenis kualitas yang ada , yaitu **Kualitas Desain dan Kualitas Konformansi**.

   Kualitas desain mengacu pada karakteristik yang ditentukan oleh desainer terhadap suatu item tertentu. Nilai material, toleransi dan spesifikasi kinerha, semua memberikan kontribusi terhadap kualitas desain. Karena material dengan nilai yang lebih tinggi digunakan dan toleransi yang lebih ketat serta tingkat kinerja yang lebih baik ditentukan, maka kualitas desain dari suatu produk bertambah, bila produk dihasilkan sesuai dengan spesifikasi yang ditentukan.

   Kualitas konformansi adalah tingkat dimana spesifikasi desain terus diikuti selama pembuatan. Semakin tinggi tingkat konformansi, semakin tinggi tingkat kualitas Perangkat Lunak.

2. Kontrol Kualiatas

   Kontrol kualitas merupakan serangkaian pemeriksaan, kajian dan pengujian yang digunakan pada keseluruhan siklus pengembangan untuk memastikan bahwa setiap produk memenuhi persyaratan yang ditetapkan.

Kontrol kualitas mencakup loop (kalang) umpan balik pada proses yang menciptakan produk kerja. Kombinasi pengukuran dan umpan balik memungkinkan kita memperbaiki proses bila produk kerja yang diciptakan gagal memenuhi spesifikasi mereka. Pendekatan tersebut memandang kontrol kualitas sebagai bagian dari proses pemanufakturan

3. Jaminan Kualitas
   Jaminan kualitas terdiri atas fungsi auditing dan pelaporan manajemen. Tujuan jaminan kualitas untuk memberikan data yang diperlukan oleh manajemen untuk menginformasikan masalah kualitas produk, sehingga dapat memberikan kepastian dan konfidensi bahwa kualitas produk dapat memenuhi sasaran.

4. Biaya Kualitas
   Biaya kualitas menyangkut semua biaya yang diadakan untuk mengejar kualitas atau untuk menampilkan kualitas yang berhubungan dengan aktifitas. Studi tentang biaya kualitas dilakukan untuk memberikan garis dasar bagi biaya yang sedang digunakan, untuk mengidentifikasi kemungkinan pengurangan biaya serta memberikan basis perbandingan yang ternormalisasi.

   Biaya Kualitas dapat di bagi ke dalam biaya biaya yang dihubungkan dengan pencegahan, penilaian dan kegagalan.
   Biaya pencegahan meliputi :
   - Perencanaan kualitas
   - Kajian teknis formal
   - Perlengkapan pengujian
   - Pelatihan

   Biaya Penilaian meliputi aktivitas untuk memperoleh wawasan mengenai kondisi produk “pertama kali” pada masing masing proses.

   Contoh biaya penilaian meliputi :

   - Inspeksi in-Proses dan interproses
   - Pemeliharaan dan kalibrasi peralatan
   - Pengujian

   Biaya Kegagalan adalah biaya yang akan hilang bila tidak ada cacat yang muncul sebelum produk disampaikan kepada pelanggan. Biaya kegagalan dapat dibagilagi ke dalam biaya :

   1) Biaya Kegagalan Internal
      Adalah biaya yang diadakan bila kita mendeteksi suatu kesalahan dalam produk sebelum produk dipasarkan.
      Biaya kegagalan internal meliputi :

- Pengerjaan kembali
- Perbaikan
- Analisis Mode Kegagalan

2) Biaya Kegagalan Eksternal
   Adalah biaya yang berhubungan dengan cacat yang ditemukan setelah produk disampaikan kepada pelanggan.
   Biaya Kegagalan Eksternal meliputi :
   - Resolusi Keluhan
   - Penggantian dan Pengembalian Produk
   - Dukungan help Line
   - Kerja Jaminan

B. Jaminan Kualitas Perangkat Lunak

Kualitas perangkat lunak dapat didefinisikan sebagai :
Konformasi (Tingkat kesesuaian suatu produk terhadap hal yang telah ditetapkan) terhadap kebutuhan fungsional dan kinerja yang dinyatakan secara eksplisit, standar perkembangan yang didokumentasikan secara eksplisit, dan karakteristik implisit yang diharapkan bagi semua perangkat lunak yang dikembangkan secara profesional.

Definisi tersebut berfungsi untuk menekankan tiga hal penting, yaitu :

- Kebutuhan perangkat lunak merupakan pondasi yang melaluinya Kualitas diukur. Kurangnya penyesuaian terhadap kebutuhan juga menunjukkan rendahnya kualitas.
- Standar yang telah ditentukan menetapkan serangkaian kriteria pengembangan yang menuntun cara perangkat lunak direkayasa. Jika kriteria tersebut tidak diikuti, hampir pasti menimbulkan kualitas yang kurang baik.
- Ada serangkaian kebutuhan implisit yang sering tidak dicantumkan (misalnya kebutuhan akan kemampuan pemeliharaan yang baik). Bila perangkat lunak dapat berhasil menyesuaikan dengan kebutuhan eksplisitnya, tetapi gagal memenuhi kebutuhan implisitnya, maka kualitas perangkat lunak tersebut perlu diragukan.

Aktifitas SQA

Jaminan Kualitas perangkat lunak terdiri dari berbagai tugas yang berhubungan dengan dua konstituen yang berbeda, perekayasa perangkat lunak yang mengerjakan kerja teknis dan kelompok SQA yang bertanggung jawab terhadap perencanaan jaminan kualitas, kesalahan, penyimpanan rekaman, analisis dan pelaporan.

Tugas Kelompok SQA adalah membantu tim rekayasa perangkat lunak dalam pencapaian produk akhir yang berkualitas tinggi. The Software Engineering Institute

merekomendasikan serangkaian aktifitas SQA yang menekankan rencana jaminan kualitas, kesalahan, penyimpanan rekaman, analisis dan pelaporan.

Berikut ini akatifitas yang dilakukan (difasilitasi) oleh kelompok SQA yang independen :

1. Menyiapkan Rencana SQA untuk Suatu Proyek
2. Berpartisipasi dalam pengembangan deskripsi proses pengembangan proyek. Tim rekayasa perangkat lunak memilih sebuah proses bagi kerja yang akan dilakukan.
3. Mengkaji aktifitas rekayasa perangkat lunak untuk memverifikasi pemenuhan proses perangkat lunak yang sudah ditentukan. Kelompok SQA mengidentifikasi, mendokumentasi, dan menelusuri deviasi proses dan membuktikan apakah koreksi sudah dilakukan.
4. Mengaudit produk kerja perangkat lunak yang ditentukan untuk membuktikan kesesuaian dengan produk kerja yang ditentukan tersebut sebagai bagaian dari proses perangkat lunak.
5. Memastikan bahwa deviasi pada kerja dan produk kerja perangkat lunak didokumentasi dan ditangani sesuai prosedur pendokumentasian.
6. Mencatat ketidaksesuaian dan melaporkannya kepada manajemen senior. Item item yang tidak sesuai ditelursuri sampai item itu diubah.

C. Kajian Perangkat Lunak

Kajian perangkat lunak adalah suatu “filter” bagi proses rekayasa perangkat lunak, yaitu kajian yang diterapkan pada berbagai titik selama pengembangan perangkat lunak dan berfungsi untuk mencari kesalahan yang kemudian akan dihilangkan. Kajian perangkat lunak berfungsi untuk “memurnikan” produk kerja perangkat lunak yang terjadi sebagai hasil dari analisis, desain dan pengkodean.

D. Kajian teknik Formal

Kajian Teknik Formal (Formal Technique Research) adalah aktifitas jaminan kualitas perangkat lunak yang dilakukan oleh perekayasa perangkat lunak.
Tujuan FTR adalah :

- Menemukan kesalahan dalam fungsi, logika atau implementasinya dalam berbagai representasi perangkat lunak
- Membuktikan bahwa perangkat lunak disajikan sesuai dengan standar yang sudah ditentukan sebelumnya.
- Mencapai perangkat lunak yang dikembangkan dengan cara seragam.
- Membuat proyek lebih dapat dikelola.

Sebagai tambahan, FTR berfungsi sebagai dasar pelatihan yang memungkinkan perekayasa junior mengamati berbagai pendekatan yang berbeda terhadap analisis perangkat lunak, desain dan implementasi. FTR juga berfungsi untuk mengembangkan backup dan kontinuitas karena sejumlah orang mengenal baik bagian-bagian perangkat lunak yang tidak mereka ketahui sebelumnya.

E. Reliabilitas Perangkat Lunak

Tidak diragukan lagi bahwa reliabilitas sebuah program komputer merupakan suatu elemen yang penting. Bila sebuah program berkali-kali gagal untuk melakukan kinerja, maka sedikit meragukan apakah faktor kualitas perangkat lunak yang lain dapat diterima.

Reliabilitas perangkat lunak, tidak seperti faktor kualitas yang lain, dapat diukur, diarahkan, dan diestimasi dengan menggunakan data pengembangan historis. Reliabilitas perangkat lunak didefinisikan dalam bentuk statistik sebagai " Kemungkinan operasi program komputer bebas kegagalan di dalam suatu lingkungan tertentu dan waktu tertentu"

Contoh :
Program X diperkirakan memiliki reliabilitas 0.96 pada delapan jam pemrosesan yang dilalui. Dengan kata lain, jika program X akan dieksekusi 100 kali dan membutuhkan delapan jam waktu pemrosesan yang dilalui (waktu eksekusi), dia akan beroperasi dengan benar (tanpa kegagalan) 96 kali dari 100 kali pelaksanaan.

Keamanan Perangkat Lunak dan Analisis Resiko

Keamanan Perangkat lunak dan analisis resiko adalah aktifitas jaminan kualitas perangkat lunak yang berfokus pada identifikasi dan penilaian resiko potensial yang mungkin berpengaruh negatif terhadap perangkat lunak dan menyebabkan seluruh sistem menjadi gagal. Jika resiko dapat diidentifikasi pada awal proses rekayasa perangkat lunak, maka ciri-ciri desain perangkat lunak dapat ditetapkan sehingga akan mengeliminasi atau mengontrol resiko potensial.

F. Standar Kualitas ISO

Sistem jaminan kualitas dapat didefinisikan sebagai struktur, tanggung jawab, prosedur, proses dan sumber sumber daya organisasi untuk mengimplementasi manajeman kualitas.

ISO 9000 menjelaskan elemen jaminan kualitas dalam bentuk yang umum yang dapat diaplikasikan pada berbagai bisnis tanpa memandang produk dan jasa yang ditawarkan. Elemen elemen tersebut mencakup struktur, prosedur, proses, organbisasi dan sumber daya yang dibutuhkan untuk mengimplementasi rencana kualitas , kualitas kontrol, jaminan kualitas dan pengembangan kualitas.

Agar terdaftar dalam satu model sistem jaminan kualitas yang ada pada ISO 9000, sistem kualitas dan operasi perusahaan diperiksa oleh auditor untuk memeriksa kesesuaiannya dengan standar dan operasi efektif. Bila registrasi itu berhasil, perusahaan diberi sertifikasi dari bada registrasi yang diwakili oleh auditor. Audit pengawasan tengah tahunan terus dilakukan untuk memastikan kesesuaiannya dengan standar yang sudah ditetapkan.

STANDAR ISO 9001

ISO 9001 adalah standar jaminan kualitas yang berlaku untuk rekayasa perangkat lunak.

Standar tersebut berisi 20 syarat yang harus ada untuk mencapai sistem jaminan kualitas yang efektif , yaitu :

1. Tanggung jawab manajeman
2. Sistem Kualitas
3. Kajian Kontrak
4. Kontrol Desain
5. Kontrol data dan dokumen
6. Pembelian
7. Kontrol terhadap produk yang disuplai oleh pelanggan
8. Identifikasi dan kemampuan penelusuran produk
9. Kontrol Proses
10. Pemeriksaan dan pengujian
11. Kontrol Pemeriksaan, pengukuran dan perlengkapan pengujian
12. Pemeriksaan dan status pengujian
13. Kontrol ketidaksesuain produk
14. Tindakan preventif dan korektif
15. Penanganan, penyimpanan, pengepakan, preservasi dan penyampaian.
16. Kontrol terhadap catatan kualitas
17. Audit kualitas internal
18. Pelatihan
19. Pelayanan
20. Teknik statistik

Untuk dapat didaftar dalam ISO 9001, organisasi perangkat lunak harus membuat kebijakan dan prosedur yang memberi tekanan pada masing masing syarat tersebut dan kemudian dapat menunjukkan bahwa prosedur dan fungsi itu telah diikuti.

PERTEMUAN III

PENGUJIAN DAN IMPLEMENTASI SISTEM

KUALITAS PERANGKAT LUNAK

LANJUTAN

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 3 (Dua)	Waktu :

Modul	3 (Dua)
Topik	Kualitas Perangkat Lunak Lanjutan
Sub Topik	-
Materi	• Kriteria Penjaminan Kualitas Perangkat Lunak • Product Operations • Product Revision • Product Transition • Teknik Pengukuran
Tujuan	Mahasiswa akan dapat menerangkan kualitas dan resiko kualitas perangkat lunak

A. Kriteria Penjaminan Kualitas Perangkat Lunak

Dalam salah satu referensi disebutkan bahwa yang dimaksud dengan software quality adalah pemenuhan terhadap kebutuhan fungsional dan kinerja yang didokumentasikan secara eksplisit, pengembangan standar yang didokumentasikan secara eksplisit, dan sifat-sifat implisit yang diharapkan dari sebuah software yang dibangun secara profesional (Dunn, 1990). Berdasarkan definisi di atas terlihat bahwa sebuah software dikatakan berkualitas apabila memenuhi tiga ketentuan pokok :

1. Memenuhi kebutuhan pemakai – yang berarti bahwa jika software tidak dapat memenuhi kebutuhan pengguna software tersebut, maka yang bersangkutan dikatakan tidak atau kurang memiliki kualitas
2. Memenuhi standar pengembangan software – yang berarti bahwa jika cara pengembangan software tidak mengikuti metodologi standar, maka hampir dapat dipastikan bahwa kualitas yang baik akan sulit atau tidak tercapai.
3. Memenuhi sejumlah kriteria implisit – yang berarti bahwa jika salah satu kriteria implisit tersebut tidak dapat dipenuhi, maka software yang bersangkutan tidak dapat dikatakan memiliki kualitas yang baik.

McCall dan kawan-kawan pada tahun 1977 telah mengusulkan suatu penggolongan faktor-faktor atau kriteria yang mempengaruhi kualitas software. Pada dasarnya, McCall menitikberatkan faktor-faktor tersebut menjadi tiga aspek penting, yaitu yang berhubungan dengan :

1. Sifat-sifat operasional dari software (Product Operations).
2. Kemampuan software dalam menjalani perubahan (Product Revision)
3. Daya adaptasi atau penyesuaian software terhadap lingkungan baru (Product Transition)

B. Product Operations

Sifat-sifat operasional suatu software berkaitan dengan hal-hal yang harus diperhatikan oleh para perancang dan pengembang yang secara teknis melakukan penciptaan sebuah aplikasi. Hal-hal yang diukur di sini adalah yang berhubungan dengan teknis analisa, perancangan, dan konstruksi sebuah software. Faktor faktor McCall yang berkaitan dengan sifat-sifat operasional software adalah:

1. Correctness – sejauh mana suatu software memenuhi spesifikasi dan mission objective dari users.
2. Reliability – sejauh mana suatu software dapat diharapkan untuk melaksanakan fungsinya dengan ketelitian yang diperlukan.
3. Efficiency – banyaknya sumber daya komputasi dan kode program yang dibutuhkan suatu software untuk melakukan fungsinya
4. Integrity – sejauh mana akses ke software dan data oleh pihak yang tidak berhak dapat dikendalikan.
5. Usability – usaha yang diperlukan untuk mempelajari, mengoperasikan, menyiapkan input, dan mengartikan output dari software.

C. Product Revision

Setelah sebuah software berhasil dikembangkan dan diimplementasikan, akan terdapat berbagai hal yang perlu diperbaiki berdasarkan hasil uji coba maupun evaluasi. Sebuah software yang dirancang dan dikembangkan dengan baik, akan dengan mudah dapat direvisi jika diperlukan. Seberapa jauh software tersebut dapat diperbaiki merupakan

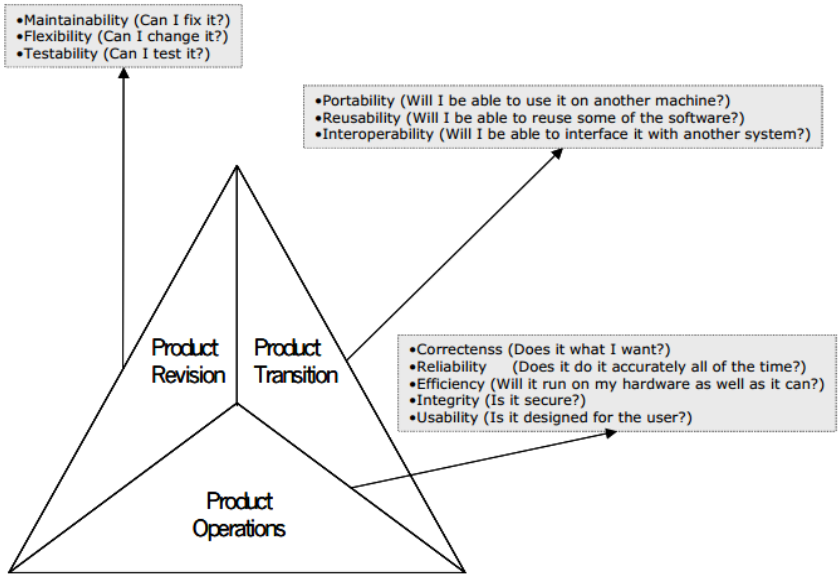
faktor lain yang harus diperhatikan. Faktor-faktor McCall yang berkaitan dengan kemampuan software untuk menjalani perubahan adalah:

1. Maintainability – usaha yang diperlukan untuk menemukan dan memperbaiki kesalahan (error) dalam software.
2. Flexibility – usaha yang diperlukan untuk melakukan modifikasi terhadap software yang operasional.
3. Testability – usaha yang diperlukan untuk menguji suatu software untuk memastikan apakah melakukan fungsi yang dikehendaki atau tidak.

D. Product Transition

Setelah integritas software secara teknis telah diukur dengan menggunakan faktor product operational dan secara implementasi telah disesuaikan dengan faktor product revision, faktor terakhir yang harus diperhatikan adalah faktor transisi – yaitu bagaimana software tersebut dapat dijalankan pada beberapa platform atau kerangka sistem yang beragam. Faktor-faktor McCall yang berkaitan dengan tingkat adaptibilitas software terhadap lingkungan baru.

1. Portability – usaha yang diperlukan untuk mentransfer software dari suatu hardware dan/atau sistem software tertentu agar dapat berfungsi pada hardware dan/atau sistem software lainnya.
2. Reusability – sejauh mana suatu software (atau bagian software) dapat dipergunakan ulang pada aplikasi lainnya.
3. Interoperability – usaha yang diperlukan untuk menghubungkan satu software dengan lainnya.



Sumber : Pressman, 1994

E. Teknik Pengukuran

Menimbang tingkat kesulitan yang dihadapi para programmer dalam mengukur secara langsung dan kuantitatif kualitas software yang dikembangkan berdasarkan pembagian yang diajukan McCall di atas, sebuah formula diajukan untuk mengukur faktor-faktor software quality secara tidak langsung menurut hubungan :

$$F_q = c_1 * m_1 + c_2 * m_2 + c_3 * m_3 + \ldots + c_n * m_n$$

dimana :
F_q = Faktor software quality
c_1 = Bobot yang bergantung pada produk dan kepentingan
m_1 = Metric yang mempengaruhi faktor software quality

Adapun metric yang dipakai dalam skema pengukuran di atas adalah sebagai berikut:

- ✓ Auditability – kemudahan untuk memeriksa apakah software memenuhi standard atau tidak.
- ✓ Accuracy – ketelitian dari komputasi dan kontrol.
- ✓ Communication Commonality – sejauh mana interface, protokol, dan bandwidth digunakan.
- ✓ Completeness – sejauh mana implementasi penuh dari fungsi-fungsi yang diperlukan telah tercapai.
- ✓ Conciseness – keringkasan program dalam ukuran LOC (line of commands).
- ✓ Consistency – derajat penggunaan teknik-teknik desain dan dokumentasi yang seragam pada seluruh proyek pengembangan software.
- ✓ Data Commonality – derajat penggunaan tipe dan struktur data baku pada seluruh program.
- ✓ Error Tolerance – kerusakan yang terjadi apabila program mengalami error.
- ✓ Execution Efficiency – kinerja run-time dari program.
- ✓ Expandability – sejauh mana desain prosedur, data, atau arsitektur dapat diperluas.
- ✓ Generality – luasnya kemungkinan aplikasi dari komponen-komponen program.

- ✓ Hardware Independence – sejauh mana software tidak bergantung pada kekhususan dari hardware tempat software itu beroperasi.
- ✓ Instrumentation – sejauh mana program memonitor operasi dirinya sendiri dan mengidentifikasi error yang terjadi.
- ✓ Modularity – functional independence dari komponen-komponen program.
- ✓ Operability – kemudahan mengoperasikan program.
- ✓ Security – ketersediaan mekanisme untuk mengontrol dan melindungi program dan data terhadap akses dari pihak yang tidak berhak.
- ✓ Self-Dokumentation – sejauh mana source-code memberikan dokumentasi yang berarti.
- ✓ Simplicity – Kemudahan suatu program untuk dimengerti.
- ✓ Traceability – kemudahan merujuk balik implementasi atau komponen program ke kebutuhan pengguna software.
- ✓ Training – sejauh mana software membantu pemakaian baru untuk menggunakan sistem.

PERTEMUAN IV

PENGUJIAN DAN IMPLEMENTASI SISTEM

PERENCANAAN PENGUJIAN (TEST PLAN)

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 4 (Empat) Waktu :

Modul	4 (Empat)
Topik	Perencanaan Pengujian (Test Plan)
Sub Topik	-
Materi	• Strategi Testing • Pendekatan Strategi Testing • Integation Testing • Top Down Integration • Bottom Up Testing • Regression Testing • Smoke Testing • Komentar Integraton Testing
Tujuan	Mahasiswa akan dapat menghasilkan perencanaan pengujian (test plan) suatu proyek pengujian perangkat lunak.

A. Strategi Testing

Strategi testing software mengintegrasikan metode metode disain test cases software ke dalam suatu rangkaian tahapan yang terencana dengan baik sehingga pengembangan software dapat berhasil.

Strategi menyediakan peta yang menjelaskan tahap-tahap yang harus dilakukan sebagai bagian dari testing, dan membutuhkan usaha, waktu, dan sumber daya bilamana tahap-tahap ini direncanakan dan dilaksanakan.

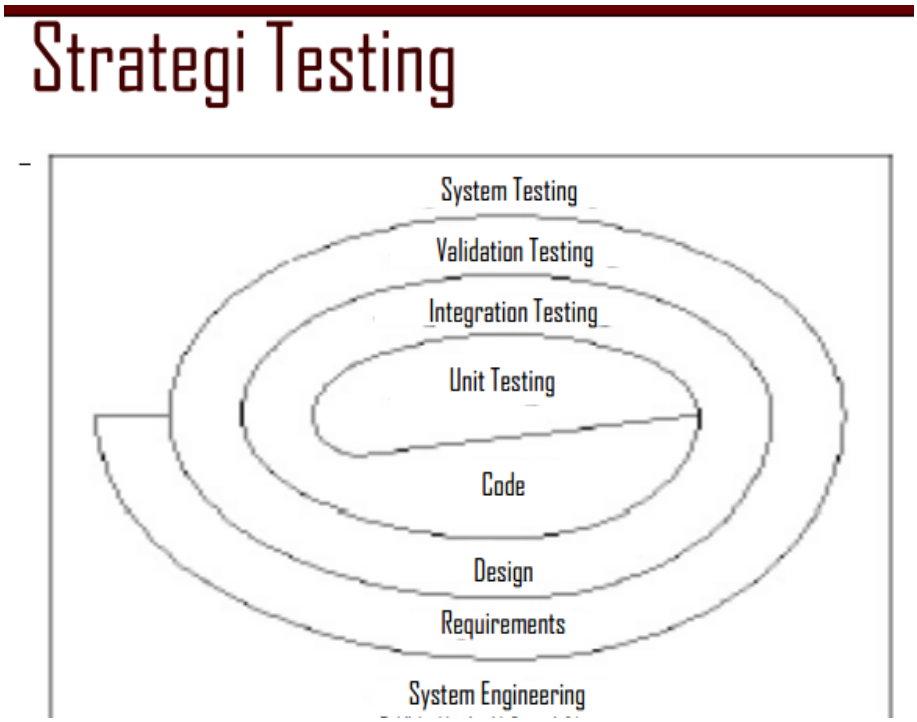
Strategi testing harus menjadi satu kesatuan dengan perencanaan tes, disain test case, ekesekusi tes, dan pengumpulan serta evaluasi data hasil testing.

Strategi testing software harus cukup fleksibel untuk dapat mengakomodasi kustomisasi pendekatan testing. Pada saat yang bersamaan, harus juga cukup konsisten dan tegas agar dapat melakukan perencanaan yang masuk akal dan dapat melakukan manajemen perkembangan kinerja proyek

B. Pendekatan Strategi Testing

Sejumlah strategi testing software diadakan untuk menyediakan kerangka testing bagi pengembang software dengan karekteristik umum sebagai berikut:

1. Testing dimulai dari tingkat komponen terkecil sampai pada integrasi antar komponen pada keseluruhan sistem komputer tercapai.
2. Teknik testing berbeda-beda sesuai dengan waktu penggunaannya.
3. Testing dilakukan oleh pengembang software dan (untuk proyek besar) dilakukan oleh suatu grup tes yang independen.
4. Testing dan debugging adalah aktifitas yang berlainan, tapi debugging harus diakomodasi disetiap strategi testing.



C. Integration Testing

"Jika semua modul-modul software telah bekerja dengan baik secara individual, mengapa harus ada keraguan apakah modul-modul tersebut dapat bekerja sama sebagai satu kesatuan?".

Integration testing adalah suatu teknik yang sistematis untuk pembangunan struktur program, dimana pada saat yang bersamaan melakukan testing untuk mendapatkan errors yang diasosiasikan dengan antar-muka.

Obyektifitasnya adalah untuk menindaklanjuti komponen-komponen yang telah melalui unit testing dan membangun suatu struktur program sesuai dengan disain yang telah dituliskan sebelumnya.

Terdapat kecenderungan untuk melakukan integrasi yang tidak secara bertahap, yaitu dengan menggunakan suatu pendekatan "Big Bang".

Pendekatan ini menggabungkan komponen komponen secara bersamaan hingga terbentuk suatu program.

Testing dilakukan pada keseluruhan program secara bersamaan.

Kekacauan cenderung terjadi, karena :

1. Sekumpulan errors akan diperoleh, dan perbaikan sulit dilakukan, karena terjadi komplikasi saat melakukan isolasi terhadap penyebab masalah.
2. Ditambah lagi dengan munculnya errors baru saat errors sebelumnya dibenahi, sehingga menciptakan suatu siklus yang tak ada hentinya.

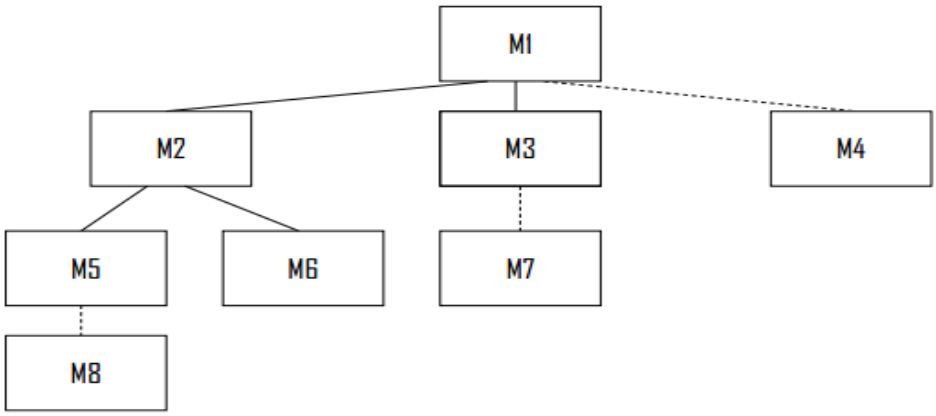
Integrasi yang dilakukan secara bertahap merupakan lawan dari penggunaan strategi "Big Bang".

Program dikonstruksi dan dites secara bertahap, meningkat sedikit demi sedikit, dimana bila terjadi errors dapat dengan mudah untuk diisolasi dan diperbaiki, antarmuka dapat dites secara komplit atau paling tidak mendekati komplit, serta pendekatan tes yang sistematis dapat digunakan.

D. Top Down Integration

Adalah pendekatan bertahap untuk menyusun struktur program. Modul-modul diintegrasikan dari atas ke bawah dalam suatu hirarki kendali, dimulai dari modul kendali utama (program utama).

Modul sub-ordinat dari modul kendali utama dihubungkan ke struktur yang paling dalam (depth-first integration) atau yang paling luas (breadth-first integration) dahulu.



- Depth-first integration, akan mengintegrasikan semua komponenkomponen pada struktur jalur kendali mayor. Misal dipilih sisi kiri terlebih dahulu, maka komponen M1, M2, M5 akan diintegrasikan dahulu, baru kemudian M8 atau M6 akan diintegrasikan
- Breadth-first integration, akan mengintegrasikan semua komponen secara langsung ke tiap tingkat, bergerak secara horisontal. Contoh komponen M2, M3 dan M4 akan diintegrasikan dahulu, kemudian baru M5 dan M6 dan seterusnya.

Lima langkah proses integrasi :

1. Modul kendali utama digunakan sebagai driver tes dan stubs tes disubtitusikan bagi semua komponen yang secara langsung menjadi sub-ordinat modul kendali utama.
2. Tergantung pada pendekatan integrasi yang dipilih, stubs subordinat digantikan dengan komponen sebenarnya.
3. Tes dilakukan saat tiap komponen diintegrasikan.
4. Saat pemenuhan tiap tes, stubs lainnya digantikan dengan komponen sebenarnya.
5. Testing regresi dilakukan untuk memastikan kesalahan baru tidak terjadi lagi.

Proses berlanjut dari langkah 2 sampai keseluruhan struktur program dilalui.

Strategi top-down integration melakukan verifikasi kendali mayor atau titik-titik keputusan di awal proses testing. Pada suatu struktur program yang difaktorkan dengan baik, pengambilan keputusan terjadi di tingkat atas dalam hirarki dan oleh sebab itu diperhitungkan terlebih dahulu. Jika kendali mayor bermasalah, dibutuhkan pengenalan

awal. Jika depth-first integration dipilih, suatu fungsi komplit dari software akan diimplementasikan dan didemonstrasikan.

Pendekatan ini terlihat tidak komplek, namun pada kenyataannya, masalah logistik akan timbul, karena proses level bawah dari hirarki dibutuhkan untuk tes level di atasnya. Stubs menggantikan modul level bawah saat dimulainya top-down testing; karenanya tidak ada data yang mengalir ke atas dari struktur program.

Tester hanya mempunyai 3 pilihan :

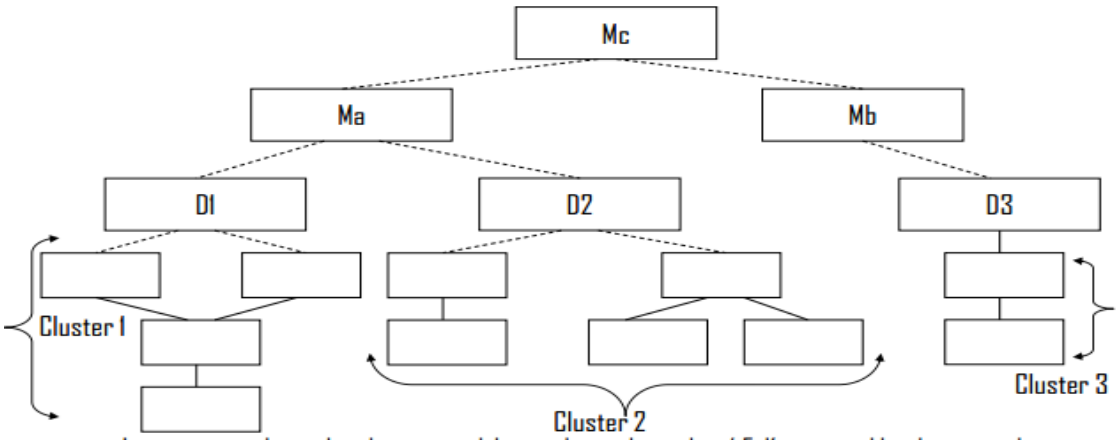
1. Tunda kebanyakan tes sampai stubs digantikan dengan modul sebenarnya, hal ini menyebabkan hilangnya beberapa kendali yang berhubungan antar tes tertentu dan modul tertentu. Tentunya akan menyulitkan untuk menentukan penyebab errors dan kecenderungan terjadi pelanggaran terhadap batasan-batasan dari pendekatan topdown.
2. Kembangkan stubs yang mempunyai fungsi terbatas untuk mensimulasikan modul sebenarnya, mungkin dapat dilakukan, namun akan menambah biaya overhead dengan semakin kompleknya stubs.
3. Integrasikan software dari bawah ke atas dalam hirarki, disebut sebagai bottom-up integration.

E. Bottom Up Testing

Sesuai namanya, integrasi ini dimulai dari modul terkecil. Karena komponenkomponen diintegrasikan dari bawah ke atas, sub-ordinat untuk tingkat bersangkutan dari komponen selalu diperlukan untuk diproses, dan kebutuhan terhadap stubs dapat dihilangkan.

Langkah-langkah strategi ini adalah :

1. Komponen level bawah dikombinasikan dalam clusters (kadang disebut builds) yang mewakili sub-fungsi software tertentu.
2. Driver ditulis untuk koordinasi masukan dan keluaran test case.
3. Cluster dites.
4. Driver dihapus dan cluster dikombinasikan, bergerak ke atas di dalam struktur program



- Integrasi mengikuti pola sebagaimana diilustrasikan pada gambar 4.5. Komponen dikombinasi untuk membentuk cluster 1, 2 dan 3.
- Tiap cluster dites dengan menggunakan driver. Komponen pada cluster 1 dan 2 adalah sub ordinat Ma Driver D1 dan D2 dihilangkan dan cluster dihubungkan langsung ke Ma, demikian seterusnya.
- Saat integrasi semakin bergerak ke atas, kebutuhan akan test drivers yang terpisah juga akan semakin sedikit. Pada kenyataannya, jika dua level atas dari struktur program diintegrasikan dari atas ke bawah, jumlah drivers akan banyak dikurangi, dan integrasi dari clusters akan sangat disederhanakan.

F. Regression Testing

Bilamana suatu hasil tes (jenis apapun) berhasil dalam menemukan errors, dan errors harus dikoreksi. Saat software dikoreksi, beberapa aspek dari konfigurasi software (program, dokumentasi, atau data pendukung) diubah.

Regression testing adalah aktivitas yang membantu untuk memastikan bahwa perubahan-perubahan yang terjadi tealah benar dan tidak menimbulkan tingkah laku yang tidak diinginkan atau penambahan errors.

Regression testing dapat dilakukan secara manual, dengan mengeksekusi kembali suatu subset dari keseluruhan test cases atau menggunakan alat bantu otomasi capture / playback.

Alat bantu capture / playback memungkinkan teknisi software untuk merekam test cases dan hasil-hasilnya untuk keperluan dipakai kembali dan dibandingkan pada sub sekuen tertentu atau keseluruhan.

Sub set tes yang dieksekusi terdiri dari 3 kelas test case yang berbeda :

1. Representasi dari contoh tes yang akan memeriksa semua fungsi software.
2. Tes tambahan yang berfokus pada fungsi software yang mungkin dipengaruhi oleh perubahan.
3. Tes yang berfokus pada komponen software yang diubah.

Saat tes integrasi dilakukan, jumlah tes regresi akan meningkat menjadi cukup besar.

Oleh karena itu, tes regresi seharusnya didisain untuk mencakup hanya pada tes-tes yang sama atau beberapa kelas errors di dalam setiap fungsi fungsi mayor program. Adalah tidak praktis dan tidak efisien untuk mengeksekusi kembali setiap tes untuk setiap fungsi program saat suatu perubahan terjadi.

G. Smoke Testing

Smoke testing adalah pendekatan integration testing yang sering digunakan ketika produk software “kecil terbatas” dibuat.

Didisain sebagai mekanisme untuk menghadapi kritisnya waktu dari suatu proyek, memungkinkan tim software untuk menjalankan proyeknya dengan suatu basis frekuensi.

Secara mendasar, pendekatan smoke testing terdiri dari aktifitasaktivitas berikut :

1. Komponen software yang telah ditranslasikan ke kode, diintegrasikan ke “build”, yang terdiri dari semua file data, pustaka, modul yang digunakan lagi, dan komponen yang dikembangkan yang dibutuhkan untuk menerapkan satu atau lebih fungsi produk.
2. Serangkaian tes didisain untuk menghasilkan kesalahan yang akan membuat “build” tetap berfungsi sebagaimana mestinya. Intensi harus mencakup “show stopper” kesalahan yang mempunyai kemungkinan terbesar membuat proyek software mengalami keterlambatan dari jadual.

3. “Build” diintegrasikan dengan “build” lainnya dan keseluruhan produk yang dilakukan smoke tes harian. Pendekatan integrasi dapat top-down atau bottom-up.

Bagaimana pun, tes yang sering dilakukan ini akan memberikan penilaian yang realistis dari proses kerja integration testing bagi manajer dan praktisi.

Menurut Mc Connell [MCO96], smoke tes adalah sebagai berikut: “Smoke tes harus memeriksa keseluruhan sistem dari akhir ke akhir. Tidak perlu terlalu lengkap, namun mampu menampilkan masalah mayor. Smoke tes harus cukup sistematis menjalankan “build”, dapat diasumsikan bahwa ia cukup stabil untuk melakukan tes yang cukup dengan sistematis.”

Smoke testing dapat dikarakteristikan sebagai suatu strategi integrasi yang berputar.

Software dibangun ulang (dengan komponen-komponen baru yang ditambahkan) dan diperiksa setiap hari.

Smoke testing menyediakan sejumlah keuntungan bila digunakan pada suatu proyek rekayasa yang mempunyai waktu kritis dan komplek, sebagai berikut:

1. Meminimalkan resiko integrasi. Karena dilakukan per hari, ketidakcocokan dan “show stopper” errors lainnya dapat dilihat per hari, sehingga terjadinya perubahan jadual akibat terjadinya errors serius dapat dikurangi.
2. Meningkatnya kualitas produk akhir. Karena pendekatan berorientasi integrasi, smoke tes melingkupi kesalahan fungsional dan arsitektural dan kesalahan disain tingkat komponen. Kesalahan ini dapat dibenahi secepatnya, kualitas yang lebih baik didapatkan.
3. Diagnosa kesalahan dan koreksi disederhanakan. Seperti halnya semua pendekatan integration testing, kesalahan yang ditemukan selama smoke testing diasosiasikan dengan “peningkatan software baru”, dimana software telah ditambahkan “build” yang mungkin menyebabkan ditemukannya kesalahan baru.
4. Penilaian proses kerja lebih mudah. Dengan lewatnya hari, lebih banyak software telah diintegrasi dan lebih banyak yang telah didemonstrasikan bekerja. Hal ini meningkatkan moral tim dan memberi manajer indikasi bagus bahwa proses kerja telah dilaksanakan.

H. Komentar Integration Testing

Telah banyak diskusi (seperti [BEI84]) tentang keunggulan dan kelemahan relatif dari topdown dibanding dengan bottom-up integration testing.

Secara umum, keunggulan satu strategi cenderung menghasilkan kelemahan bagi strategi lainnya.

Kelemahan utama dari pendekatan top-down adalah membutuhkan stubs dan kesulitankesulitan testing yang terjadi sehubungan dengannya.

Masalah-masalah yang diasosiasikan dengan stubs akan dapat diimbangi oleh keunggulan testing fungsi-fungsi kendali mayor lebih awal.

Kelemahan utama bottom-up integration adalah dimana "program sebagai entitas tidak akan ada sampai modul yang terakhir ditambahkan" [MYE79].

Kelemahan ini diimbangi dengan kemudahan dalam melakukan disain test cases dan kurangnya pemakaian stubs.

Pemilihan strategi integrasi tergantung pada karakteristik software, dan kadangkala pada jadual proyek

Pada umumnya pendekatan kombinasi (kadang disebut sandwich testing) yang menggunakan top-down integration testing untuk tingkat yang lebih atas dari suatu struktur program, digabung dengan bottomup integration testing untuk tingkat subordinat adalah hal yang terbaik.

Bila integration testing dilakukan, tester harus mengidentifikasi
modul-modul yang kritis.

Karakteristik dari modul kritis:

- Bergantung pada beberapa kebutuhan software.

- Mempunyai tingkat kendali yang tinggi.
- Mempunyai kompleksitas tinggi (digunakan cyclomatic complexity sebagai indikator).
- Mempunyai kebutuhan kinerja tertentu yang didefinisikan.

Modul-modul kritis harus dites sedini mungkin. Sebagai tambahan, regression testing harus berfokus pada fungsi modul yang kritis

PERTEMUAN V

PENGUJIAN DAN IMPLEMENTASI SISTEM

TEST SYSTEM ARCHITECTURE, CASES & COVERAGE

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 5 (Lima)	Waktu :

Modul	5 (Lima)
Topik	Test System Architecture , Cases & Coverage
Sub Topik	-
Materi	• Sasaran Pengujian • Prinsip Pengujian • Testabilitas • Desain Test Case • Pengujian White Box • Pengujian Basis Path • Pengujian Struktur Kontrol • Pengujian Black Box • Pengujian Untuk Aplikasi dan Lingkungan Khusus
Tujuan	Mahasiswa dapat menerangkan Test System Architecture, Cases, & Coverage.

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain dan pengkodean. Pengujian menyajikan anomali yang menarik bagi perekayasa perangkat lunak. Pada proses perangkat lunak, perekayasa pertama-tama berusaha membangun perangkat lunak dari konsep abstrak ke implementasi yang dapat dilihat, baru dilakukan pengujian. Perekayasa menciptakan sederetan *test case* yang dimaksudkan untuk "membongkar" perangkat lunak yang sudah dibangun. Pada dasarnya, pengujian merupakan satu langkah dalam proses rekayasa perangkat lunak yang dapat dianggap (paling tidak secara psikologis) sebagai hal yang destruktif daripada konstruktif.

1.1. Sasaran-sasaran pengujian

Beberapa sasaran pengujian diantaranya :

1. Pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan
2. Test case yang baik adalah test case yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya
3. Pengujian yang sukses adalah pengujian yang mengungkap semua kesalahan yang belum pernah ditemukan sebelumnya.

Sasaran tersebut mengimplikasikan adanya perubahan titik pandang yang dramatis. Sasaran itu berlawanan dengan pandangan yang biasanya dipegang yang menyatakan bahwa pengujian yang berhasil adalah pengujian yang tidak ada kesalahan yang ditemukan. Sasarannya adalah mendesain pengujian yang secara sistematis mengungkap kelas kesalahan yang berbeda dan melakkukannya dengan jumlah waktu dan usaha minimum.

Bila pengujian dilakukan secara sukses (sesuai dengan sasaran tersebut), maka akan ditemukan kesalahan di dalam perangkat lunak. Sebagai keuntungan sekunder, pengujian menunjukkan bahwa fungsi perangkat lunak bekerja sesuai dengan spesifikasi dan bahwa persyaratan kinerja telah dipenuhi. Sebagai tambahan, data yang dikumpulkan pada saat pengujian dilakukan memberikan indikasi yang baik mengenai reliabilitas perangkat lunak dan beberapa menunjukkan kualitas perangkat lunak secara keseluruhan.

1.2. Prinsip Pengujian

Prinsip Pengujian meliputi :

- ***Semua pengujian harus dapat ditelusuri sampai ke persyaratan pelanggan***. Sebagaimana telah kita ketahui, sasaran pengujian perangkat lunak adalah untuk mengungkapkan kesalahan. Hal ini memenuhi kriteria bahwa cacat yang paling fatal (dari titik pandang pelanggan) adalah cacat yang menyebabkan program gagal memenuhi persyaratannya.
- ***Pengujian harus direncanakan lama sebelum pengujian itu mulai*.** Perencanaan pengujian dapat dimulai segera setelah model pernyaratan dilengkapi. Definisi detail mengeani test case dapat dimulai segera setelah model desain diteguhkan. Dengan demikian, semua pengujian dapat direncanakan dan dirancang sebelum semua kode dibangkitkan.
- ***Prinsip Pareto berlaku untuk pengujian perangkat lunak.*** Secara singkat prinsip Pareto mengimplikasikan bahwa 80 persen dari semua kesalahan yang ditemukan selama pengujian sepertinya akan dapat ditelusuri sampai 20 persen dari semua modul program. Masalahnya, bagaimana mengisolasi modul yang dicurigai dan mengujinya dengan teliti.
- ***Pengujian harus mulai "dari yang kecil" dan berkembang ke pengujian "yang besar".*** Pengujian pertama yang direncanakan dan dieksekusi biasanya berfokus pada modul program individual. Selagi pengujian berlangsung maju, pengujian mengubah fokus dalam

usaha menemukan kesalahan pada cluster modul yang terintegrasi dan akhirnya pada sistem secara keseluruhan.

- ***Pengujian yang mendalam tidak mungkin.*** Jumlah jalur permutasi untuk program yang berukuran menengah-pun sangat besar. Karena itulah tidak mungkin untuk mengeksekusi setiap kombinasi jalur skema pengujian. Tetapi dimungkinkan untuk secara tepat mencakup logika program dan memastikan bahwa semua kondisi dalam desain prosedural telah diuji.
- ***Untuk menjadi paling efektif, pengujian harus dilakukan oleh pihak ketiga yang independent.*** Yang dimaksud dengan kata "yang paling efektif" adalah pengujian yang memiliki probabilitas tertinggi di dalam menemukan kesalahan (sasaran utama pengujian). Karena perekayasa perangkat lunak yang membuat sistem tersebut bukanlah orang yang paling tepat untuk melakukan semua pengujian bagi perangkat lunak.

1.3. Testabilitas

Testabilitas perangkat lunak adalah seberapa mudah sebuah program komputer dapat diuji. Karena pengujian sangat sulit, perlu diketahui apa yang dapat dilakukan untuk membuatnya menjadi mudah. Kadang-kadang pemrogram bersedia melakukan hal-hal yang akan membantu proses pengujian dan checklist mengenai masalah-masalah desain yang mudah, fitur dan lain sebagainya yang berguna dalam bernegosiasi dengan mereka.

Checklist berikut memberikan serangkaian karakteristik yang membawa kepada perangkat lunak yang dapat diuji :

- **Operabilitas**. "Semakin baik dia bekerja, semakin efisien dia dapat diuji"
  - Sistem memiliki beberapa bug (bug menambah analisis dan biaya pelaporan ke proses pengujian).
  - Tidak ada bug yang memblok eksekusi pengujian
  - Produk berkembang di dalam tahapan fungsional (memungkinkan pengemabngan dan pengujian secara simultan)
- **Observabilitas**. "Apa yang Anda lihat adalah apa yang Anda uji"
  - Output yang berbeda dikeluarkan oleh masing-masing input
  - Tahap dan variabel sistem dapat dilihat atau diantrikan selama eksekusi.
  - Sistem dan variabel yang lalu dapat dilihat atau diantrikan (misal : log transaksi)
  - Semua faktor yang mempengaruhi output dapat dilihat
  - Kesalahan itnernal dideteksi secara otomatis melalui mekanisme selftesting
  - Kesalahan internal dilaporkan secara otomatis
  - Kode sumber dapat diakses
- **Kontrolabilitas.** "Semakin baik kita dapat mengontrol perangkat luank, semakin banyak pengujian yang dapat diotomatisasi dan dioptimalkan"
  - Semua output yang mungkin dapat dimunculkan melalui beberapa kombinasi input
  - Semua kode dapat dieksekusi melalui berbagai kombinasi input
  - Keadaan dan varibale perangkat lunak dan perangkat keras dapat dikontrol secara langsung oleh perekayasa pengujian
  - Format input dan output konsistem dan terstruktur
  - Pengujian dapat dispesifikasi, dioptimasi dan direproduksi dengan baik
- **Dekomposabilitas**. "Dengan mengontrol ruang lingkup pengujian, kita dapat dengan lebih cepat mengisolasi masalah dan melakuakn pengujian kembali secara lebih halus"
  - Sistem perangkat luank dibangun dari modul-modul independen
  - Modul-modul perangkat lunak dapat diuji secara independen
- **Kesederhanaan**. "Semakin sedikit yang diuji, semakin cepat kita dapat mengujinya"
  - Kesederhanaan fungsional (seperti, kumpulan fitur adalah kebutuhan minimum untuk memenuhi persyaratan).

    - Kesederhanaan struktural (seperti, arsitektur dimodularisasi untuk membatasi penyebaran kesalahan)
    - Kesederhanaan kode (seperti, standar pengkodean diadopsi demi kemduahan inspeksi dan pemeliharaan).
- **Stabilitas**. "Semakin sedikti perubahan, semakin sedikit ganggunan dalam pengujian"
    - Pengujian ke perangkat lunak tidak sering
    - Perubahan ke perangkat lunak terkontrol
    - Perubahan ke perangkat lunak memvalidasi pengujian yang sudah ada
    - Kegagalan perangkat lunak dapat diperbaiki dengan baik
- **Kemampuan untuk dapat dipahami**. "Semakin banyak informasi yang kita miliki, semakin halus pengujian yang akan dilakukan"
    - Desain dipahami dengan baik
    - Ketergantungan di antara komponen internal, eksternal dan yang dipakai bersama, dipahami dengan baik.
    - Perubahan ke desain dikomunikasikan
    - Dokumentasi teknik dapat diakses dengan cepat
    - Dokumentasi teknis diorganisasikan dengan baik
    - Dokumentasi teknis spesifik dan detail
    - Dokumentasi teknis akurat

Berikut adalah atribut-atribut pengujian yang "baik" :

1. ***Pengujian yang baik memiliki probabilitas yang tinggi untuk menemukan kesalahan.*** Untuk mencapai hal ini, penguji harus memahami perangkat lunak dan berusaha mengembangkan gambaran mental mengenai bagaimana perangkat lunak dapat gagal. Idealnya kelas-kelas kegagalan itu diselidiki. Sebagai contoh, kelas kegagalan potensial pada GUI adalah kegagalan untuk mengenali posisi mouse yang sesuai. Serangkaian pengujian akan dirancang untuk menguji mouse untuk memperlihatkan kesalahan di dalam pengenalan posisi mouse.
2. ***Pengujian yang baik tidak redudan****.* Waktu pengujian dan sumber daya terbatas. Tidak ada manfaatnya melakukan pengujian dengan tujuan yang sama dengan pengujian lainnya. Setiap pengujian arus memiliki tujuan yang berbeda.
3. ***Pengujian yang baik seharusnya "jenis terbaik".*** Dalam suatu kelompok pengujian yang memiliki tujuan yang serupa, batasan waktu dan sumber daya dapat menghalangi eksekusi hanya kelompok kecil dari pengujian tersebut. Pada kasus semacam ini maka pengujian yang memiliki kemungkinan paling besar untuk mengungkap seluruh kelas kesalahan yang tinggi harus digunakan.
4. ***Pengujian yang baik tidak boleh terlalu sederhana atau terlalu kompleks****.* Secara umum masing-masing test case harus dieksekusi secara terpisah.

1.4. Desain Test Case

Lebih dari dua dekade terakhir telah berkembang berbagai metode desai test case untuk perangkat lunak. Metode-metode tersebut memberikan kepada pengembang sebuah pendekatan yang sistematik ke pengujian. Yang lebih penting, metode itu memberikan mekanisme yang dapat membantu memastikan kelengkapan pengujian dan memberikan kemungkinan tertinggi untuk

mengungkap kesalahan pada perangkat lunak.

Semua produk yang direkayasa dapat diuji dengan satu atau dua cara : (1) **dengan mengetahui fungsi yang ditentukan dimana produk dirancang untuk melakukannya**, pengujian dapat dilakukan untuk memperlihatkan bahwa masing-masing fungsi beroperasi sepenuhnya, pada waktu yang sama mencari kesalahan pada setiap fungsi; (2) **mengetahui kerja internal suatu produk**, maka pengujian dapat dilakukan untuk memastikan bahwa "semua roda gigi berhubungan", yaitu operasi internal bekerja sesuai dengan spesifikasi dan semua komponen ineternal telah diamati dengan baik. Pendekatan pengujian pertama disebut pengujian ***black box*** dan yang kedua disebut ***white box***.

Jika perangkat lunak komputer dipertimbangkan, maka pengujian black box berkaitan dengan pengujian yang dilakukan pada interface perangkat lunak. Meskipun didesain untuk mengungkap kesalahan, pengujian black box digunakan untuk memperlihatkan bahwa fungsi-fungsi perangkat lunak adalah operasional, bahwa input diterima dengan baik dan output dihasilkan dengan tepat dan integrasi informasi eksternal (seperti file data) dipelihara. Pengujian balck box menguji beberapa aspek dasar suatu sistem dengan sedikit memperhatikan struktur logika internal perangkat lunak tersebut.

Pengujian white box perangkat lunak didasarkan pada pengamatan yang teliti terhadap detail prosedural. Jalur-jalur logika yang melewati perangkat lunak diuji dengan memberikan test case yang menguji serangkaian kondisi dan atau loop tertentu. "Status program tersebut"dapat diuji pada pada berbgai titik untuk menentukan apakah status yang diharapkan atau dituntut sesuai dengan status aktual.

1.5. Pengujian White-Box

Pengujian white box, kadang-kadang disebut pengujian *glass box*, adalah metode desaintest case yang menggunakan struktur kontrol desain prosedural untuk memperoleh test case. Dengan menggunakan metode pengujian white box, perekayasa sistem dapat melakukan test case yang :

1. memberikan jaminan bahwa semua jalur independen pada suatu modal telah digunakan, paling tidak satu kali.
2. Menggunakan semua keputusan logis pada sisi *true* dan *false*
3. Mengeksekusi semua loop pada batasan mereka dan baas operasional mereka
4. Menggunakan struktur data internal untuk menjamin validitasnya

Pada titik ini, dapat diajukan pertanyaan yang beralasan, yaitu "Mengapa menghabiskan waktu dan energi untuk menguji logika jika kita dapat dengan lebih baik memperluas kerja yang dapat memastikan bahwa persyaratan program telah dipenuhi ?" Bila dinyatakan dengan cara lain, mengapa kita tidak menggunakan semua energi kita untuk melakukan pengujian black box ? Jawabannya ada pada sifat cacat perangkat lunak :

- *Kesalahan logis dan asumsi yang tidak benar berbanding terbalik dengan probabilitas jalur program yang akan dieksekusi.* Kesalahan cenderung muncul dalam kerja kita pada saat kita mendesain dan mengimplementasi fungsi, kondisi atau kontrol yang berada di luar mainstraim. Pemrosesan setiap hari cenderung dipahami dengan baik sementara pemrosesan "kasus khusus" cenderung berantakan

- *Kita sering percaya bahwa jalur logis mungkin tidak akan diseksekusi bila pada kenyataannya akan diseksekusi pada basis reguler*. Aliran logika dari suatu program kadang- kadang bersifat konterintuitif yang berarti asumsi kita yang tidak kita sasari mengenai aliran dan data kontrol dapat menyebabkan kita membuat kesalahan desai yang akan terungkap hanya setelah pengujian jalur mulai.
- *Kesalahan tipografis adalah random.* Bila sebuah program diterjemahkan ke dalam kode sumber bahasa pemrograman maka dimungkinkan akan terjadi banyak kesalahan pengetikan. Beberapa akan ditemukan dengan mekanisme pengecekan sintaks, tetapi yang lainnya akan tetap tidak terdeteksi sampai pengujian mulai.

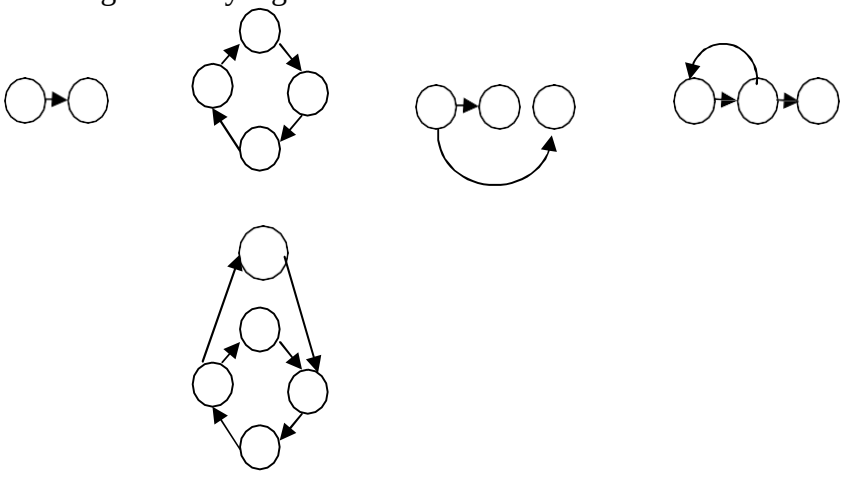
Masing-masing alasan tersebut memberikan suatu argurmen untuk melakukan pengujian white box. Pengujian black box, tidak peduli seberapa cermat dilakukan, dapat menangkap bentuk kesalahan tersebut.

1.6. Pengujian Basis-Path

Pengujian basis path adalah teknik pengujian white box yang diusulkan pertama kali oleh Tom McCabe. Metode basis ini memungkinkan desainer test case mengukur kompleksitas logis dari desai prosedural dan menggunakannya sebagai pedoman untuk menetapkan basis set dari jalur eksekusi. Test case yang dilakukan untuk menggunakan basis set tersebut dijamin menggunakan setiap statment di dalam program paling tidak sekali selama pengujian.

1.6.1. Notasi Diagram Alir

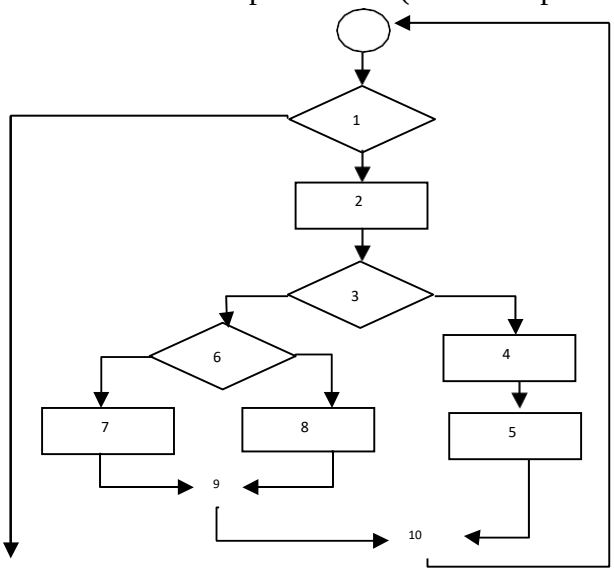
Sebelum metode babsis path diperkenalkan, terlebih dahulu akan dijelaskanmengenasi notasi sederhana dalam bentuk diagram alir (grafik alir). Diagral alir menggambarkan aliran kontrol logika yang menggunakan notasi seperti ditunjukkan pada gambar 1. Masing-masing gagasan terstruktur memiliki simbol diagram alir yang sesuai.



Sequence If While Until Case

Gambar 1. Notasi diagram aliran

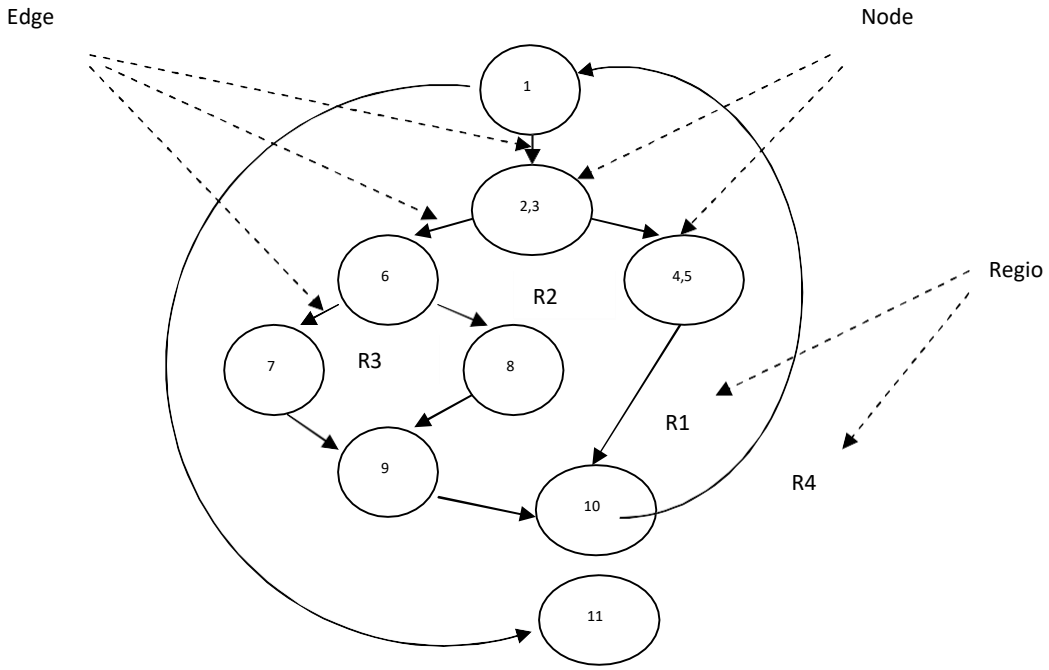
Gambar 2 merepresentasikan desain prsedural grafik alir. Paga gambar tersebut tampak struktrul kontrol progrm. Gambar 3 memetakan grafik alir tersebut ke dalam grafik alir yang sesuai. Pada gambar tersebut masing-masing lingkaran disebut simpulu grafik alir, merepresentasikan satu atau lebih statemen prosedural. Urutan kotak proses dan belahketupat keputusan dapat memetakan simpul tunggal. Anak panah pada grafik alir tersebut disebut edges atau links, merepresentasikan aliran kontrol dan analog dengan anak panah bagan alir. Edge harus berhenti pada suatu simpul, mesikipun nila simpul tersebut tidak merepresentasikan statemen prosedural (misal simpul untuk bangun if-then-



else). Area yang dibatasi oleh edge dan simpul disebtu region. Pada saat menghitung region kita perlu memasukkan area di luar grafik dan menghitungnya sebgai sebuah region.

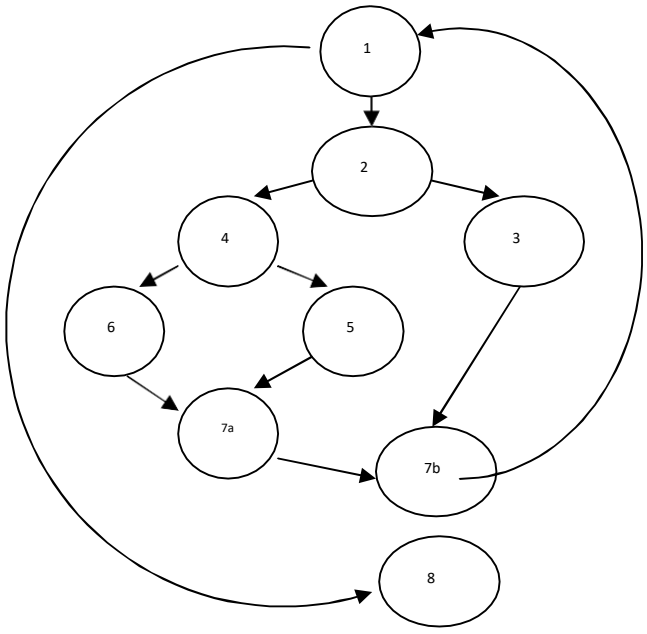
11

Gambar 2. Bagan Alir



Gambar 3. Grafik Alir

Setiap representasi desain prosedural dapat diterjemahkan ke dalam suatu grafik alir. Gambar 4 memperlihatkan segmen PDL dan grafik airnya yang sesuai. Tampak bahawa statemen PDL telah diberi nomor dan penomoran yang sesuai digunakan untuk grafik alir tersebut.



PDL

Procedure : sort

1 : do while records remain Read record;

2: if record field 1 = 0

3 : then process record; Store in buffer; Increment counter;

4 : elseif record field 2 = 0

5 : then reset counter;

6: else process record; Store in file;

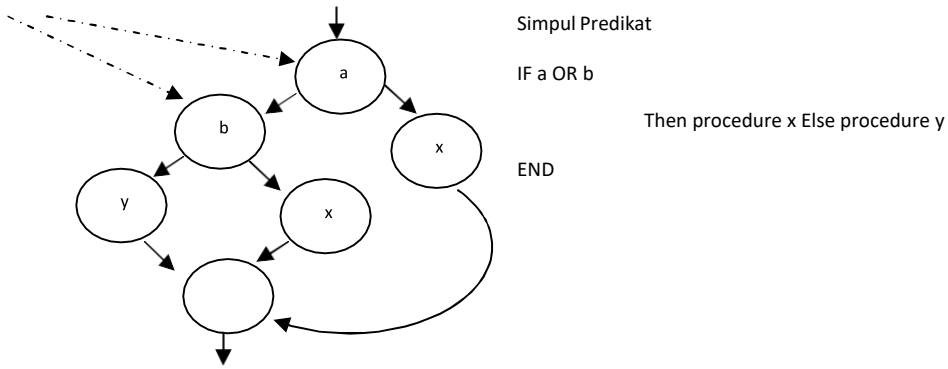
7a: endi f Endif

7b: enddo

8 : end

Gambar 4. Menterjemahkan PDL ke grafik alir

Bila ada kondisi gabungan dalam desain prosedural, maka pembuatan grafik alir menjadi sangat rumit. Kondisi gabungan terjadi bila satu atau lebih operator Boolean (logika OR, AND, NAND, NOR) ada pada statemen kondisional. Gambar 5 menunjukkan sebauh saegmen PDL menterjemahkan ke dalam grafik alir. Tampak bahwa simpul (node) yang terpisah diciptakan untuk
masing-masing kondisi a dan b pada statemen IF a OR b. Masing-masing simpul yang berisi sebuah kondisi disebut simpul predikat dan ditandai dengan dua atau lebih edge yang berasal darinya.



Gambar 5. Logika gabungan

1.6.1. Kompleksitas Siklomatis

Kompleksitas siklomatis adalah metriks perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila metriks ini digunakan dalam konteks metode pengujian basis path, maka nilai yang terhitung untuk kompleksitas siklomatis menentukan jumlah jalur independen dalam basis set suatu program an memberi batas atas bagi jumlah pengujian yang harus dilakukan untuk memastikan bahwa semua statemen telah dieksekusi sedikitnya satu kali.

Jalur independen adalah jalur yangmemlalui progarm yang mengintroduksi sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Bila dinyatakan dengan terminologi grafik alir, jalur independen harus bergerak sepanjang paling tidak satu edge yang tidak dilewatkan sebelum jalur tersebut ditentukan. Contoh, serangkaian jalur independen untuk grafik alir yang ditunjukkan pada gambar 3 adalah :

Jalur 1 : 1-11
Jalur 2 : 1-2-3-4-5-10-1-11
Jalur 3 : 1-2-3-6-8-9-10-1-11
Jalur 4 : 1-2-3-6-7-9-10-1-11

Tampak bahwa masing-masing jalur baru memeperkenalkan sebuah edge baru.

Jalur 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 tidak dianggap jalur independen karena merupakan gabungan dari jalur-jalur yang sudah ditentukan dan tidak melewati beberapa edge baru.

Jalur 1,2,3 dan 4 yang ditentukan di atas terdiri dari sebuah basis set untuk grafik alir pada

gambar 3. Bila pengujian dapat dilakukan untuk memaksa adanya eksekusi dari jalur-jalur tersebut, maka setiap statemen pada program tersebut akan diseksekusi paling tidak satu kali dan setiap kondisi sudah akan dieksekusi pada sisi true dan fale-nya. Perlu dicatat bahwa basis set tidaklah unik. Pada dasarnya semua jumlah basis set yang berbeda dapat diperoleh untuk suatu desain prosedural yang diberikan.

Bagaimana kita tahu banyaknya jalur yang dicari ? Komputasi kompleksitas siklomatis memebrikan jawaban. Fondasi kompleksitas siklomatis adalah teori grafik dan memberi kita metriks perangkat lunak yang sangat berguna. Kompleksitas dihitunga dalam satu adari tiga cara berikut :

1. Jumlah region grafik alir sesuai dengan kompleksitas siklomatis
2. Kompleksitas siklomatis *V(G)* untuk grafik alir G ditentukan sebagai *V(G)=E-N+2* dimana E adalah jumlah edge grafik alir dan N adalah jumlah simpul grafik alir
3. Kompleksitas siklomatis *V(G)* untuk grafik alir G ditentukan sebagai *V(G)=P+1,* dimana P adalah jumlah simpul predikat yang diisikan dalam grafik alir G.

Pada gambar 3, kompleksitas siklomatis dapat dihitung dengan menggunakan masing-masing algoritma di atas :

1. Grafik alir mempunyai 4 region
2. *V(G)* = 11 edge – 9 simpul + 2 = 4
3. *V(G) =* 3 simpul yang diperkirakan + 1 = 4

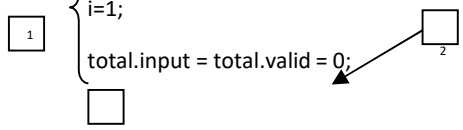
Dengan demikian, kompleksitas siklomatis dari grafik alir pada gambar 3 adalah 4.

Yang lebih penting, nilai untuk *V(G)* memeberi kita batas atas untuk jumlah jalur independen yang membentuk basis set, dan implikasinya, batas atas jumlah pengujian yang harus didesain dan dieksekusi untuk menjamin semua statemen program.

1.6.2. Melakukan Test Case

Metode pengujian basis path dapat diaplikasikan pada desain prosedural atau kode sumber. Pengujian basis path memiliki sederetan langkah. Average prosedur, yang digambarkan dalam PDL pada gambar 6 menggambarkan masing-masing langkah pada metode desain test case. Tampak bahwa average, meskipun merupakan suatu algoritma yang sederhana berisi kondisi gabungan dan loop.

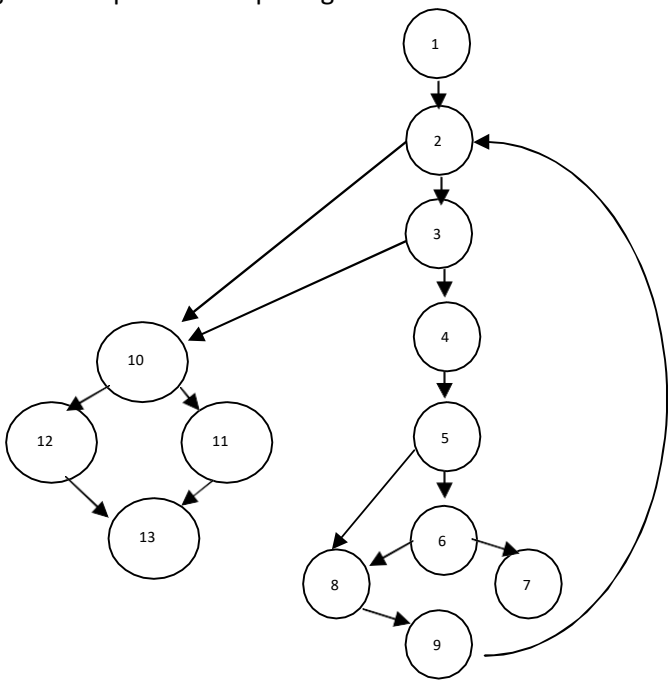
```
PROCEDURE average

  *This procedure compute the average of 100 or fewer numbers that lie between bounding values; it also
    compute the sum and teh total number valid


INTERFACE RETURNS average, total.input, total valid;
INTERFACE ACCEPTS value, minimum, maximum;


TYPE value[1:11] IS SCALAR ARRAY;

TYPE average, total.input, total valid;
      minimum, maximum, sum IS SCALAR

TYPE i IS INTEGER


i=1;

total.input = total.valid = 0;
```



```
sum = 0;
DO WHILE value[i] < > -999 and total.input < 100
    Increment total.imput by 1
    IF value[i] >= minimum AND value[i] <= maximum
        THEN increment total.valid by 1;
                Sum = sum + value[i]
        ELSE skip
    ENDIF
    increment i by 1;
DDO
    IF total.valid > 0
        THEN average = sum /total.valid;
        ELSE average = -999;
    ENDIF
END average
```

Gambar 6. PDL untuk desain test case dengan simpul-simpul yang diidentifikasikan

1. Dengan menggunakan desain atau kode sebagai dasar, gambarkan sebuah grafik alir yang sesuai. Grafik alir diciptakan dengan menggunakan simbol dan aturan konstruksi yang disajikan pada subbab 1.6. PDL untuk average pada gambar 6 diciptakan dengan menomori statemen-statemen PDL yang akan dipetakan ke dalam simpul grafik alir yang sesuai. Grafik alir yang sesuai diperlihatkan pada gambar 7.



Gambar 7. Grafik aliran dari average prosedur

2. Tentukan kompleksitas siklomatis dari grafik alir resultan. Kompleksitas siklomatis , *V(G)*

ditentukan dengan mengaplikasikan satu dari algoritma-algoritma yang telah digambarkan di atas. Perlu dicatat bahwa V9G) dapat ditentukan tantpa mengembangkan grafik alir dengan menghitung semua statemen kondisional dalam PDL (untuk average prosedur, kondisi gabungan menghitung 2) dan penambahan 1. Pada gambar 7 :
V(G) = 6 region

V(G) = 18 edge – 14 simpul + 2 = 6

V(G) = 5 simpul prekdikat + 1 = 6

3. Tentukan sebuah basis set dari jalur independen secara linier. Harga V(G) memeberikan jumlah jalur independen secara linier melalui struktur kontrol program . Dalam kasus average prosedur, kita dapat menentukan enam jalur :
   Jalur 1 : 1-2-10-11-13

   Jalur 2 : 1-2-10-12-13

   Jalur 3 : 1-2-3-10-11-13

   Jalur 4 : 1-2-3-4-5-8-9-2- ....

   Jalur 5 : 1-2-3-4-5-6-8-9-2-....

   Jalur 6 : 1-2-3-4-5-6-7-8-9-2-.....

4. Siapkan test case yang akan memaksa adanya eksekusi setiap basis set. Data harus dipilih sehingga kondisi pada simpul-simpul predikat terpasang secara tepat pada saat masing-masing juluar diuji. Test case yang memenuhi basis set yang digambarkan di atas adalah :
   - Test case jalur 1 :
     Harga (k)= input valid, dimana k < i yang
     dteteapkan di bawah Harga (i) = -999 dimana 2 $\leq$
     2 i $\leq$ 100

     Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai k dan total yang tepat.

     Catatan : jalur 1 tidak dapat diuji sendirian karena harus diuji sebagai bagian dari pengujian jalur 4,5 dan 6

   - Test case
     jalur 2 : Harga
     (i) =-999
     Hasil yang diharapakan : rata-rata -999, total yang lain pada nilai awal

   - Test case jalur 3 :
     Usahakan untuk memproses 101 nilai
     atau lebih 100 nilai pertama harus valid

     Hasil yang diharapkan : sama seperti test case 1

   - Test case jalur 4 :

Nilai (i) = input valid dimana
i < 100 Nilai (k) < minimum,
dimana k < i

Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai-nilai n dan total yang tepat

- Test case jalur 5 :
  Nilai (i) = input valid dimana
  i< 100 Nilai (k) > maksimum,
  dimana $k \leq i$

  Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai-nilai n dan total yang tepat

- Test case jalur 6 :
  Nilai (i) = input valid dimana i < 100

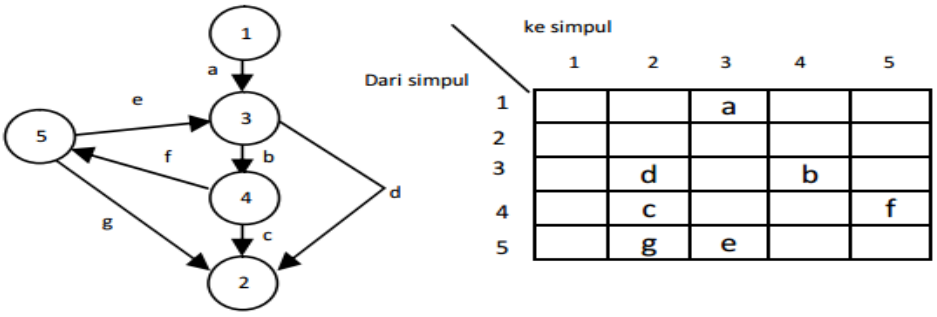
  Hasil yang diharapkan : rata-rata yang benar berdasarkan nilai-nilai ni dan total yang tepat.

Masing-masing test case dieksekusi dan dibandingkan untuk mendapat hasil yang diharapkan. Sekali semua test telah dilengkapi maka penguji dapat yakin bahwa semua statemen pada program telah dieksekusi paling tidak satu kali. Penting untuk dicatat bahaw beberapa jalur independ (misal jalur 1 pada contoh di atas) tidak dapat diuji secara terpisah (sendiri) karena kombinasi data yang diperlukan untuk melintasi jalur tersebut tidak dapat dicapai di dalam aliran normal dari program. Dalam kasus semacam ini, jalur-jalur diuji sebgai bagian dari pengujian jalur yang lain.

1.6.3. Matriks Grafik

Prosedur untuk mendaptakan grafik alir dan menentukan serangkaian basis path , cocok dengan mekanisasi. Untuk mengembangkan peranti perangkat lunak yangmembantu pengujian basis patha, struktur data yang disebut matriks grafis dapat sangat berguna.

Matriks garfis adalah matriks bujur sangkar yang ukurannya sama dengan jumlah simpul pada grafik alir. Masing- masing baris dan kolom sesuai dengan simpul yang diidentifikasikan dan entri matriks sesuai dengan edge di atantara simpul. Contoh sederhana grafik air dan matriks grafisnya yang sesuai diperlihatkan pada gambar 8.

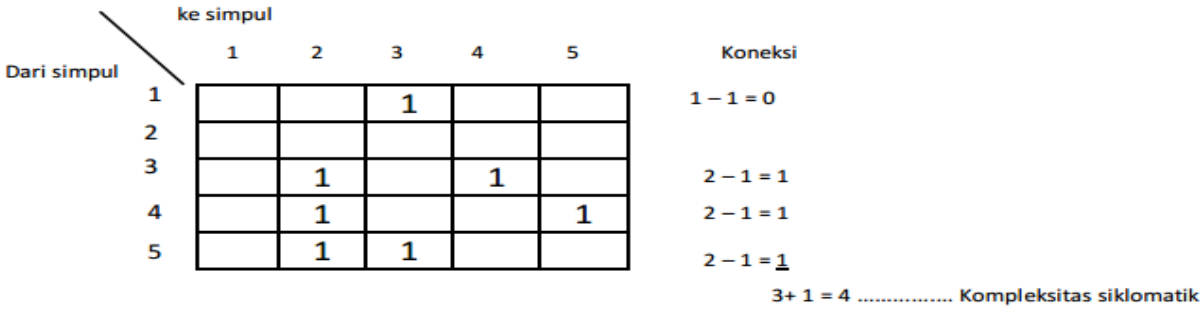


Gambar 8. Matriks koneksi

Pada gambar tersebut, masing-masing simpul pada grafik alir diidentifikasikan oleh bilangan, sementara masing- masing edge diidentifikas dengan huruf. Entri huruf dibuat di dalam atriks tersebut untuk pencocokan dengan koneksi di antara dua simpul. Sebgai contoh, simpul 3 disambungkan dengan simpul 4 oleh edge b. Untuk titik ini, matriks grafis tidak lebih dari sekedar representasi tabuler dari grafik alir. Tetapi denganmenambahkan sebuah link weighy pada masing- masing entri matriks, maka matriks grafis dapat menjadi alat yang sangat kuat untuk mengevaluasi struktur kontrol program selama pengujian. Link weight memberikan informasi tambahan mengenai aliran kontrol. Dalam bentuknya yang paling sederhana, link weight adalah 1 (ada hubungan) atau 0 (tidak ada hubungan). Tetapi link weight dapta ditetapkan lain, yaitu properti yang lebih menarik :

- Probabilitas di mana sebuah link (edge) akan dieksekusi
- Waktu perosesan yang digunakan slama pelewatan sebuah link
- Memori yang diperlukan selama peleawatan link
- Sumber daya yang diperlukan selama pelewatan link

Untuk menggambarkanya, kita gunakan pembebanan sederhana untuk menunjukkan hubungan (0 atau 1). Matriks grafis pada gambar 8 digambar lagi seperti ditunjukkan pada gambar 9. Masing-masng hruuf telah diganti dengan angka 1 yang menunjukkan bahwa ada hubungan (nol telah dihilangkan supaya jelas). Dengan bentuk seperti itu, matrriks grafis disebut matriks koneksi. Pada gambar 9 masing-masing baris dengan dua entri atau lebih merepresentasikan sebauah simpul predikat. Dengan demikian dengan mengejrjakan aritmatika yang diperlihatkan di sebelah kanan sambungan matriks akan memberi kita metode lain untuk menentukan kompleksitas siklomatis (subbab 1.6.2).



Gambar 10. Beragam jenis loop

Beizer memberikan perlakukan yang teliti dari algoritma matematika tambahan yang dapat

diaplikasikan pada matriks-matriks grafis. Dengan menggunakan teknik tersebut, maka analisis yang dipergunakan untuk mendesain test case dapat diotomasi sebagian atau sepenuhnya.

1.7. Pengujian Struktur Kontrol

Teknik pengujian basis path yang digambarkan pada subbab di atas adalah salah satu dari sejumlah teknik untuk pengujian struktur kontrol. Meskipun pengujian basis path sederhana dan efektif, tetapi pengujian itu tidak memadai. Dalam bagian ini akan dibahas variasi lain pada pengujian struktur kontrol. Hal ini memperluas kupasan pengujian dan meningkatkan kualitas pengujian white-box.

1.7.1. Pengujian Kondisi

Pengujian kondisi adalah sebuah metode desain test case yang menggunakan kondisi logis yang ada pada suatu program. Kondisi sederhana adalah variabel Boolean atau suatu persamaan hubungan, dapat didahului dengan satu operator NOT (“¬”). Persamaan relasional mengambil bentuk :

$$E_1 \text{ (operator-relasional) } E_2$$

Dimana E_1 *dan* E_2 merupakan persamaan aritmatika dan (operator relasional) adalah salah satu dari operator berikut : “<”,”≤”,”=”,”≠”,””¬=”(pertidaksamaan),”>” atau “≥”. Kondisi gabungan terdiri dari dua atau lebih kondisi sederhana, oprator Boolean dan tanda kurung. Diasumsikan bahwa operator Boolean yang diijinkan dalam suatu kondisi gabungan meliputi OR (“|”), AND(“&”) dan NOT (“¬”). Kondisi tanpa persamaan realsional disebut persamaan Boolean.

Dengan demikian, tipe-tipe komponen yang mungkin di dalam suatu kondisi meliputi operator Boolean, sebuah variabel Boolean, sepasang tanda kurung Boolean (yang mengelilingi) suatu kondisi gabungan atau sederhana), sebuah operator relasional atau sebuah persamaan aritmatika. Bila suatu kondisi tidak benar, maka palling tidak satu komponen dari kondisi itu salah. Dengan demikian, tipe kesaahan pada suatu kondisi meliputi berikut ini :

1. Kesalahan operator Boolean (adanya operator Boolenan yang salah/hilang/ekstra)
2. Kesalahan variabel Boolean
3. Kesalahan tanda kurung Boolean
4. Kesalahan operator relasional
5. Kesalahan persamaan aritmatika

Metode pengujian kondisi berfokus pada pengujian masing-masing kondisi di dalam program. Strategi pengujian kondisi biasanya memiliki dua keuntungan. Pertama, pengukuran kupasan pengujian dari suatu kondisi adalah sederhana. Kedua, cakupan pengujian terhadap kondisi di dalam suatu program memberikan pedoman untuk melakukan pengujian tambahan untuk program tersebut.

Tujuan pengujian kondisi adalah mendeteksi tidak hanya kesalahan di dalam kondisi program, tetapi juga kesalahan lain pada program. Jika pengujian ditentukan untuk program P efektif untuk mendeteksi kesalahan pada kondisi yang ada pada P, maka kemungkinan besar pengujian juga efektif untuk mendeteksi kesalahan lain pada P. Sebgai tambahan, jika strategi pengujian efektif untuk mendeteksi kesalahan-kesalahan pada suatu kondisi, maka kemungkinan bear strategi tersebut juga efektif untuk mendeteksi kesalahan pada suatu program.

Sejumlah strategi pengujian kondisi telah diusulkan. Pengujian cabang mungkin merupakan strategi pengujian kondisi yang paling sederhana. Untuk sudatu kondisi gabunga C, cabang-cabang true dan false dari C dan setiap kondisi sederhana pada C perlu dieksekusi paling tidak satu kali.

Pengujian domain membutuhkan tiga atau empat pengujian untuk dilaukan pada sebuah persamaan realsional. Karena perasamaan relasional mengambil bentuk :

$$E_1 \text{ (operator-relasional) } E_2$$

Maka diperlukan tiga pengujian untuk membuat nilai E_1 lebih tinggi, sama dengan atau kurang dari nilai E_2 secara berurutan. Bila operator relasional salah dan E_1 dan E_2 benar, maka ketiga pengujian itu menjamin pendeteksian kesalahan operator relasional. Untuk mendeteksi kesalahan pada E_1 dan E_2 maka tes yang membuat harga E_1 lebih besar atau kurang dari harga E_2 harus membuat perbedaan antara dua harga itu menjadi sekecil mungkin.

Untuk persamaan Booleand dengan n variabel, semua 2^n pengujian yang mungkin ($n > 0$)
perlu dilakukan. Strategi itu dapat mendeteksi operator, variabel dan kesalahan tanda kurung Boolean, tatapi hanya praktis jika n kecil.

Pengujian error sensitive untuk persamaan Boolean dapat juga dilakukan. Untuk persamaan Boolean tunggal (persamaan Boolean dimana masing-masing variabel Boolean terjadi hanya sekali) dengan n variable Boolean ($n > 0$), kita dapat dengan mudah memunculkan serangkaian pengujian dengan kurang dari 2^n pengujian sehingga rangkaian pengujian itu menjamin pendeteksian kesalahan opertaor Boolean bertingkat serta efektif untuk mendeteksi kesalahan-kesalahan yang lain.

Tai mengusulkan strategi pengujiankondisi yang didasarkan atas teknik yang sudah diuraikan. Disebut pengujian BRO (branch and relational operator), tekni tersebut menjamin pendeteksian kesalahan cabgan dan operator relasional, dengan kondisi bahwa semua variabel Boolean dan opertaor relasional pada kondisi itu terjadi hanya sekali dan tidak memiliki variabel umum.

Strategi BRO menggunakan batasan kondisi bagi suatu kondisi C. Batasan kondisi untuk C dengan n kondisi sederhana ditentukan sebagai (D_1, D_2, ...D_n) dimana D_i ($0 < i \leq n$) merupakan simbol yangmenentukan batasan pada hasil akhir dari kondisi sederhana ke-i dalam kondisi C. Batasan kondisi D untuk kondisi C dikatakan dipenuhi oleh eksekusi dari C bila selama eksekusi dari C, hasil akhir dari masing-masing kondisi sederhana di dalam C memeneuhi batasan yang bersesuaian di dalam D.

Untuk variable Boolean B, kita menentukan baasan pada hasil akhir B yang menyatakan

bahwa B harus true (t) atau false (f). Dengan carayang sama, untuk persamaan realsional, simbol- simbol >, = dan < digunakan untuk menentukan batasan pada hasil akhir persamaan.

Contoh-1, diketahui kondisi berikut :

$$C_1 : B_1 \& B_2$$

Dimana B_1 dan B_2 adalah variabel Boolean. Batasan kondisi untuk C_i adalah bentuk (D_1,D_2) dimana masing-masing dari D_1 dan D_2 adalah "t" atau "f". Nilai t, f adalah batasan kondisi untuk C1 dan dicakup oleh pengujian yang membuat harga B_1 menjadi true (t) dan harga B_2 menjadi false (f). Strategi pengujian BRO mengharuskan himpunan batasan [(t,t),(f,t),(t,f)] dicakup oleh eksekusi C_1. Bila C_1 tidak benar karena satu atau lebih kesalahan operator Boolean, maka sedikitinya satu anggota dari himpunan batasan akan memaksa C_1 untuk gagal.

Contoh-2 diketahui kondisi berikut :

$$C_2 : B_1 \& (E_3 = E_4)$$

Dimana B_1 adalah persamaan Boolean dan E_3 dan E_4 adalah persamaan aritmatika. Batasan kondisi untuk C_2 adalah bentuk (D_1,D_2) dimana D_1 adalah "t" atau "f" dan D_2 adalah >, = atau <. Karena C_2 sama seperti C_1 kecuali bahwa kondisi sederhana kedua di dalam C_2 adalah persamaan elasional, kita dapat membangun himpunan batasan untuk C_2 denganmemodifikasi himpunan pembatas [(t,t),(f,t),(t,f)] yang ditentukan untuk C_1. Perhatikan bahwa "t" untuk $(E_3 = E_4)$ mengimplikasikan "=" dan bahwa "f" untuk $(E_3=E_4)$ mengimplikasikan "<" atau ">". Dengan menggantikan (t,t) dengan (t,=) maka hasil himpunan batasan untuk C_2 adalah {t,=),(t,<), (t,>). Cakupan himpunan batasan tersebut akan menjamin pendeteksian Boolean dan kesalahan-kesalahan operator realsional pada C_2.

Contoh 3, diketahui kondisi sebagai berikut :

$$C_3 : (E_1 > E_2) \& (E_3 = E_4)$$

Dimana E_1,E_2, E_3 dan E_4 adalah persamaan aritmatika. Pembatas kondisi untuk C_3 adalah bentuk (D_1, D_2), dimana masing-masing dari D_1 dan D_2 adalah >, = atau <. Karena C_3 sama seperti C_2, kecuali bahwa kondisi sederhana yang pertama pada C_3 adalah persamaan relasional, maka kita dapat mmembangun himpunan batasan untuk C_3 denganmemodifikasi himpunan batasan untuk C_2 dan akan diperoleh :

$$\{(>,=), (=,=),(<,=),(>,>),(>,<)\}$$

Cakupan dari himpunan batasan tersebut akan menjamin pendeteksian kesalahan operator relasional pada C_3.

1.7.2. Pengujian Aliran Data

Metode pengujian aliran data memilih jalur pengujian dari suatu program sesuai dengan lokasi definisi dan menggunakan variabel-variabel pada program. Sejumlah pengujian aliran data telah dipelajari dan dibandingkan.

Untuk menggambarkan pendekatan pengujian aliran data, diasumsikan bahwa masing-masaing statement pada suatu program diberi nomor statemen yang unik dan setiap fungsi tidak memodfikasi parameter atau variabel globalnya. Untuk statemen dengan S sebgai nomor statementnya :

DEF(S) ={X|statemen S berisi sebuah definisi dari X} USE(S) = X|statemen S berisi suatu penggunaan dari X}

Bila statemen S adalah statemen if atau loop, maka himpunan DEF-nya kosong dan himpunan USE- nya didasarkan pada kondisi statemen S. Definisi variabel X pada statemen S dikatakan hidup pada staemen S' jika ada suatu jalur dari statemen S ke statemen S' yang tidak berisi defiisi yang lain dari X.

Rantai definition-use (atau rantai DU) dari variabel X berbentuk [X,S,S'] dimana S dan S' adalah nomor statemen X pada DEF(S') dan USE(S') dan definisi X pada statemen S hidup pada statemen S'.

Strategi pengujian aliran data sederhana mengharuskan setiap rantai DU dicakup paling tidak satu kali. Strategi ini disebut strategi pengujian DU. Pengujian DU tidak menjamin percakupan semua cabang sebuah program. Akan tetapi, satu cabang tidak dijamin dicakup oleh pengujian DU hanya pada situasi jarang seperti konstruksi if-then-else dimana bagian then tidak memilii definisi mengenai sembarang variabel dan bagian else tidak ada. Dalam situasi seperti ini, cabgan else dari statemen if tersebut tidak perlu dicakup oleh pengujian DU.

Strategi pengujian aliran data berguna untuk memilih jalur pengujian dari suatu program yang berisi statemen if dan loop yang tersarang. Untuk menggambarkannya, perhatikan aplikasi pengujian DU untuk memilih jalur pengujian pada PDL berikut :

```
Proc x

    B1;

    Do while C1
    If C2

      Then

            If C4

              Then B4;
               Else B5;

           Endif;
       Else

            If C3

                Then B2;
                Else B3;

           Endif;
      Endif

    Enddo;
    B6;

Endproc;
```

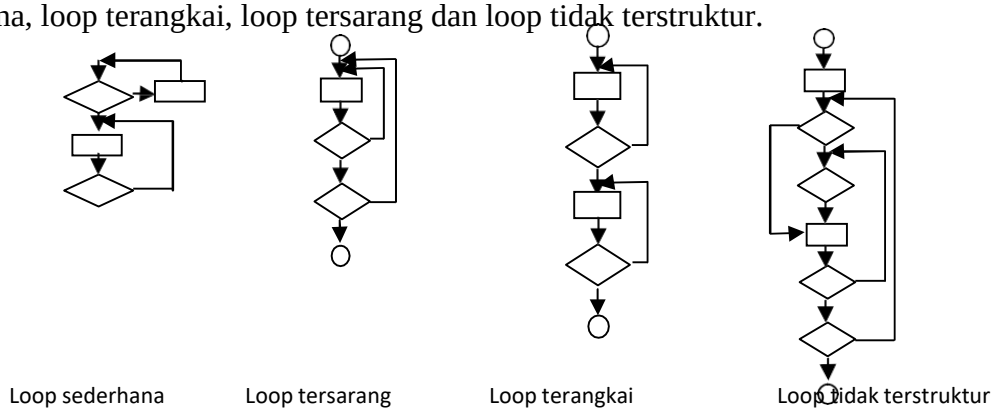
Untuk membuat strategi pengujian DU memilih jalur pengujian dari diagaram aliran kontrol, kita perlu tahu terdefinisi dan penggunaan masing-masing kondisi atau blok pada PDL. Anggap bahwa variabel X didefinisikan pada statemen terakhir dari blok B1, B2, B3, B4 dan B5 dan digunakan dalam statemen pertama dari blok B2, B3, B4, B5 dan B6. Startegi pengujian DU memerlukan ekseusi jalur terpendek masing-masing dari B_i, $0 < i \leq 5$ ke masing-masing dari B_j, $1 < j \leq 6$. Pengujian semacam itu mencakup banyak penggunaan variabel X dalam kondisi C1, C2, C3 dan C4). Meskipun ada 25 rantai DU dari variabel X, kita hanya membutuhkan lima jalur untuk mencakup rantai-ranatai DU tersebut. Alasannya adalah diperlukan lima jalur untuk mencakup rantai DU X dari B_i, $0 < i \leq 5$ ke B6 dan rantai Du yang lain dapat dicakup degan membuat lima jalur tersebut berisi iterasi dari loop.

Bila kita mengaplikasikan strategi pengujian cabang untuk memilih jalur pengujian dari PDL tersebut di atas, kita tidak memerlukan informasi tambahan. Untuk memilih jalur dari diagram pengujian BRO, kita perlu tahu struktur masing-masing kondisi atau blok (setelah pemilihan suatu jalur dari suatu program, kita harus menentukan apakah jalur tersebut mungkin dikerjakan dengan mudah untuk program itu, misal apakah ada paling tidak satu inut yang menggunakan jalur tersebut).

Karena statemen pada suatu program berhubungan satu dengan yang lainnya sesuai dengan definisi dan penggunaan variabel, maka pendekatan aliran data efektif untuk perlindungan dan kesalahan. Tetapi, masalah cakpan pengujian pengukuran dan pemilihan jalur pengujian untuk pengujian aliran data lebih sulit daripada masalah yang berhubugan dengan pengujian kondisi.

1.7.3. Pengujian Loop

Pengujian loop merupakan teknik pengujian white-box yang secara ekslusif berfokus pada validitas konstruksi loop. Empat kelas loop yang berbeda dapat didefinisikan : loop seerhana, loop terangkai, loop tersarang dan loop tidak terstruktur.



Gambar 10. Beragam jenis loop

- **Loop sederhana**. Himpunan berikut harus diaplikasikan pada loop sederhana, diman n adalah jumlah maksimum yang diijinkan melewati loop tersebut.
    1. Abaikan keselurhan loop
    2. Hanya satu yang melewati loop
    3. Dua yang melewati loop
    4. m melewati loop diman m < n
    5. n – 1, n, n+1 melewati loop
- **Loop tersarang**. Bila kita ingin memperluas pendekatan pengujian bagi loop sederhana ke loop tersarang, jumlah pengujian mungkin akan berkembang secara geometris sesuai tingkat pertambahan persarangan sehingga sejumlah pengujian menjadi tidak praktis. Beizer mengusulkan suatu pendekatan yang membantu mengurangi jumlah pengujian :
    1. Mulai pada loop yang paling dalam. Atur sema loop ke nilai minimum.
    2. Lakukan pengujian loop sederhana untuk loop yang paling dalam sementara menjaga loop yang paling luar pada nilai parameter iterasi minimumnya (misal pencacah loop). Tambahkan pengujian yang lain untuk nilai out of range aau nilaiyang tidak diperbolehkan
    3. Bekerja menuju ke luar, dengan melakukan pengujian untuk loop selanjutnya, teteapi menjaga semua loop bagian luar yang lain pada nilai minimumnya dan loop tersarang lainnya pada harga “tertentu”.
    4. Lanjutkan sampai semua loop telah tersarang.

- **Loop terangkai**. Loop terangkai dapat diuji denganmenggunakan pendekatan yang ditentukan untuk loop sederhana bila masing-masing dari loop itu independen terhadap yang lain. Tetapi bila dua loop dirangkai dan pencacah loop untuk loop 1 digunakan sebagai harga awal untuk loop 2, kemudian loop tersebut menjadi tidak independen, maka pendekatan diaplikasikan ke loop tersarang direkomendasi.
- **Loop tidak terstruktur**. Kapan saja memungkinkan kelas loop ini harus didesain lagi untuk mencerminkan penggunaan konsepsi pemrograman terstruktur.

1.8. Pengujian Black-Box

Pengujian black-box berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengjian black-box memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua peryaratan fungsional untuk suatu program. Pengujian black-box bukan merupakan alternatif dari teknik white-box tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode white-box.

Pengujian black-box berusaha menemukan kesalahan dalam kategori sebagai berikut :

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan interface
3. Kesalahan dalam struktur data atau akses database eksternal
4. Kesalhan kinerja
5. Inisialisasi dan kesalahan terminasi

Tidak seperti pengujian white box yang dilakukan pada awal proses pengujian, pengujian black box cenderung diaplikasikan selama tahap akhir pengujian. Karena pengujianblack box memperhatikan struktur kontrol, maka perhatian berfokus pada domain informasi. Pengujian didesain untuk menjawab pertanyaan-pertanyaan berikut :

- Bagiamana vaiditas fungsional diuji ?
- Kelas input apa yang akan membuat test cse menjadi baik ?
- Apakah sistem sangat sensitif terhadap harga input tertentu ?
- Bagaimana batasan dari suatu data diisolasi ?
- Kecepatan data apa dan volume data apa yang dapat ditolerir oleh sistem ?
- Apa pengaruh kombinasi tertentu dari data terdahap operasi sistem ?

Dengan mengaplikasikan teknik black box, maka kita menarik serangkaian test case yang memenuhi kriteria berikut ini :

1. Test case yang mengurangi, dengan harga lebih dari saatu, jumlah test case tambahan yang harus didesain untuk mencapai pengujian yang dapat dipertanggungjawabkan.
2. Test case yang memberi tahu kita sesuatu mengenaikehadiran atau ketidakhadiran kelas kesalahan daripada memberi tahu kesalahan yang berhubugnan hanya dengan pengujian spesifik yang ada.

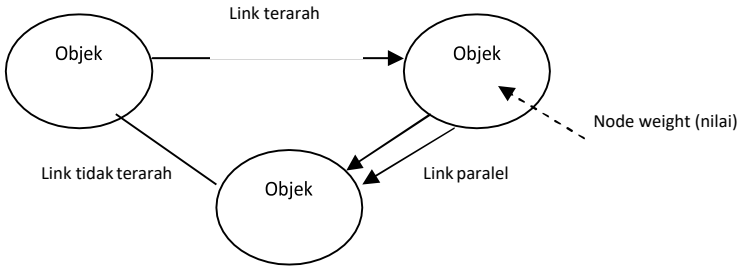
1.8.1. Metode Pengujian Graph-based

Langkah pertama pada pengujian black box adalah “memahami objek” yang dimodelkan di dalam perangkat lunak dan hubungan yang akan menghubungkan objek tersebut. Setelah hal itu dilakukan maka langkah selanjutnya adalah menentukan sederaetan pengujian yang membuktikan bahwa “semua objek memiliki hubungan yang diharapakan satu dengan yang

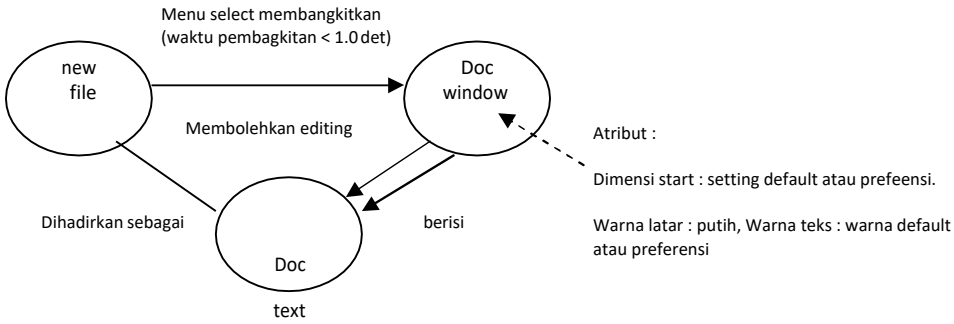
lainnya". Dengan kata lain, pengujian perangkat lunak dimulai dnegan membuat grafik dari objek-objek yang penting dan hubungan objek-objek serta kemudian memikirkan sedereetan pengujian yang akan mencakup grafik tersebut sehingga masing-masing objek dan hubungan digunakan dan kesalahan ditemukan. Untuk melakuka langkah-langkah tersebut, perekayasa perangkat lunak memulianya dengan membuat suatu grafik, sekkumpulan simpul yangmerepresentasikan objek; link yang merepresentasikan hubungan antar objek; node weight yang menggambarkan properti dari suatu simpul (misal : nilai data tertentu atau tingkah laku keadaan) dan links weight yang menggambarkan beberapa karakteristik suatu link.

Representasi simboki dari grafik diperihatkan pada gambar 11. Simpul-simpul direpresentasikan sebagai lingkaran yang dihubungkan oleh ling yang memakai sejumlah bentuk yang berbeda. Link terarah (direpresentasikan oleh sebuah anak panah) menunjukkan bahwa hubungan bergerak hanya dalam satu arah. Link dua arah yang disebut link simetris, mengimplikasikan bahwa hubungan tersebut berlaku dalam dua arah. Link paralel digunakan pada saat sejumlah hubungan yang berbeda dibangun di antara simpul-simpul grafik.

Sebagai contoh sederhana, perhatikan bagian dari suatu grafik untuk aplikasi pengolah kata gambar 12 :



Gambar 11. Notasi Grafik



Gambar 12. Contoh sederhana Notasi Grafik

Objek #1 = new file menu
select Objek #2 = document
window Objek #3 =
document text

Seperti diperlihatkan pada gambar, pemilihan menu New File menghasilkan document window. Node weight dari document window memberikan daftar atribut window tersebut yang akan diharapakan pada saat window dimunculkan. Link weight menunjukkan bahwa window harus dimunculkan dalam kurang dari 1.0 detik. Link tak terarah membangun hubungan simetris di antara new file select menu dan text document dan lion parale menunjukkan hubungan antara document window dan document text. Kenyataanya grafik yang jauh lebih detail harus dimumculkan sebagai pendahuluan ke desain test case. Perekayasa perangkat lunak kemudian melakukan test case dengan melewatkan grafik tersebut dan mencakup masing-masing dari hubugan yang diperlihatkan. Test case itu didesain untuk menemukan kesalahan pada berbagai hubungan.

Beizer menggambarkan sejumlah metode pengujian behavioral yang dapat menggunakan grafik :

- **Pemodelan aliran transaksi**. Simpul-simpul merepresentasikan langkah-langkah pada beberapa transkasi (misal, langkah yang diperlukan untuk membuat reservasi pesawat dengan menggunakan layanan online) dan link merepresenasikan hubungan logis di antara langkah-langkah (misal flight.information.input diikuti oleh validation/availibility.processing). Diagram aliran data dapat digunakan untuk membantu menciptakan grafik tipe ini.
- **Pemodelan keadaan terbatas**. Simpul-simpul merepresentasikan keadaan perangkat lunak yang dapat diamati oleh pemakai yang berbeda (misal, masing-masing "layar muncul sebagai sebuah juru tulis entri order yang mengambil urutan telepon), dan link merepresentasikan transisi yang terjadi untuk bergerak dari satu keadaan ke keadaan lainnya (misal : order-information diperiksa selam inventory-availibiity-look-up diikuti dengan customer billing-information-input). Diagram transisi keadaan dapat digunakan untuk membantu membuat grafik tipe ini.
- **Pemodelan aliran data**. Simpul-simpul merupakan objek data, sementara link adalah transformasi yang terjadi untuk menterjemahkan satu objek data ke objek data yang lain. Contoh, simpul FICA.tax.withheld (FTW) dihitung dari Gross.wages (GW) dengan menggunakan hubungan FTW =0,062 x GW.
- **Pengujian timing**. Simpul-simpul merepresentasikan objek program dan link adalah hubungan sekuensial antara objek-objek tersebut. Link weight digunakan untuk menentukanwaktu eksekusi yang dibutuhkan pada saat program mengeksekusi.

Pengujian graph-base dimulai dengan definisi semua simpul dan node weight, dimana objek dan atribut diidentifikasi. Model data dapat digunakan sebagai titik awal, tetapi penting untuk dicatat bahwa banyak simpul dapat merupakan objek program (tidak secara eksplisit direpresentasikan dalam model data). Untuk memeberikan indikasi dari titik mulai dan berhenti untuk grafik tersebut, berguna sekali bila simpul masuk dan keluarnya ditentukan.

Sekali simpul diidentifikasi maka link dan link weight harus dibangun. Secara umum, link harus diberi nama, meskipun link yang merepresentasikan aliran kontrol di antara objek program tidak perlu diberi nama.

Dalam banyak kasus, model grafik dapat memiliki loop. Pengujian loop dapat diaplikasikan pada tingkat tingkah laku (black box). Grafik tersebut akan membantu mengindentifikasi loop-loop yang memerlukan pengujian tersebut.

Masing-masing hubungan dipelajari secara terpisah sehingga test case dapat dilakukan. Transitivitas hubungan sekuensial dipelajari untuk menentukan bagaimana pengaruh hubungan tersebut menyebar pada objek yang ditentukan pada suatu grafik. Transitivitas dapat digambarkan dengan memperhatikan tiga objek : X , Y dan Z. Perhatikan hubugnan berikut ini

:

X diperlukan untuk
menghitung Y Y diperlukan
untuk menghitug Z

Sehingga dibangun hubungan transitif antara X
dan Z X diperlukan untuk menghitung Z

Berdasarkan hubungan transitif ini, pengujian untuk menemukan kesalahan dala penghitungan Z harus memepertimbangkan berbagai harga untuk X maupun Y.

Simetri hubungan (lin grafik) juga merupakan panduan penting ke desain test case. Bila suatu link benar-benar dua arah (simetris), maka penting untuk menguji fitur tersebut. Fitur UNDO pada berbagai aplikasi personal komputer mengimplementasikan simetri yang terbatas, yaitu bahwa UNDO mengijinkan suatu aksi dihapuskan setelah aksi itu diselesaikan. Hal itu harus diuji secaramendalam dan semua perkecualian (yakni tempat dimana UNDO tidak dapat digunakan) harus dicatat. Akhirnya, setiap simpul pada grafik harus memiliki hubungan yang mengarah kepada dirinya sendiri : pada dasarnya dalah loop “no anction” atau”action null”. Hubungan refleksif itu juga harus diuji.

Pada saat disain test case dimulai, maka sasaran pertamanya adalah mencapai cakupan simpul (node coverage). Ini berarti bahwa pengujian harus didesain untuk memeprlihatkan bahwa tidak aa simpul yang dnegna sembrono diabaikan dan bahwa node weight (atribut objek) adalah benar.

Selanjutnya menenkankan cakupan link. Masing-masing hubungan diuji berdsarkan propertinya. Misal, hubungan simetri untuk memperlihatkan bahwa pada dasarnya hubungan itu memiliki dua arah. Hubungan transitif diuji untuk memeperlihatkan bahwa ada transitivitas. Hubunga relfreksif diuji untuk memastikan bahwa loop null ada. Bila link weight telah ditentukan, pengujian dilakukan utnuk menunjukkan bahwa weight telah ditentukan, pengujian dilakukan untuk menunjukkan bahwa weight itu valid. Akhirnya, pengujian loop dipanggil.

1.8.2. Partisi Ekivalensi

Partisi ekivalensi adalah metode pengujian black-box yangmembagai domain input dari sautu program ke dalam kelas data dari mana test case dapat dialkukan. Test case yang ideal mengungkap kelas kesalahan (misal, pemrosesan yang tidak benar terhadap semua data karakter) yang akan memerlukan banyak kasus untuk dieksekusi sebelum kesalaha umum diamati. Partisi ekivalensi berusaha menentukan sebuah test case yang mengungkap kelas-kelas kesalahan, sehingga mengurangi jumlah total test case yang harus dikembangkan.

Desain test case untuk partisi ekivalensi didasarkan pada evaluasi terhdap kelas ekivalensi untuk suatu kondisi input. Dengan menggunakan konsep yan telah dijelaskan pada bagian sebelumnya, bila serangkaian objek dapat di-link oleh hubungan yan gsimetris, transitif dan refleksif,

maka ada kelas ekivalensi. Kelas ekivalensi merepresentasikan serangkaian keadaan valid atau invalid untuk kokdisi input. Secara khusus, suatu kondisi input dapat berupa harga numeris, suatu rentang harga atau serangkaian haraga terkait atau sebuah kondisi Boolean. Kelas ekivalensi dapat ditentukan sesuai pedoman berikut ini :

1. Bila kondisi input menentukan suatu range, maka satu kelas ekivalensi valid dan dua yang invalid ditentukan
2. Bila suatu kondisi input membutuhkan suatu harga khusus, maka satu kelas ekivalensi vaild

dan uda yang invalid ditentukan

3. Bila suatu kondisi menentukan anggota suatu himpunan, maka satu kelas ekivalensi valid atau dua yang invalid ditentukan.
4. Bila suatu kondisi input adalah Boolean, maka satu kelas valid dan satu yang invalid ditentukan.

Sebagai contoh, perhatikan data yang dijaga sebagai bagian dari dari suatu aplikasi perbankan otomatis. Pemakai dapat "menghubungi" bank tersebut dengan menggunakan komputer personalnya, sebuah password enam digit dan diikuti dengn serangkaian perintah kata kunci yang memicu berbagai fungsi perbankan. Perangkat lunak yang dipasok untuk aplikasi perbankan menerima data dalam bentuk :

Kode area-kosong atau tiga nomor digit
Prefik – tiga nomor digit tidak mulai dengan 1
atau 0 Sufik - empat nomor digit
Password – enam nilai alfanumeris digit
Perintah "cek", "deposit","bayar pajak" dan sebagainya.

Kondisi input yang sesuai dengan masing-masing elemen data untuk aplikasi perbankan dapat ditentukan sebagai :

Kode area	:	Kondisi input, Boolean – kode area mungkin atau mungkin tidak ada kondisi. Input range – nilai yang ditentukan antara 200 dan 999 dengan perkecualian khusus.
Prrefiks	:	Kondisi input range – harga yang ditetapkan > 200 dengan tanpa
digit 0 Sufiks	:	Kondisi input, harga – panjang empat digit
Password	:	Kondisi input, Boolean-password dapat ada atau tidak ada Kondisi input, harga – antrian enam karakter
Perintah	:	Kondisi input, himpunan – berisi perintah yang sudah ditulis di atas.

Dengan mengaplikasikan pedoman untuk derivasi kelas ekivalensi, test case untuk masing- masing item data domain input dapat dikemabngkan dan dieksekusi. Test case dipilih sehingga jumlah atribut yang terbesar dari suatu kelas ekivalensi digunakan saat itu juga.

1.8.3. Analisis Nilai Batas

Kareana alasan yang tidak jelas, jumlah kesalahan yang lebih besar terjadi pada batas domain input daripada di pusat. Karena itulah analisis nilai batas/boundary value analysis (BVA) telah dikembangkan sebgai teknik pengujian. Analisis nilai batas memunculkan pemilihan test case yang menggunakan nilai batas.

Analisis nilai batas adalah teknik desain proses yang melengkapi partisi ekivalensi. Daripada memilih sembarang elemen kelas ekivalensi, BVA lebih mengarah kepada pemilih test case pada "edge" dari kelas. Daripada hanya berfokus pada kondisi input, BVA melakukan test case dari domain output.

Dalam banyak hal, pedoman untuk BVA sama dengan yang diberikan untuk partisi ekivalensi:

1. Bila suatu kondisi input mengkhususkan suatu range dibatasi oleh nilai a dan b,maka test case harus didesain dengan nilai a dan b, persis di atas dan di bawah a dan b secara berkesesuaian.
2. Bila suatu kondisi input mengkhususkan sejumlah nilai, maka test case harus dikembangkan dengan menggunakan jumlah minimum dan maksimum. Nilai tepat di atas dan di bawah

minimum dan maksimum juga diuji.

3. Pedoman 1 dan 2 diaplikasikan ke kondisi output. Contohnya anggap bahwa suhu vs.tabel tekanan diperlukan sebagai output dari program analisis rekayasa. Test case harus didesain untuk menciptakan laporan output yang menghasilkan jumlah minimum (dan maksimum) entri tabel yang diijinkan.
4. Bila struktur data program telah memesan batasan (misal, suatu array memiliki suatu batas yang ditentukan dari 100 entri), pastikan untuk mendesain test case yang menggunakan struktur data pada batasannya.

Sebagian besar perekayasa perangkat lunak secara intuitif melakukan BVA sampai beberapa tingaktaa. Dengan mengaplikasikan pedoman tersebut, pengujian batasan akan lebih lengkap, sehingga kemungkinan untuk berhasil mendeteksi kesalahan menjadi lebih besar.

1.8.4. Pengujian Perbandingan

Ada banyak situasi (seperti avionik pesawat udara) dimana reliabilitas perangkat lunaknya sangat kritis. Dalam aplikasi semacam itu, perangkat lunak dan perangkat keras redundan sering digunakan untuk meminimalkan kemungkinan kesalahan. Pada saat perangkat lunak redundan dikembangkan, tim rekayasa perangkat lunak yang terpisah mengembangkan versi-versi independen dari suatu aplikasi dengan menggunakan spesifikasi yang sama. Dalam situasi semacam itu, setiap versi dapat diuji dengan data uji yang sama untuk memastikan bahwa semua versi memberikan output yang identik. Kemudian semua versi dieksekusi secara paralel dnegan perbandingan real time hasil untuk memastikan konsistensi.

Dengan pelajaran mengenai sistem redundan, para peneliti mengusulkan agar versi perangkat lunak independen dikembangkanbagi aplikasi kritis, bahkan bila hanya sebuah versi tunggal saja yang akan digunakan pada sistem berbasis komputer yang disampaikan. Versi independen membentuk basis teknik pengujian black box yang disebut pengujian perbandingan atau pengujian back to back.

Bila implementasi bertingkat dari spesifikasi yang sama telah dihasilkan, maka test case yang didesain dengan menggunakan teknik black box yang lain (misal partisi ekivalensi) diberikan sebagai input untuk masing-masing versi perangkat lunak tersebut. Bila output dari masing-masing versi sama, maka diasumsikan bahwa implementasinya benar. Bila outputnya berbeda, maka masing- masing dari aplikasi itu diperiksa untuk menentukan cacat pada suatu versi atau agar perbedaan itudapat lebih jelas. Pada sebagian besar kasus, perbandingan output dapat dilakukan dengan suatu peranti yang diotomatisasi.

Pengujian perbandingan tidaklah mudah. Bila spesifikasi dari mana semua fungsi telah dikembangkan mengandung kesalahan, maka semua versi kemungkinan besar merefleksikan kesalahan. Lagi pula jika masing-masing versi independen identik tetapi tidak benar, maka pengujian kondisi akan gagal mendeteksi kesalahan.

1.9. Pengujian Untuk Aplikasi dan Lingkungan Khusus

Pada saat perangkat lunak komputer menjadi semakin kompleks,maka kebutuhan akan pendekatan pengujian yang khusus juga makin berkembanga. Metode pengujian black-box dan white box yang dibicarakan sebelumnya dapat diaplikasikan pada semua lingkungan, arsitektur dan aplikasi, tetapi kadang-kadang dalam pengujian diperlukan pedoman dan pendekatan

yang unik.

Pada bagian ini akan dibahas pedoman pengujian bagi lingkungan arsitektur dan aplikasi khusus yang umumnya ditemui oleh para perekayasa perangkat lunak.

1.9.1. Pengujian GUI

Graphical User Interfaces (GUI) menyajikan tantangan yang menarik bagi para perekayasa. Karena komponen reusable berfungsi sebagai bagian dari lingkungan pengembangan GUI, pembuatan interface pemakai telah menjadi hemat waktu dan lebi teliti. Pada saat yang sama, kompleksitas GUI telah berkembanga, menimbulkan kesulitan yang lebih besar di dalam desain dan eksekusi test case.

Karena GUI modern memiliki bentuk dan cita rasa yang sama maka dapat dilakukan sederetan pengujian standar. Pertanyaan berikut dapat berfungsi sebagai panduan untuk serangkaian pengujian generik untuk GUI :

Untuk windows :

- Apakah window akan membuka secara tepat berdasarkan tipe yang sesuai atau perintah berbasis menu ?
- Dapatkah window di-resize, digerakkan atau digulung ?
- Apakah semua isi data yang diisikan pada window dapat dituju dengan tepat dengansebuah mouse, function keys, anak panah penunjuk dan keyboard ?
- Apakah window dengan cepat muncul kembali bila dia ditindih dan kemudian dipanggil lagi ?
- Apakah semua fungsi yang berhubungan dengan window dapat diperoleh bila diperlukan ?
- Apakah semua fungsi yang berhubungan dengan window operasional ?
- Apakah semua menu pull down, tool bar,scroll bar, kotak dialog, tombol ikon dan kontrol yang laina dapat diperoleh dan dengan tepat ditampilkan untu window tersebut ?
- Pada saat window bertingkat ditampilkan, apakah nama window tersebut direpresentasikan secara tepat ?
- Apakah window yang aktif disorot secara tepat ?
- Bila multitasking digunakan, apakah semua window diperbarui pada waktu yang sesuai ?
- Apakah pemilihan mouse bertingkat atau tidak benar di dalam window menyebabkan efek samping yang tidak diharapkan ?
- Apkaha audio prompt dan atau color prompt ada di dala window atau sebagai konsekuensi dari operasi windoe disaajikan menurut spesifikasi ?
- Apakah window akan menutup secara tepat ?

Untuk menu pull down dan operasi mouse :

- Apakah menu bar yang sesuai ditampilkan di dalam konteks yang sesuai ?
- Apakah menu bar aplikasi menampilkan fitur-fitur yang terkait dengan sistem (misal tampilan jam) ?
- Apakah operasi menu pull down bekerja secara tepat ?
- Apakah menu breakway, palette dan tool bar bekerja secara tepat ?
- Apakah semua fungsi menu dan subfungsi pulldown didaftar secara tepat ?
- Apakah semua fungsi menu dapat dituju secara tepat oleh mouse ?
- Apakah typface, ukuran dan format teks benar ?
- Mungkinkah memanggil masing-masing fungsi menu denganmenggunakan perintah berbasis teks alternatif ?
- Apakah fungsi menu disorot berdasarkan konteks operasi ang sedang berlangsung di dalam

suatu window ?

- Apakah semua menu function bekerja seperti diiklankan ?
- Apakah nama-nama menu funvtion bersifat self explanatory ?
- Apakah help dapat diperoleh untuk masing-masing item menu, apakah dia sensitif terhadap konteks ?
- Apakah operasi mouse dikenali dnegna baik pada seluruh konteks interaktif ?
- Bila klik ganda diperlukan, apakah operasi dikenali di dalam konteks ?
- Jika mouse mempunyai tombol ganda, apakah tombol itu dikenali sesuai konteks ?
- Apakah kursor, indikator permosesan (seperti jam) dan pointer secara tepat berubah pada saat operasi yang berbeda dipanggil ?

Entri Data :

- Apakah entri data alfanumeris dipantulkan dan diinput ke sistem ?
- Apakah mode grafik dari entri (misal, slide bar) bekerja dengan baik ?
- Apakah data invalid dikenali dengan baik ?
- Apakah pesan input data sangat pintar ?

Sebagai tambahan untuk pedoman tersebut, grafik pemodelan keadaan yang terbatas dapat digunakan untuk melakukan sederetan pengujian yang menekankan objek program dan data spesifik yang relevan dengan GUI.

Karena sejumlah besar permutasi yang bekesesuaian dengan operasi GUI, maka pengujian harus didekati denga menggunakan peranti otomatis. Sudah banyak peranti pengujian GUI yang muncul dipasaran selama beberapa tahun terakhir.

1.9.2. Pengujian Arsitektur Client Server

Arsitektur client/server (C/S) menghadirkan tantangan yang berarti bagi para penguji perangkat lunakan. Sifat terdistribusi dari lingkungan client/server, masalah kinerja yang berhubungan dengan pemrosesan transaksi, kehadiran potensial dari sejumlah platform perangkat keras yang berbeda, kompleksitas komunikasi jaringan,kebutuahn aka layanan client multipel dari suatu database terpusat dan persyaratan koordinasi yang dibebabkan pada server, semua secara bersama-sama membuat pengujian terhadap arsitektur C/S dan perangkat lunak yang ada didalamnya menjadi jauh lebih sulit daripada pengujian aplikasi yang berdiri sendiri. Kenyataannya studi industri terakhir menunjukkan pertambahan berarti di dalam waktu pengujian dan biaya ketika lingkungan C/S dikembangkan.

1.9.3. Pengujian Dokumentasi dan Fasilitas Help

Istilah “pengujianperangkat lunak” memunculkan citra terhdapa sejumlah besar test case yang disiapkan untuk menggunakan program komputer dan data yang dimanipulasi oleh program. Dengan melihat kembali definisi perangkat lunak yang disajikan di awal, penting untuk dicatat bahwa pengujian harus berkembang ke elemen ketiga dari konfigurasi perangkat lunak, yaitu “dokumentasi”.

Kesalahan dala dokumentasi dapat menghancurkan penerimaan program seperti halnya kesalahan pada data atu kode sumber. Tidak ada yang lebih membuat frustasi dibanding mengikuti tuntuan pemakai secara tepat dan mendapatkan hasil atau tingkah laku yang tidak sesuai dengan yang diprediksi oleh dokumen. Karena itulah pengujian dokumentasi harus menjadi suatu bagian yang berarti dari setiap rencana pengujian perangkat lunak.

Pengujian dokumentasi dapat didekati dalam dua fase. Fase pertama, kajian teknis formal yang menguji kejelasan editorial dokumen. Fase kedua, live test, menggunakan dokumentasi dalam kaitannya dengan penggunaan program aktual.

Live test untuk dokumentasi dapat didekati dengan menggunaka teknik yang analog dnegan berbagai metode pengujian black box. Pengujian graph-based ddapat digunakan untuk menggambarkan penggunaan program tersebut; partisi ekivalensi dan analisis nilai batas dapat

digunakan untuk menentukan berbagai kelas input dan interaksi yang sesuai. Kegunaan program kemudian ditelusuri pada seluruh dokumen :

- Apakah dokumen tersebut secara akurat menggambarkan bagimana menyelesaikan masing-masing mode penggunaan ?
- Apakah deskripsi dari masing-masing urutan interaksi akurat ?
- Apakah contoh-contoh akurat ?
- Apakah terminologi, gambaran menu dan respon sistem konsisten dengan program aktual ?
- Apakah relatif mudah untuk menempatkan panduan di dalam dokumentasi ?
- Dapatkah trouble shooting dilakukan dnegan mudah dnegan dokumentasi ?
- Apakah tabel dokumen dari isi dan indeks akurat dan lengkap ?
- Apakah desain dokumen (layout, typeface, indetasi, grafik) kondusif untuk pemahaman dan asimilasi cepat terhadap informasi ?
- Apakah semua pesan kesalahan ditampilkan bagi pemakai dan digambarkan secara lebih detail di dalam dokumen ?
- Bila link hiperteks digunakan, apakah mereka akurat dan lengkap ?

Satu-satunya cara yang dapat berjalan untuk menjawab pertanyaan-pertanyaan tersebut adalah dengan menggunakan bagian ketiga yang independen (misal : pemakai yang diseleksi) yang menguji dokumentasi di dalam konteks kegunaan program. Semua diskrepansi dicatat dan area ambiguitas atau kelemahan dokumen ditentuka untuk penulisan ulang yang potensial.

1.9.4. Pengujian Sistem Real-Time

Sifat asinkron dan tergantung waktu yang ada pada banyak aplikasi real time menambahkan elemen baru yang sulit dan potensial untuk bauran pengujian-waktu. Tidak hanya desainer teset case yang harus memeprtimbangkan test case balck box dan white box tetapi juga penanganan kejadian (yaitu pemrosesan interupsi), timing data dan paralelisme tugas-tugas (proses) yang menangani data. Pada banyak situasi, data pengujian yang diberikan pada saat sebuah sistem real time ada dalam satu keadaan akan menghasilkan pemrosesan yang baik, sementara data yang sama yang diberikan pada saat sistem berada dalam keadaan yang berbeda dapat menyebabkan kesalahan.

Contohnya, perangkat lunak real time yang mengontrol alat foto kopi yang baru menerima interupsi operator (yakni operator mesin menekan kunci kontrol seperti reset) dengan tanpa kesalahan pada saat mesin sedang membuan kopian. Interupsi operator yang sama ini bila diinputkan pada saat mesin ada dalam keadaan "jammed" akan menyebabkan sebuah kode diagnostik yang menunjukkan lokasi jam yang akan hilang (kesalahan).

Hubungan erat perangkat lunak real time dan lingkungan perangkat kerasnya dapat juga menyebabkan pengaruh kegagalan perangkat keras pada pemrosesasn perangkat lunak. Kesalahan semacam itu dapat sangat sulit untuk bersimulasi secara realistis.

Metode desain test case yang komprehensif untuks istem real time harus berkembang. Tetapi strategi emapat langkah berikut dapat diusulkan :

- **Pengujian tugas**. Langkah pertama dalam pengujian perangkat lunak real time adalah menguji masing-masing tugas secara independen, yaitu pengujian white box dan black box yang didesain dan dieksekusi secara independen bagi masing-masing tugas. Masing-masing tugas dieksekusi secara independen selama pengujian ni. Pengujian tugas mengungkap kesalahan di dalam logika dan fungsi, tetapi tidak akan mengungkap timing atau kesalahan tingkah laku.

- **Pengujian tingkah laku**. Dengan menggunakan model yang idciptakan dengan peranti CASE, dimungkinkan untuk mensimulasi tingkah laku sistem real time dan menguji tingkah lakunya sebagai konsekuensi dari event eksternal. Aktivitas analisis ini dapat berfungsi sebgai dasar bagi desain test case yang dilakukan pada saat perangkat lunak real time dibangun. Dengan menggunakan teknik yang sama dengan partisi ekuivalensi, event-event (misal interupsi, signal, kontrol, data) dikategorikan untuk pengujian. Sebagai contoh, event untuk mesin fotokopi dapat merupakan inerupsi pemakai (misal, pencacah reset), interupsi mekanis (misal, paper jammed), interupsi sistem (misal tone rendah) dan mode kegagalan (misal roller yang terlalu panas). Masing-masing event tersebut diuji secara individual dan tingkah laku sistem yang dapat dieksekusi diperiksa untuk mendeteksi kesalahan yang terjadi sebagai akibat pemrosesasn yang terkait denga event tersebut. Perilaku model sisetm (dikembangkan selama aktivias analisis) dan perangkat lunaka yang dapat dieksekusi dapat dibandingkan untuk penyesuaian. Sekali masing-masing kelas event telah diuji,maka event-event disajikan pada sistem dalam urutan acak dan dengan frekuensi acak. Perilaku sistem diuji untuk mendeteksi kesalahan perilaku.
- **Pengujian antar tugas**. Setelah kesalahan-keaslahan pada tugas individual dan pada perilaku sistem diisolasi, maka pengujian beralih kepada kesalahan yang berkaitan dengan waktu. Tugas-tugas asinkronous yang dikenali untuk saling berkomunikasi diuji dnegna tingkat data yang berbeda dan pemrosesan dipanggil untuk menentukan apakah kesalahan sinkronisasi antar tugas akan terjadi. Sebagai tambahan, tugas-tugas yang berkomuniaksi melalui antrian pesan atau penyimpanan data, diuji untuk menemukan kesalahan dalam ukuran area penyimpanan data tersebut.
- **Pengujian sistem**. Perangkat lunak dan perangkat keras diintegrasi dan suatu rentang penuh dari pengujian sistem dilakukan di dalam usahan mengungkap keslahan pada interface perangkat lunak/perangkat keras.

Sebagian besar sitem real time memproses interupsi, karena itu pengujian penanganan terhadap kejadian-kejadian. Boolean merupakan hal yang penting. Dengan menggunakan diagram keadaan transisi dan spesifikasi kontrol, penguji mengembangkan daftar semua interupsi yang mungkin dan pemroesan yang terjadi sebagai konsekuensi dari interupsi. Kemudain pengujian didesain untuk menilai karakteristik sistem berikut ini :

- ✓ Apakah priortias interupsi ditetapkan dan ditangani secara tepat ?
- ✓ Apakah pemrosesan untuk masing-masing interupsi ditangani dengan tepat ?
- ✓ Apakah kinerja (misal waktu pemrosesasn) dari masing-masing prosedur penangan interupsi sesuai dengan persyaratan) ?
- ✓ Apakah volume interupsi yang tinggi yang terjadi pada waktu kritis menimbulkan masalah di dalam fungsi atau kinerja ?

Sebagai tambahan, area data global juga digunakan untuk mentransfer informasi sebagai bagian dari pemrosesan interupsi yang harus diuji untuk menilai potensi munculnya efek samping.

PERTEMUAN VI

PENGUJIAN DAN IMPLEMENTASI SISTEM

BUG TRACKING DATABASE

MODUL KULIAH

UNIVERSITA BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 6 (Lima)	Waktu :

Modul	6 (Lima)
Topik	Bug Tracking Database
Sub Topik	-
Materi	• Bug • Traditional Report Bug • Bug Tracking Database • Keuntungan Menggunakan Bug Tracking Database • Laporan Bug • Membuat dan merancang database
Tujuan	• Mahasiswa dapat menerangkan manfaat dari alat bantu pengujian Bug Tracking Database. • Mahasiswa dapat menghasilkan laporan deskripsi kesalahan dengan baik. • Mahasiswa dapat menguraikan siklus hidup pengelolaan kesalahan.

A. BUG

Bug adalah suatu kesalahan pada software yang menyebabkan program tidak berfungsi dengan semestinya

Error adalah perbedaan atau kesalahan perhitungan/prediksi yang tidak sesuai dengan yang diharapkan, sehingga program/software tersebut tidak bisa melakukan apa yang diinginkan.

Bug Tracking adalah sebuah software yang dirancang untuk membantu Quality Ansurance dan membantu programmer untuk mengarsipkan laporan bug & Error sebuah software.

Bug Tracking Software pada umumnya diintegrasikan dengan software manajemen projek.

B. Tradisional Report Bug

Cara Konvensional Melaporkan Bug-Error ke Programmer :

1. Teriak-Bicara Langsung ke Programmer
2. Ditulis di Kertas
3. Dikirim Melalui Email
4. Dikirim melalui Instan Messanger

Cara Reporting bug/error tradisional akan menjadikan catatan catatan bug/error berserakan dimana-mana dan sulit untuk diarsipkan.

Manfaat Bug Tracking (Pelacakan Bug) :

1. Menghindari catatan bug/error yang hilang
2. Menghindari bug yang terlewat diperbaiki.
3. Mengurangi kemungkinan duplikasi bug
4. Mengingatkan programmer untuk menyelesaikan Bug/error

C. Bug Tracking Database

Merupakan alat bantu yang diperlukan oleh organisasi pengujian (testing Organization) dalam memegang peranannya untuk melakukan pengujian.

Dengan mendokumentasikan setiap kesalahan pada suatu sistem dengan baik, maka bug (kesalahan) dapat segera diperbaiki dan meningkatkan kualitas produk.

Sebagai bukti nyata atas masalah[2] kualitas yang pernah terjadi yang kemudian dpt digunakan sebagai referensi

D. Keuntungan Penggunaan Bug Tracking Databases

Adapun keuntungan menggunakan Bug Tracking Databases adalah :

1. Mengkomunikasikan *bug* dengan jelas. Laporan kesalahan yang ditulis dengan baik sesuai standar akan menjelaskan suatu masalah lebih baik daripada menggunakan email atau catatan biasa.
2. Memudahkan pemantauan dan pencarian bug yang pernah terjadi dengan melakukan penomoran bug secara otomatis
3. Proses perbaikan dapat dilakukan berdasarkan prioritas dan efek bug pada system.
4. Pengelolaan bug dalam suatu siklus pengelolaan dapat dilakukan dengan lebih baik. Untuk memantau agas bug yang ada dpt diperbaiki secepat mungkin sesuai dengan prioritasnya.
5. Memberikan informasi baru bagi pengembang, tester, dan manajer. Bug Tracking Databases yang dirancang dengan baik akan memberikan gambaran histori yang baik yang dapat digunakan sebagai referinsi kemudian hari.
6. Sumber informasi bagi *support department.*

.

E. Laporan Bug

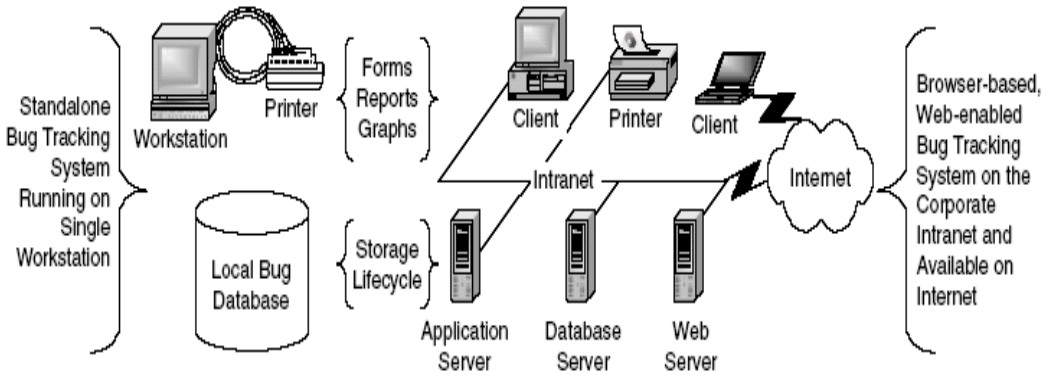


Figure 4.1 Simple and sophisticated bug tracking system architectures.

Gambar diatas merupakan arsitektur sistem pelacakan kesalahan. Adapun Deskripsi Kesalahan (Failure Description) dimaksud adalah terdiri dari 3 bagian :

- ✓ **Summary**: satu atau dua buah kalimat yang menjelaskan suatu bug dan memberikan informasi mengenai dampak yang akan dialami oleh pengguna atau *customer*
- ✓ **Steps to reproduce**: menjelaskan langkah[2] untuk menimbulkan *bug* tersebut.
- ✓ **Isolation**: konfirmasi bahwa informasi dan hasil yang diperoleh tester adalah problem yang nyata dan menjelaskan faktor2 yang menyebabkan bug tersebut

Bagian dari *bug report* yang berisi deskripsi mengenai suatu bug yang terjadi.

Merupakan alat bagi tester untuk mengkomunikasikan suatu masalah kepada *programmer* atau *developer team.*

Laporan bug adalah dokumen teknis yang menggambarkan berbagai gejala atau mode kegagalan yang terkait dengan satu bug.

Laporan bug yang baik memberikan informasi tentang apa yang tim manajemen proyek perlu memutuskan kapan dan apakah memperbaiki masalah, menangkap informasi yang dibutuhkan pemrogram untuk memperbaiki dan debug masalah.

Sistem pelacakan bug adalah beberapa program atau aplikasi yang memungkinkan tim proyek melaporkan, mengelola, dan menganalisis laporan bug dan tren bug.

Ketentuan penulisan Failure Description sebagai berikut :

1. Laporan bug harus ditulis pada saat melaksanakan testing.
2. Laporan harus akurat, lengkap dan ringkas. Tidak terlalu singkat tetapi tidak terlalu bertele-tele.

3. Menjelaskan pada pembaca apa yang ditemukan oleh tester, bukan apa yang dilakukan oleh tester

Contoh Laporan Bug Yang Baik :
SpeedyWriter for Wndows 98 on the open file if arial font is selected

1. Step To Reproduce
   - ✓ Open Speedwriter and create a new file
   - ✓ Type in two or more lines of random text
   - ✓ Highlight the text, pull down the font menu and select arial
   - ✓ Text is transformed into meaningless garbage
   - ✓ I was able to reproduce this problem three out of three tries.

2. Isolation
   On the vague suspicion that this was just a formatting provblem, i saved the file,closed spedywriter and reopened the file. The garbage remained. If ou save the file before arializing the contents, the bug does not accur with existing files.
   This happenns only under windows 98. This bug doesn't occur with other font.

Laporan bug adalah dokumen teknis yang menggambarkan berbagai gejala atau mode kegagalan yang terkait dengan satu bug.

Laporan bug yang baik memberikan informasi tentang apa yang tim manajemen proyek perlu memutuskan kapan dan apakah memperbaiki masalah, menangkap informasi yang perlu dipecahkan dan diprogram oleh programmer.

Sistem pelacakan bug adalah beberapa program atau aplikasi yang memungkinkan tim proyek melaporkan, mengelola, dan menganalisis laporan bug dan tren bug.

Summary
Arial, Wingdings, and Symbol fonts corrupt new files.

Steps to Reproduce
1. Started SpeedyWriter editor, then created new file.
2. Typed four lines of text, repeating "The quick fox jumps over the lazy brown dog" each time.
3. Highlighted all four lines of text, then pulled down the font menu, and selected Arial.
4. All text converted to control characters, numbers, and other apparently random binary data.
5. Reproduced three out of three tries.

Isolation
New to build 1.1.018; same test case passed against builds 1.1.007 (System Test entry) through 1.1.017.
Reproduced with same steps using Wingdings and Symbol fonts.
On vague suspicion this was a formatting problem, saved file, closed SpeedyWriter and reopened file, but data corruption remained.
Saving file before changing font prevents bug.
Bug does not occur with existing files.
Only happens under Windows 98, not Solaris, Mac, or other Windows flavors.

Figure 4.2 A good SpeedyWriter bug report.

Summary
SpeedyWriter has trouble with Arial.

Steps to Reproduce
1. Open SpeedyWriter.
2. Type in some text.
3. Select Arial.
4. Text gets screwed up.

Isolation
N/A

Figure 4.3 A vague, incomplete bug report.

Terdapat 10 Langkah membuat laporan Bug yang paling baik yaitu :

1. Structure
2. Reproduce
3. Isolate
4. Generalize
5. Compare
6. Summarize
7. Condense
8. Disambiguate
9. Neutralize
10. Review

F. Membuat Dan Merancang Database

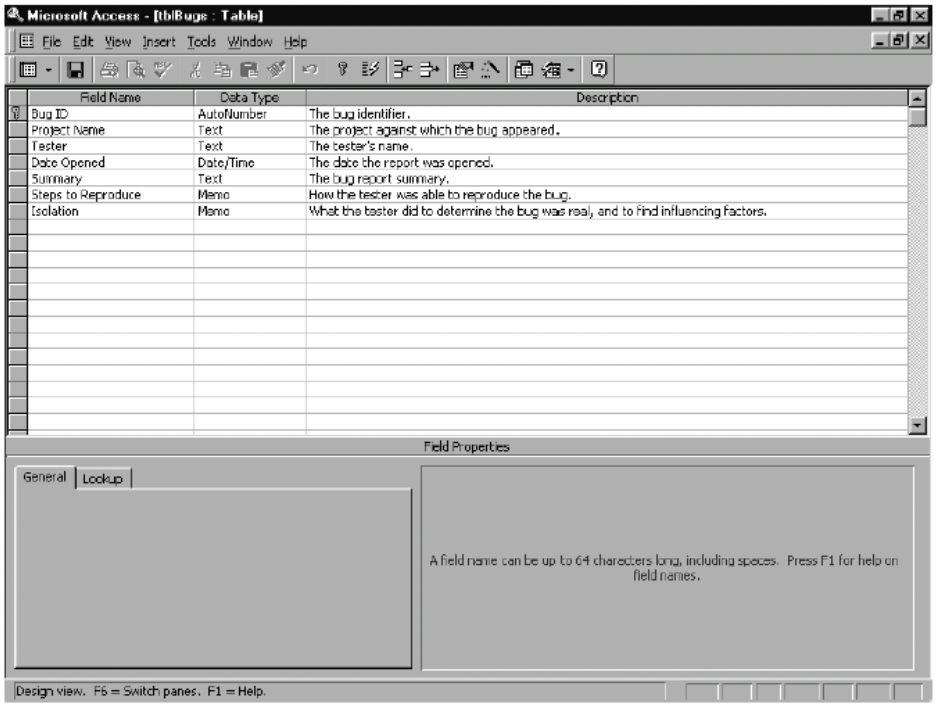


Figure 4.5 The design for a basic bug tracking database.

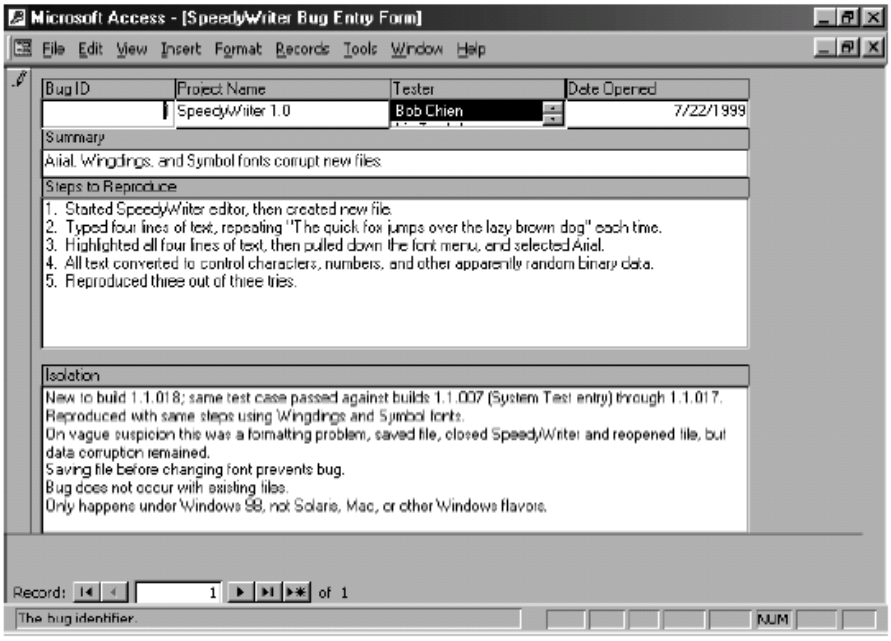


Figure 4.6 A bug report in the SpeedyWriter bug database, using a bug entry form.

Pemeringkatan yang terpenting :

1. Tingkat Keparahan

- ✓ Kehilangan Data, kerusakan perangkat keras atau masalah keamanan
- ✓ Hilangnya fungsi tanpa solusi
- ✓ Hilangnya fungsi dengan solusi
- ✓ Sebagian kehilangan fungsi
- ✓ Dapat di Custum dengan mudah atau sepele

2. Tingkat Prioritas
   - ✓ Kehilangan total keseuruhan nilai sistem
   - ✓ Hilangnya nilai sistem yang tidak dapat diterima
   - ✓ Kemungkinan pengurangan nilai sistem yang dapat diterima
   - ✓ Pengurangan nilai sistem yang dapat diterima
   - ✓ Penurunan nilai sistem yang tak berarti

Siklus hidup bug :

- Review (Diulas)
- Rejected (Ditolak)
- Open (Dibuka)
- Assigned (Ditugaskan)
- Test (diuji)
- Reopened (Dibuka kembali)
- Closed (ditutup)
- Deffered (ditangguhkan)

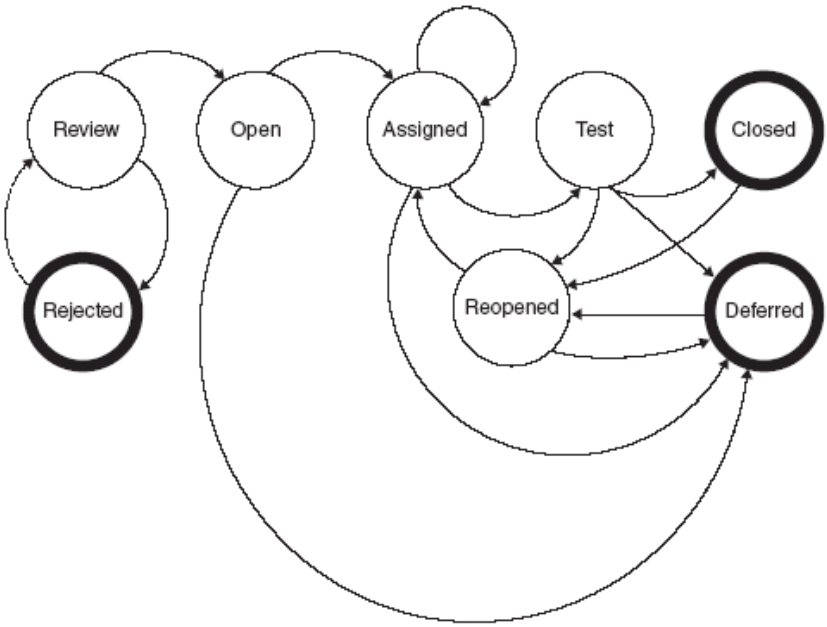


Figure 4.8 A bug report life cycle or workflow.

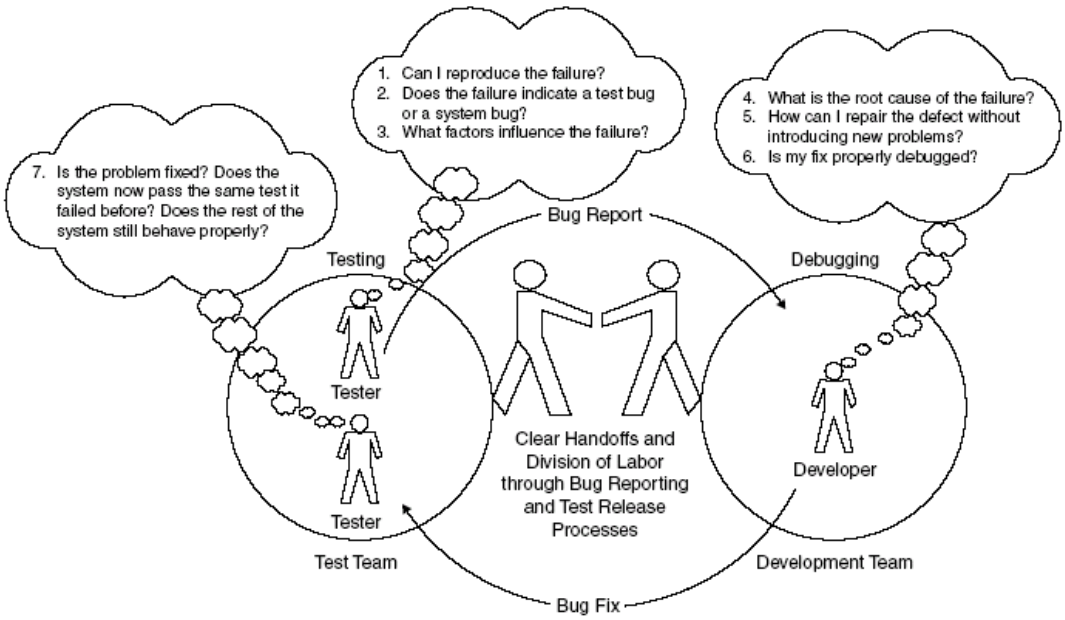


Figure 4.9 Questions, players, and handoffs in the bug life cycle.

PERTEMUAN VII

PENGUJIAN DAN IMPLEMENTASI SISTEM

ANALISA RISIKO BAHAYA DENGAN METODE FMEA

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 7 (Tujuh) Waktu :

Modul	7 (Tujuh)
Topik	*ANALISA RISIKO BAHAYA DENGAN METODE FMEA*
Sub Topik	-
Materi	• Latar Belakang • Aturan Dasar • Manfaat • Proses FMEA • Timing • Penggunaan • Keuntungan • Jenis
Tujuan	

A. LATAR BELAKANG

Analisis dan metode risiko bahaya seperti Mode Kegagalan dan Efek Analysis (FMEA), Fault TreeAnalisis (FTA), Event Tree Analysis (ETA), Cause- Konsekuensi Analisis (CCA), Hazard awal Analisis (prHA), Analisis Keandalan Manusia (HRA), dan Hazard dan Operabilit.

Failure Mode and Effects Analysis (FMEA) adalah salah satu metode analisa failure/potensi kegagalan yang diterapkan dalam pengembangan produk, system engineering dan manajemen operasional.

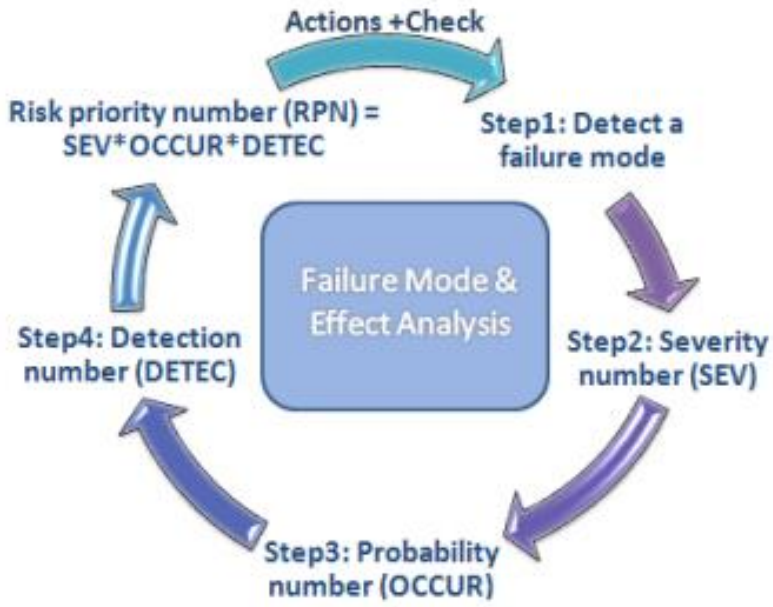
Kegiatan FMEA membantu untuk mengidentifikasi mode kegagalan potensial berdasarkan pengalaman dengan produk sejenis dan proses - atau berdasarkan fisika umum dari logika kegagalan.

Hal ini banyak digunakan dalam pengembangan dan manufaktur industri di berbagai tahapan siklus hidup produk.Analisis Efek mengacu mempelajari konsekuensi dari kegagalan-kegagalan pada tingkat sistem yang berbeda.

Analisis fungsional diperlukan sebagai masukan untuk menentukan mode kegagalan yang benar, di semua tingkat sistem, baik untuk FMEA fungsional atau Sepotong-Part (hardware). FMEA Sebuah FMEA digunakan untuk struktur Mitigasi untuk pengurangan risiko berdasarkan baik kegagalan (modus) pengurangan keparahan efek atau berdasarkan menurunkan probabilitas kegagalan atau keduanya.

FMEA adalah prinsip induktif penuh (logika maju) analisis, namun probabilitas kegagalan hanya dapat diperkirakan atau dikurangi dengan memahami mekanisme kegagalan. Idealnya probabilitas ini akan diturunkan menjadi "tidak mungkin terjadi" dengan menghilangkan (root) menyebabkan . Oleh karena itu penting untuk memasukkan dalam FMEA kedalaman informasi yang tepat tentang penyebab kegagalan (analisis deduktif).

FMEA dilakukan untuk menganalisa potensi kesalahan atau kegagalan dalam sistem atau proses, dan potensi yang teridentifikasi akan diklasifikasikan menurut besarnya potensi kegagalan dan efeknya terhadap
proses. Metode ini membantu tim proyek untuk mengidentifikasi potential failure mode yang berbasis kepada kejadian dan pengalaman yang telah lalu yang berkaitan dengan produk atau proses yang serupa. FMEA membuat tim mampu merancang proses yang bebas waste dan meminimalisir kesalahan serta kegagalan.



Awalnya, FMEA digunakan di industri manufaktur dalam siklus DMAIC dalam proyek Lean Manufacturing. Kini penggunaan tool Failure Mode and Effects Analysis telah meluas ke industri jasa (service). Secara umum, sebelum melakukan FMEA, tim perlu mengidentifikasi beberapa informasi mengenai :

1. Produk / barang / jasa
2. Fungsi
3. Efek dari kegagalan / kesalahan
4. Penyebab kesalahan
5. Kontrol yang dilakukan saat ini untuk mencegah kesalahan
6. Cara penanggulangan yang direkomendasikan
7. Detail-detail lain yang relevan.

B. ATURAN DASAR

Aturan dasar setiap FMEA termasuk satu set prosedur proyek yang dipilih; asumsi yang dianalisis ditetapkan; perangkat keras yang telah dimasukkan dan dikeluarkan dari analisis dan alasan untuk pengecualian. Aturan-aturan dasar juga menggambarkan tingkat indenture analisis, status hardware dasar, dan kriteria sistem dan keberhasilan misi.

Setiap upaya harus dilakukan untuk menentukan semua aturan-aturan dasar sebelum FMEA dimulai; Namun, aturan-aturan dasar dapat diperluas dan diklarifikasi sebagai hasil analisis. Satu set khas tanah aturan (asumsi) berikut :

1. Hanya satu modus kegagalan ada pada suatu waktu.
2. Semua input (termasuk perintah software) untuk item yang dianalisis hadir dan pada nilai nominal.
3. Semua tersedia dalam jumlah yang cukup.

4. Daya nominal tersedia.

C. MANFAAT

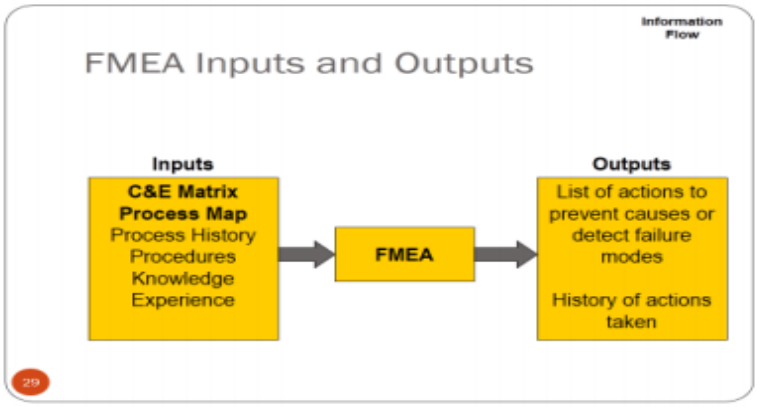
Manfaat utama dari dilaksanakan FMEA dengan benar adalah sebagai berikut:

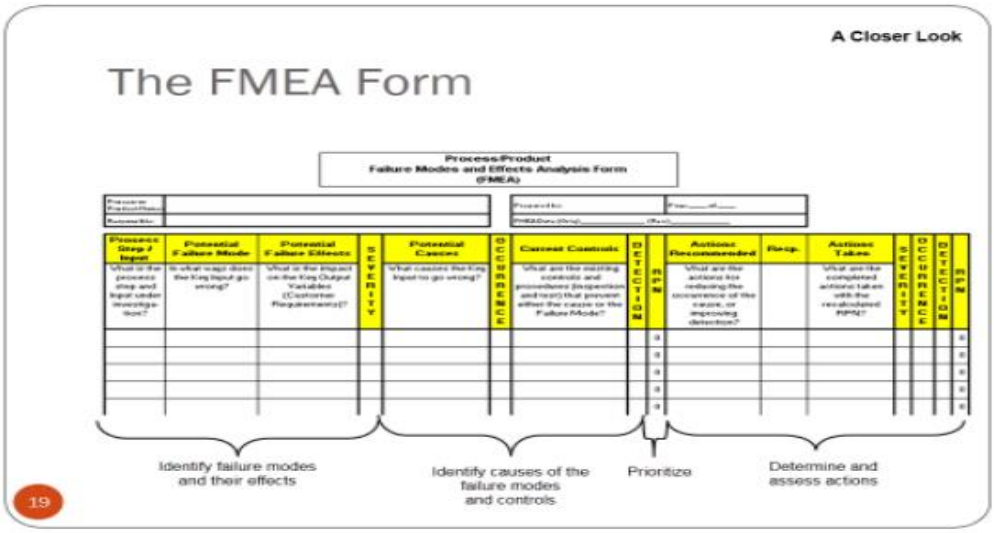
1. metode Ini menyediakan dokumentasi untuk memilih desain dengan probabilitas yang tinggi atas keberhasilan operasi dan keselamatan.
2. Sebuah metode yang seragam didokumentasikan menilai mekanisme potensi kegagalan, mode kegagalan dan dampaknya terhadap sistem operasi, sehingga daftar mode kegagalan peringkat menurut keseriusan dampak sistem mereka dan kemungkinan terjadinya.
3. Identifikasi awal dari titik tunggal kegagalan (SFP) dan masalah sistem antarmuka, yang mungkin penting untuk keberhasilan dan / atau keamanan misi. Ini juga memberikan verifikasi metode yang berubah antara elemen berlebihan yang tidak terancam dengan kegagalan tunggal.
4. Metode yang efektif untuk mengevaluasi dampak dari perubahan yang diusulkan untuk desain dan / atau prosedur operasional pada keberhasilan misi dan keselamatan.
5. Sebuah dasar untuk prosedur pemecahan masalah dalam penerbangan dan untuk mencari pemantauan kinerja dan kesalahan-deteksi perangkat.
6. Kriteria perencanaan awal tes.

D. PROSES FMEA

Langkah yang diperlukan dalam melakukan Failure Mode and Effects Analysis (FMEA) yaitu :

1. Kumpulkan seluruh anggota tim
2. Tetapkan aturan dasar
3. Kumpulkan informasi yang relevan dan lakukan review
4. Identifikasi item atau proses yang akan dianalisa
5. Identifikasi fungsi, kegagalan, efek, penyebab, dan kontrol dari setiap item atau proses yang dianalisa
6. Evaluasi resiko berkaitan dengan isu atau potensi yang teridentifikasi melalui analisa
7. Prioritaskan dan rumuskan aksi / solusi
8. Lakukan tindakan pembetulan dan evaluasi ulang resiko yang ada
9. Distribusikan, review dan update analisa sesuai kebutuhan.





Berikut adalah contoh dari FMEA disederhanakan untuk proses instalasi sabuk pengaman di sebuah pabrik perakitan mobil.

FAILURE MODE & EFFECTS ANALYSIS (FMEA)

Date: 1/1/2000
Revision: 1.3

Process Name: Left Front Seat Belt Install Process Number: SBT 445

Failure Mode	A) Severity Rate 1-10 10 = Most Severe	B) Probability of Occurance Rate 1-10 10 = Highest Probability	C) Probability of Detection Rate 1 - 10 10 = Lowest Probability	Risk Preference Number (RPN) AxBxC
1) Select Wrong Color Seat Belt	5	4	3	**60**
2) Seat Belt Bolt Not Fully Tightened	9	2	8	**144**
3) Trim Cover Clip Misaligned	2	3	4	**24**

Seperti yang Anda lihat, tiga mode kegagalan potensial telah diidentifikasi. Modus kegagalan nomor dua memiliki RPN dari 144, dan oleh karena itu merupakan prioritas tertinggi untuk perbaikan proses.

FMEA ini sering diselesaikan sebagai bagian dari proses peluncuran produk baru. Target minimal RPN dapat dibentuk untuk memastikan tingkat tertentu kemampuan proses sebelum pengiriman produk kepada pelanggan. Dalam acara itu, adalah bijaksana untuk menetapkan pedoman untuk menilai nilai-nilai untuk Severity, Kejadian, dan Deteksi membuat RPN seobjektif mungkin.

From Resource Engineering, Inc.

FMEA
Checklists and Forms

FMEA ANALYSIS WORKSHEET - PART 1

Because this form is so wide, we have split it over two pages. Click here to see the second half.

FMEA Analysis Worksheet

Process/Product:
FMEA Team:
Team Leader:

FMEA Process

Component & Function	Potential Failure Mode	Potential Effect(s) of Failure	Severity	Potential Cause(s) of Failure	Occurrence	Current Controls	Detection	RPN	Recomm'd Action	Respond. & Target Completion Date

Contoh PFMEA worksheet

FMEA Ref.	Barang	Potensi modus kegagalan	Potensi penyebab (s) / mekanisme	Misi Tahap	Efek lokal kegagalan	Berikutnya efek tingkat yang lebih tinggi	Sistem Tingkat Akhir Efek	(P) Probabilitas (perkiraan)	(S) Severity	(D) Detection (Indikasi untuk Operator, Pemeliharaan)	Deteksi Dormansi Periode	Tingkat risiko P * S (+ D)	Tindakan untuk lebih Investigasi / bukti	Mitigasi / Persyaratan
1..1	Pengembalian Pengurangan	Proses internal dari Menerima untuk Kembali Produk untuk Pelanggan	a) Pelanggan Panggilan TN b) Masalah yang dibahas dengan Wakil TN	Kepahan masalah	Jenis Aksi Memutuskan	Item Diterima kembali di TN	Proses Kembali Barang ke IFS	Toko Kembali di Lokasi yang tepat untuk Evaluasi.	Teknisi Mengevaluasi produk kembali dan tindak lanjut dengan solusi.	Tutup proses di IFS				

1. Probablitas (P)
   Hal ini diperlukan untuk melihat penyebab modus kegagalan dan kemungkinan terjadinya. Hal ini dapat dilakukan dengan analisis, perhitungan / FEM, melihat barang serupa atau proses dan modus kegagalan yang telah didokumentasikan untuk mereka di masa lalu.

   Penyebab kegagalan dipandang sebagai kelemahan desain. Semua penyebab potensi mode kegagalan harus diidentifikasi dan didokumentasikan. Ini harus dalam hal teknis.

   Contoh penyebab adalah: kesalahan manusia dalam penanganan, kesalahan Manufacturing diinduksi, Kelelahan, Creep, memakai Abrasive, algoritma yang salah, tegangan yang berlebihan atau tidak tepat kondisi operasi atau menggunakan (tergantung pada aturan-aturan dasar yang digunakan).

   Sebuah modus kegagalan diberi Ranking Probabilitas.

Penilaian	Makna
Sebuah	Sangat Tidak mungkin (Hampir tidak mungkin atau ada kejadian diketahui pada produk yang sama atau proses, dengan banyak jam berjalan)
B	Terpencil (relatif sedikit kegagalan)
C	Sesekali (kegagalan sesekali)
D	Cukup Kemungkinan (diulang kegagalan)
E	Sering (kegagalan hampir tak terelakkan)

2. Severity (S)

   Tentukan Severity untuk skenario terburuk efek akhir yang merugikan (negara). Hal ini mudah untuk menulis efek ini turun dalam hal apa pengguna mungkin melihat atau pengalaman dalam hal kegagalan fungsional.

   Contoh efek akhir ini adalah: hilangnya penuh fungsi x, kinerja terdegradasi, fungsi dalam modus terbalik, terlambat fungsi, fungsi menentu, dll Setiap efek akhir diberikan nomor Severity (S) dari, katakanlah, saya (tidak berpengaruh) untuk VI (bencana), berdasarkan biaya dan / atau hilangnya nyawa atau kualitas hidup. Angka-angka ini memprioritaskan mode kegagalan (bersama-sama dengan probabilitas dan pendeteksian).

Penilaian	Makna
Saya	Tidak ada efek yang relevan pada keandalan atau keselamatan
II	Sangat kecil, tidak ada kerusakan, tidak ada luka, hanya menghasilkan tindakan perawatan (hanya diketahui oleh pelanggan diskriminatif)
AKU AKU AKU	Kecil, kerusakan rendah, luka ringan (mempengaruhi sangat sedikit dari sistem, diketahui oleh pelanggan rata-rata)
IV	Moderat, rusak sedang, luka mungkin (sebagian besar konsumen terganggu, kerusakan sebagian besar keuangan)
V	Kritis (menyebabkan hilangnya fungsi utama; Kehilangan semua Margin keselamatan, 1 kegagalan jauh dari bencana, kerusakan parah, luka parah, max 1 kematian mungkin)
VI	Bencana (produk menjadi tdk berlaku; kegagalan dapat mengakibatkan pengoperasian yang tidak aman lengkap dan mungkin beberapa kematian)

3. Detecion (D)

   Sarana atau metode yang gagal terdeteksi, terisolasi oleh operator dan / atau pengelola dan waktu mungkin diperlukan. Hal ini penting untuk kontrol pemeliharaan (Ketersediaan sistem) dan itu adalah khusus penting untuk beberapa skenario kegagalan. Ini mungkin melibatkan mode kegagalan aktif (misalnya ada efek sistem langsung, sementara sistem berlebihan / item otomatis mengambil alih atau ketika kegagalan hanya bermasalah selama misi atau sistem negara tertentu) atau laten kegagalan (misalnya mekanisme kegagalan kerusakan, seperti tumbuh retak logam , tapi tidak panjang kritis). Ini harus dibuat jelas bagaimana modus kegagalan atau penyebab dapat ditemukan oleh operator pada operasi sistem normal atau jika dapat

ditemukan oleh kru pemeliharaan oleh beberapa tindakan diagnostik atau otomatis dibangun dalam tes sistem. A dormansi dan / atau masa laten dapat dimasukkan.

4. Dormansi atau Latency Period

   Rata-rata waktu yang mode kegagalan dapat terdeteksi dapat dimasukkan jika diketahui. Sebagai contoh:

   - ✓ Proses Durasi setelah menerima di port, menunggu waktu, X Jam atau X Days.
   - ✓ Proses Durasi untuk masuk ke IFS, X Jam, hari atau minggu.
   - ✓ Proses Menetapkan untuk Teknisi untuk evaluasi (misalnya 8 jam rata-rata).
   - ✓ Evaluasi teknis dilakukan dan produk Ulasan, jam atau hari.
   - ✓ Tentukan solusi untuk pelanggan.
   - ✓ Kembali produk kepada pelanggan.

5. Indikasi

   Jika kegagalan terdeteksi memungkinkan sistem untuk tetap di tempat yang aman / negara bekerja, situasi kegagalan kedua harus dieksplorasi untuk menentukan apakah atau tidak indikasi akan jelas ke semua operator dan apa tindakan korektif mereka mungkin atau harus mengambil.

   Indikasi ke operator harus dijelaskan sebagai berikut :

   - ✓ Normal. Sebuah indikasi yang jelas bagi operator ketika sistem atau peralatan beroperasi secara normal.
   - ✓ Abnormal. Sebuah indikasi yang jelas bagi operator ketika sistem telah berfungsi atau gagal.
   - ✓ Salah. Indikasi yang salah dengan operator karena kerusakan atau kegagalan indikator (yaitu, instrumen, perangkat penginderaan, perangkat peringatan visual atau terdengar, dll).

6. Tingkat Risiko (P * S) dan (D)

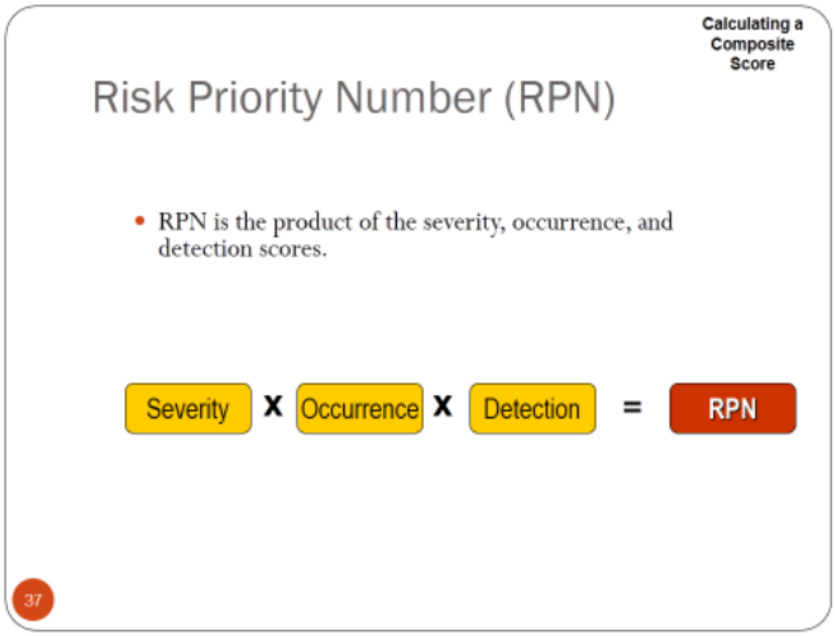
   Risiko adalah kombinasi End Effect Probabilitas Dan Severity. Di mana probabilitas dan tingkat keparahan termasuk efek pada non-pendeteksian (waktu dormansi). Hal ini dapat mempengaruhi kemungkinan efek akhir kegagalan atau terburuk efek kasus Severity.

   Perhitungan yang tepat mungkin tidak mudah dalam semua kasus, seperti yang di mana beberapa skenario (dengan beberapa peristiwa) yang mungkin dan pendeteksian / dormansi memainkan peran penting (seperti untuk sistem berlebihan).

   Dalam hal kesalahan Tree Analysis dan / atau Kegiatan Pohon mungkin diperlukan untuk menentukan probabilitas dan tingkat risiko yang tepat. Tingkat risiko awal dapat dipilih berdasarkan pada Matrix Risiko seperti yang ditunjukkan di bawah, berdasarkan Mil. Std. 882. [24] Semakin tinggi tingkat risiko, semakin pembenaran dan mitigasi yang diperlukan untuk memberikan bukti dan menurunkan risiko ke tingkat yang dapat diterima.

Berisiko tinggi harus ditunjukkan kepada manajemen tingkat yang lebih tinggi, yang bertanggung jawab untuk akhir pengambilan keputusan

Probabilitas / Severity ->	Saya	II	AKU AKU AKU	IV	V	VI
Sebuah	Rendah	Rendah	Rendah	Rendah	Moderat	Tinggi
B	Rendah	Rendah	Rendah	Moderat	Tinggi	Tidak dapat diterima
C	Rendah	Rendah	Moderat	Moderat	Tinggi	Tidak dapat diterima
D	Rendah	Moderat	Moderat	Tinggi	Tidak dapat diterima	Tidak dapat diterima
E	Moderat	Moderat	Tinggi	Tidak dapat diterima	Tidak dapat diterima	Tidak dapat diterima



E. TIMING

FMEA Harus diperbarui setiap kali :

1. Sebuah Siklus baru dimulai (Produk baru/proses).
2. Perubahan yang dibuat untuk kondisi operasi.
3. Sebuah perubahan dibuat dalam desain.
4. Peraturan baru yang dilembagakan.
5. Umpan balik pelanggan menunjukkan masalah.

F. PENGGUNAAN

Penggunaan dari metode FMEA adalah untuk :

1. Pengembangan persyaratan sistem yang meminimalkan kemungkinan kegagalan.
2. Pengembangan desain dan sistem pengujian untuk memastikan bahwa kegagalan telah dieliminasi atau
3. risiko berkurang untuk tingkat yang dapat diterima.
4. Pengembangan dan evaluasi sistem diagnostik
5. Untuk membantu dengan pilihan desain (trade-off analisis).

G. KEUNTUNGAN

Adapun keuntungan dari menggunakan metode FMEA adalah :

1. Meningkatkan kualitas, keandalan dan keamanan produk / proses.
2. Meningkatkan citra perusahaan dan daya saing.
3. Meningkatkan kepuasan pengguna.
4. Mengurangi waktu pengembangan sistem dan biaya.
5. Mengumpulkan informasi untuk mengurangi kegagalan masa depan, pengetahuan teknik capture.
6. Mengurangi potensi masalah garansi.
7. Identifikasi awal dan penghapusan mode kegagalan potensial.
8. Tekankan pencegahan masalah.
9. Meminimalkan perubahan akhir dan biaya yang terkait.
10. Katalis untuk kerja tim dan ide pertukaran antara fungsi.
11. Mengurangi kemungkinan jenis yang sama dari kegagalan di masa.
12. Mengurangi dampak pada perusahaan profit margin.
13. Meningkatkan hasil produksi.
14. Maximises laba.

H. JENIS

1. **Fungsional** : sebelum solusi desain yang disediakan (atau hanya pada tingkat tinggi) fungsi dapat dievaluasi efek kegagalan fungsional potensial. Umum Mitigasi ("desain untuk" persyaratan) dapat diusulkan untuk membatasi konsekuensi dari kegagalan fungsional atau membatasi kemungkinan terjadinya dalam pengembangan awal ini. Hal ini didasarkan pada gangguan fungsional dari sebuah sistem. Tipe ini juga dapat digunakan untuk evaluasi Software.
2. **Konsep Desain / Hardware** : analisis sistem atau subsistem dalam tahap konsep desain awal untuk menganalisis mekanisme kegagalan dan kegagalan fungsional tingkat yang lebih rendah, khususnya untuk solusi konsep yang berbeda secara lebih rinci. Ini dapat digunakan dalam perdagangan-off studi.
3. **Rinci Desain / Hardware** : analisis produk sebelum produksi. Ini adalah yang paling rinci (dalam mil 1629 disebut Sepotong-Bagian atau Hardware FMEA) meningkatkan FMEA dan digunakan untuk mengidentifikasi perangkat keras yang mungkin (atau lainnya) modus kegagalan sampai ke tingkat bagian terendah. Ini harus didasarkan pada hardware breakdown (misalnya BoM = Bill of Material). Efek Kegagalan

Severity, kegagalan Pencegahan (Mitigasi), Kegagalan Deteksi dan Diagnostik dapat sepenuhnya dianalisis dalam FMEA ini.

4. **Proses** : analisis manufaktur dan perakitan proses. Kualitas dan keandalan dapat dipengaruhi dari kesalahan proses. Input untuk FMEA ini antara lain proses kerja / tugas Breakdown.

PERTEMUAN VIII

PENGUJIAN DAN IMPLEMENTASI SISTEM

TEST TRACKING SPREADSHEET

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 8 (Delapan) Waktu :

Modul	8 (Delapan)
Topik	*Test Trackin Spreadsheet*
Sub Topik	-
Materi	• Test Tracking Spread Sheet • Test Tracking Spread Sederhana • A Test Summary Worksheet • Mengguakan Test Tracking Spreadshet pada pengujian Project. • Membuat perbaikan
Tujuan	Mahasiswa dapat menerangkan Test Tracking Spreadsheet

A. Test Tracking Spread Sheet

Test tracking spreadsheet merupakan suatu alat yang digunakan untuk mengelola eksekusi pengujian. Pada t*est Tracking Spreadsheet* ini dapat dimasukkan:

1. *Test case summary,* yang digunakan saat penguji ingin melacak status dari setiap *test case,* konfigurasi mana yang dijalankan saat pengujian, dan siapa yang menjalankan pengujian (Black, 2009:200).
2. *Test suite summary,* yang berisi data jumlah *test case* dalam setiap *suite* dan berapa *test case* dalam status (*Fail, Pass, Skip, in queue*, dan *Block*). (Black, 2009:203).

B. Test Tacking Spreadsheet sederhana

Paling sedikit memuat 9 data sebagai berikut:

1. Test Suite/Case
2. Status: Pass,Fail,Warning
3. System Configuration
4. Bug ID
5. Personnal In Charge (penanggung jawab)
6. Comment
7. Kolom T, bernilai 1 apabila merupakan test case.
8. Kolom F, bernilai 1 apabila test case fail.
9. Kolom P, bernilai 1 apabila test case pass

Membangun Test Tracking Spreadsheet sederhana terdapat 4 rangakaian pengujian yaitu :

- ✓ Environmental (Linkungan)
- ✓ Load, Capacity, and Volume
- ✓ Basic Functionality (Fungsi dasar)
- ✓ Standards

C. A Test Summary Worksheet

		System	Bug			Roll Up		
Test Suite/Case	**State**	**Config**	**ID**	**By**	**Comment**	**T**	**F**	**P**
Environmental								
Operating Thermal Profile	Pass	D		HS		1	0	1
Operating Temp/Humid Cycle	Pass	D		HS		1	0	1
Nonoperating Temp/Humid Cycle	Fail	D	017	HS		1	1	0
Nonoperating Drop	Pass	D		HS		1	0	1
Nonoperating Shock	Pass	D		HS		1	0	1
Nonoperating Thermal Shock	Pass	D		HS		1	0	1
Packaging Drop					Package not ready yet.	1	0	0
Packaging Shock					Package not ready yet.	1	0	0
Suite Summary						8	1	5
Load, Capacity, and Volume								
CPU and Memory	Pass	A,B,C		JR		1	0	1
FDD/Jaz/CD-ROM/DVD	Fail	A,B,C	020	JR		1	1	0
RAID	Pass	A,B,C		JR		1	0	1
Tape	Fail	A,B,C	020	JR		1	1	0
Network	Pass	A,B,C		JR		1	0	1
Modem Bank	Pass	A,B,C		JR		1	0	1
USB/Parallel/Serial	Pass	A,B,C		JR		1	0	1
Suite Summary						7	2	5
Basic Functionality								
Configure/Register NT						1	0	0
Configure/Register Solaris						1	0	0
Configure/Register Novell						1	0	0
FDD/Jaz/CD-ROM/DVD						1	0	0
RAID						1	0	0
Tape						1	0	0
Network-NT						1	0	0
Network-Novell						1	0	0
Network-PC-NFS						1	0	0
Modem Bank						1	0	0
USB/Parallel/Serial						1	0	0
UI (Video/Kbd/Mouse)						1	0	0
Suite Summary						12	0	0
Standards								
Solaris Logo						1	0	0
Windows NT Logo						1	0	0
Novell Logo						1	0	0
Suite Summary						3	0	0

D. Mengguakan Test Tracking Spreadshet pada pengujian Project

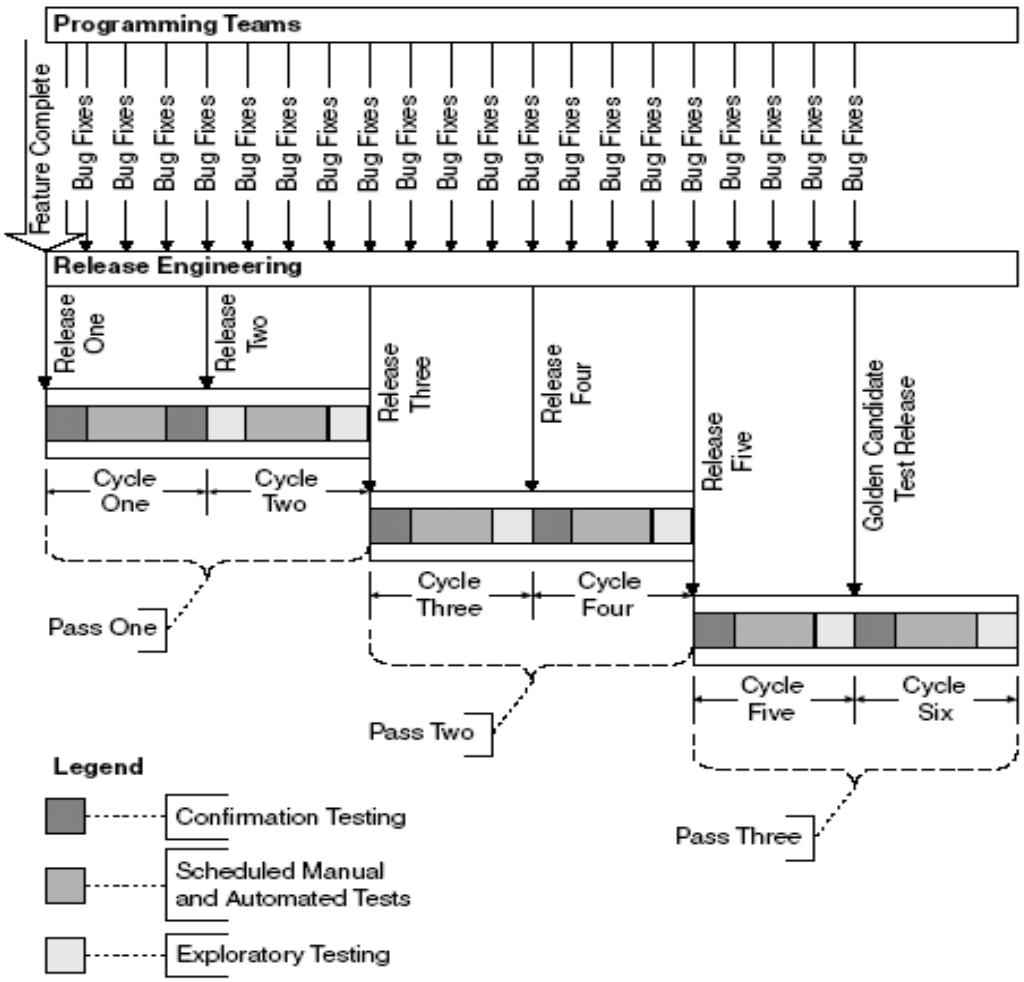


Figure 5.4 Test passes, test releases, and test cycles.

E. Membuat Perbaikan

- ✓ Menugaskan Pengidentifikasi dan Penguji untuk Menguji Rangkaian dan Kasus
- ✓ Menambah Tanggal dan Jam Informasi: Rencana vs Aktual
- ✓ Memahami Berapa Lama Pengujian Berjalan
- ✓ Meningkatkan Ketepatan Kasus Uji keadaan
- ✓ Memprioritaskan Test Suites and Cases
- ✓ Meneliti Kolom Roll Up
- ✓ Ringkaskan dan Data Grup
- ✓ Termasuk Detail Kasus Uji
- ✓ Cakupan Pelacakan

PERTEMUAN IX

PENGUJIAN DAN IMPLEMENTASI SISTEM

PENGELOLAAN PERUBAHAN PADA PENGUJIAN YANG SEDANG BERJALAN

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 9 (Sembilan) Waktu :

Modul	9 (Sembilan)
Topik	Pengelolaan Perubahan pada Pengujian yang sedang berjalan
Sub Topik	-
Materi	• Mengelola Perubahan pada proses Testing • Saran dalam menghadapi kegentingan pada proses pengujian • Alat Bantu • Basis Data Logistik • Simple Change Management Database
Tujuan	Mahasiswa dapat menerangkan pengelolaan Perubahan pada Pengujian yang Sedang Berjalan

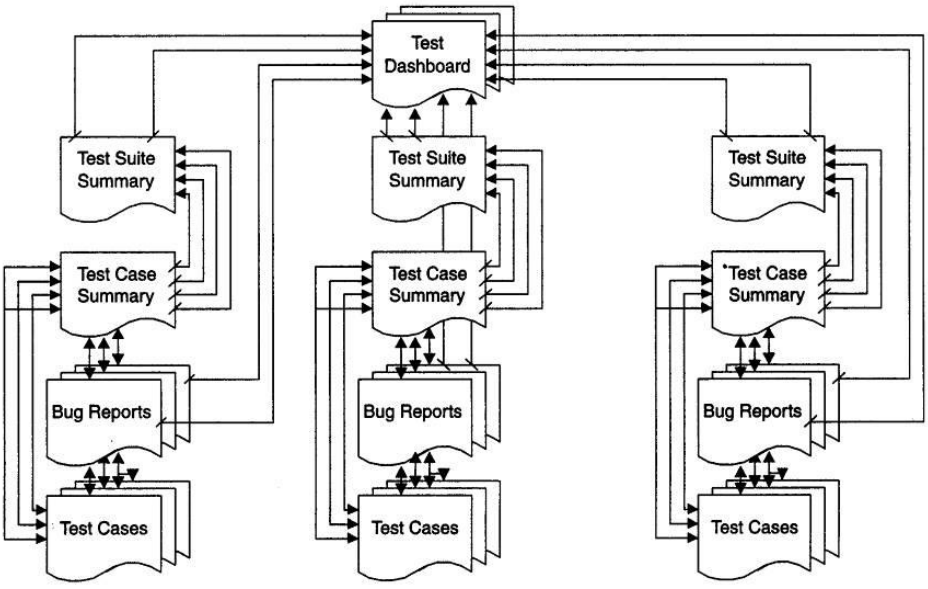
A. Mengelola Perubahan pada Proses Testing

Apabila kita sudah menangani proyek pengujian perangkat lunak satu atau dua kali maka kita akan mengalami masa-masa kegentingan yang sangat memeras tenaga dan membingungkan.

Masalah yang dihadapi: kenyataan yang dihadapi tidak sesuai yang diharapkan, perencanaan yang selalu berubah, dan ketergantungan yang tidak jelas.

B. Beberapa Saran dalam menghadapi kegentingan pada proses pengujian.

- ✓ Tetap maju sementara mendapatkan seluruh kenyataan: keinginan terhadap kepastian, kemajuan yang tetap harus terlaksana.
- ✓ Ketergantungan, Jadwal, dan peringatan: pentingnya untuk menindaklanjuti.
- ✓ Masalah Pengiriman: Proses Revisi dan pengeluaran (*release*).
- ✓ Masalah instalasi: Konfigurasi lingkungan pengujian.
- ✓ Pemeriksaan (*auditing*) dan memperbaharui (*update*) hasil pengujian.
- ✓ Mendefinisikan proses pelaksanaan pengujian.=> Mengaplikasikan ***Exploratory Testing.***
- ✓ Ketika pengujian gagal: meminimumkan kesalahan pada interpretasi hasil pengujian.
- ✓ Bila saat kegentingan, liburan dan tradisi kebudayaan bertubrukan.



Gambar Hubungan antara Test tracking dan Laporan

C. Alat Bantu

Disini akan dibahas 2 buah alat bantu baru untuk mengatasi kegentingan yang terjadi pada proses pengujian.

Pertama: Basis Data Logistik (***Logistics Database***) yang dapat menelusuri lokasi, konfigurasi, dan kebutuhan pengujian akan perangkat keras, perangkat lunak, infrastruktur, dan staf.

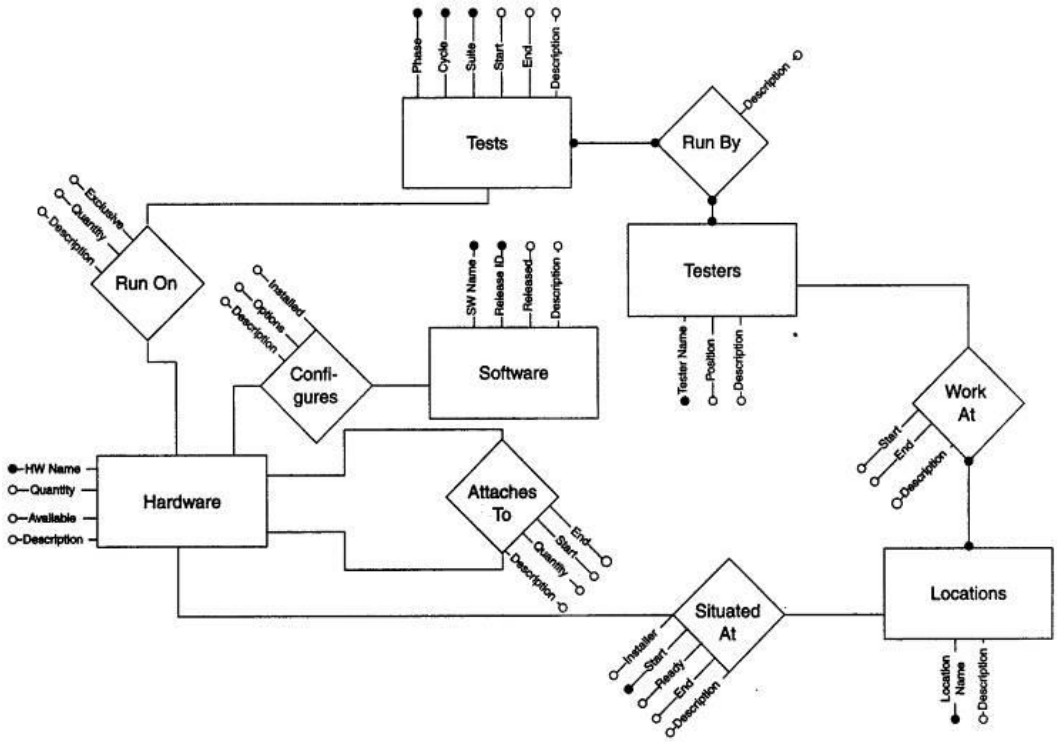
Kedua: Basis Data Pengelola Perubahan Sederhana (*Simple Change Management Database*) yang membantu dalam menanggapi bagian perbaikan yang tak terelakan yang muncul pada setiap proyek pengujian.

D. Basis Data logistik

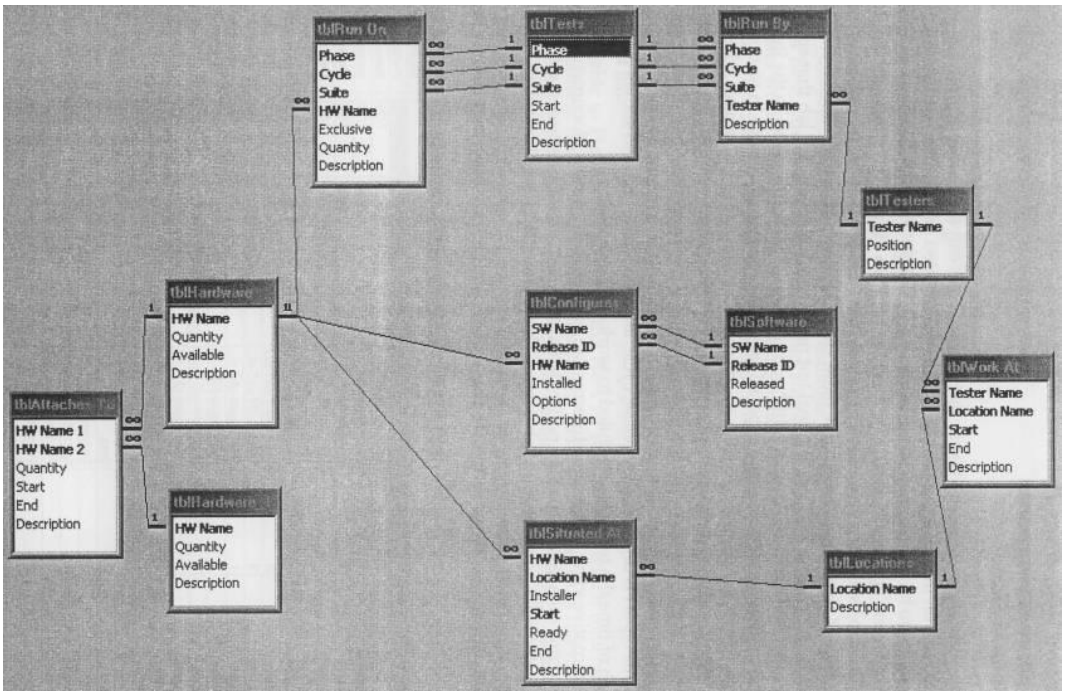
Mengelola logistik perangkat keras pengujian dan konfigurasi perangkat lunak.

Basis data ini dapat melakukan penelusuran terhadap :

- ✓ Lokasi, instalasi, dan perpindahan perangkat keras.
- ✓ Konfigurasi perangkat lunak pada saat ini, pada masa lalu, dan konfigurasi yang akan direncanakan.
- ✓ Interkoneksi perangkat keras dan jaringan computer.
- ✓ Lokasi pengujian.
- ✓ Infrastruktur pengujian
- ✓ Penugasan enggineer pengujian dan lokasinya.
- ✓ Penyebaran sumber daya manusia.



Gambar. E-R Diagram



Gambar. Penerapan Basis Data Logistik

Phase	Cycle	Suite	Start	End
Component	1	Edit Engine	7/19	7/21
Component	1	User Interface	7/22	7/23
Component	2	Edit Engine	7/26	7/28
Component	2	File	7/26	7/27
Component	2	Tools	7/28	7/28
Component	2	User Interface	7/29	7/30
Component	3	File	8/2	8/3
Component	3	Tools	8/4	8/4
Integration	1	Edit Engine-UI	8/2	8/4
Integration	2	Edit Engine-File	8/9	8/10
Integration	2	File-UI	8/11	8/12
Integration	3	Edit Engine-UI	8/16	8/18
System	1	Documentation/Packaging	8/16	8/20
System	1	Install/Configure	8/16	8/20
System	1	Performance	8/16	8/18
System	2	Beta	8/23	9/5
System	2	Compatibility	8/23	8/29
System	2	Error Handling/Recovery	8/23	8/24
System	2	File Sharing	8/25	8/27
System	3	Compatibility	8/30	9/5
System	3	Documentation/Packaging	8/30	9/3
System	3	Error Handling/Recovery	8/30	8/31
System	3	File Sharing	9/1	9/3
System	3	Install/Configure	8/30	9/3
System	3	Performance	8/30	9/1

Gambar. Contoh : Speedy Writer Test

	Tester Name	Position
⊞	John Goldstein	Technician
⊞	Lin-Tsu Woo	Engineer
⊞	Liz Campbell	Technician
⊞	Muhammad Zamanzadeh	Manager
⊞	Sales/Marketing	Sales, Marketing, and Users
⊞	STC	Test Lab

Gambar.Daftar Tester

Phase	Cycle	Suite	Tester Name
Component	1	Edit Engine	Lin-Tsu Woo
Component	1	User Interface	Lin-Tsu Woo
Component	2	Edit Engine	Lin-Tsu Woo
Component	2	File	Liz Campbell
Component	2	Tools	Liz Campbell
Component	2	User Interface	Lin-Tsu Woo
Component	3	File	Liz Campbell
Component	3	Tools	Liz Campbell
Integration	1	Edit Engine-UI	Lin-Tsu Woo
Integration	2	Edit Engine-File	Lin-Tsu Woo
Integration	2	File-UI	Lin-Tsu Woo
Integration	3	Edit Engine-UI	Lin-Tsu Woo
System	1	Documentation/Packaging	John Goldstein
System	1	Install/Configure	Lin-Tsu Woo
System	1	Install/Configure	Liz Campbell
System	1	Performance	Lin-Tsu Woo
System	2	Beta	Sales/Marketing
System	2	Compatibility	STC
System	2	Error Handling/Recovery	Lin-Tsu Woo
System	2	File Sharing	Lin-Tsu Woo
System	3	Compatibility	STC
System	3	Documentation/Packaging	John Goldstein
System	3	Error Handling/Recovery	Lin-Tsu Woo
System	3	File Sharing	Lin-Tsu Woo
System	3	Install/Configure	Lin-Tsu Woo
System	3	Install/Configure	Liz Campbell
System	3	Performance	Lin-Tsu Woo

Gambar. Daftar Penugasan Tester

Tester Name	Location Name	Start	End
John Goldstein	JG Home	1/1	8/6
John Goldstein	JG Home	8/21	9/3
John Goldstein	JG Home	9/6	12/31
John Goldstein	SC Engr	8/9	8/20
John Goldstein	SC SM	9/4	9/5
Lin-Tsu Woo	SC Engr	1/1	12/31
Liz Campbell	SC Engr	1/1	12/31
Muhammad Zamanzadeh	SC Engr	1/1	12/31
Sales/Marketing	SC SM	1/1	12/31
STC	STC Lab	1/1	12/31

Gambar. Daftar Lokasi Tester

Tester Assignments

TesterName	*Position*	*Phase*	*Cycle*	*Suite*	*Start*		*End*
John Goldstein	**Technician**						
		System Test					
			1	Documentation/Packaging	8/16		8/20
			3	Documentation/Packaging	8/30		9/3
		John Goldstein works on this project from			8/16	to	9/3
Lin-Tsu Woo	**Engineer**						
		Component Test					
			1	Edit Engine	7/19		7/21
			1	User Interface	7/22		7/23
			2	Edit Engine	7/26		7/28
			2	User Interface	7/29		7/30
		Integration Test					
			1	Edit Engine-UI	8/2		8/4
			2	Edit Engine-File	8/9		8/10
			2	File-UI	8/11		8/12
			3	Edit Engine-UI	8/16		8/18
		System Test					
			1	Performance	8/16		8/18
			1	Install/Configure	8/16		8/20
			2	Error Handling/Recovery	8/23		8/24
			2	File Sharing	8/23		8/27
			3	Error Handling/Recovery	8/30		8/31
			3	Performance	8/30		9/1
			3	Install/Configure	8/30		9/3
			3	File Sharing	9/1		9/3
		Lin-Tsu Woo works on this project from			7/19	to	9/3
Liz Campbell	**Technician**						
		Component Test					
			2	File	7/26		7/27
			2	Tools	7/28		7/28
			3	File	8/2		8/3
			3	Tools	8/4		8/4
		System Test					
			1	Install/Configure	8/16		8/20
			3	Install/Configure	8/30		9/3
		Liz Campbell works on this project from			7/26	to	9/3

Saturday, December 22, 2001 *Page 1 of 2*

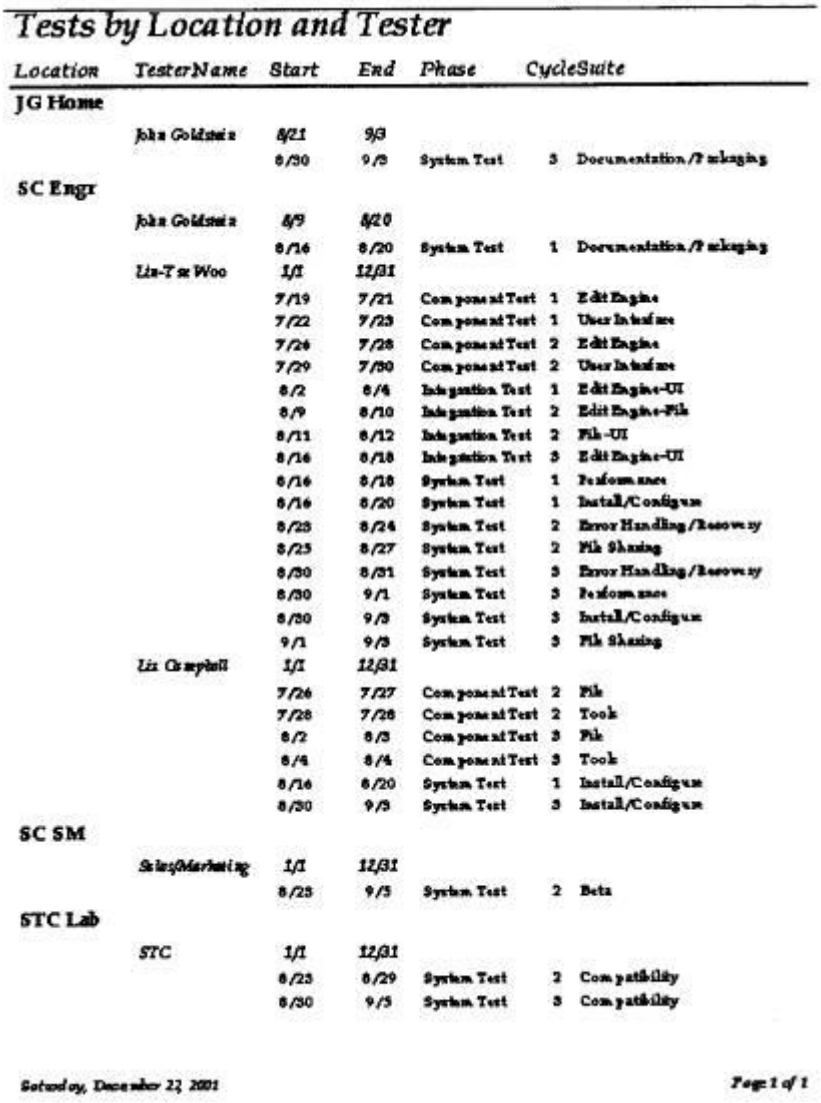
Gambar. Laporan Penugasan Tester

Test Schedule

Phase	*Cycle*	*Suite*	*TesterName*	*Start*		*End*
System Test						
	1					
		Documentation/Packaging		*8/16*		*8/20*
			John Goldstein			
		Install/Configure		*8/16*		*8/20*
			Lin-Tzu Woo			
			Lis Campbell			
		Performance		*8/16*		*8/18*
			Lin-Tzu Woo			
			System Test, Cycle 1 runs from	8/16	to	8/20
	2					
		Sales		*8/23*		*9/5*
			Sales/Marketing			
		Compatibility		*8/23*		*8/29*
			STC			
		Error Handling/Recovery		*8/23*		*8/26*
			Lin-Tzu Woo			
		File Sharing		*8/25*		*8/27*
			Lin-Tzu Woo			
			System Test, Cycle 2 runs from	8/23	to	9/5
	3					
		Compatibility		*8/30*		*9/5*
			STC			
		Documentation/Packaging		*8/30*		*9/3*
			John Goldstein			
		Error Handling/Recovery		*8/30*		*8/31*
			Lin-Tzu Woo			
		File Sharing		*9/1*		*9/3*
			Lin-Tzu Woo			
		Install/Configure		*8/30*		*9/3*
			Lin-Tzu Woo			
			Lis Campbell			
		Performance		*8/30*		*9/1*
			Lin-Tzu Woo			
			System Test, Cycle 3 runs from	8/30	to	9/5
			System Test runs from	8/16	to	9/5

Saturday, December 22, 2001 *Page 3 of 3*

Gambar. Laporan Jadwal Pengujian

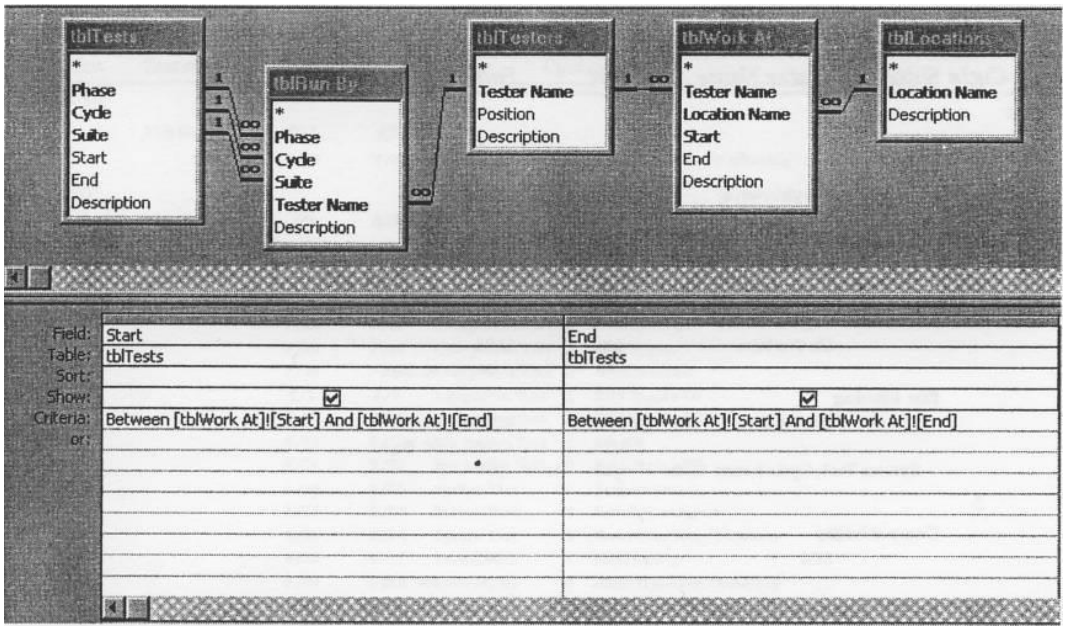


Tests by Location and Tester

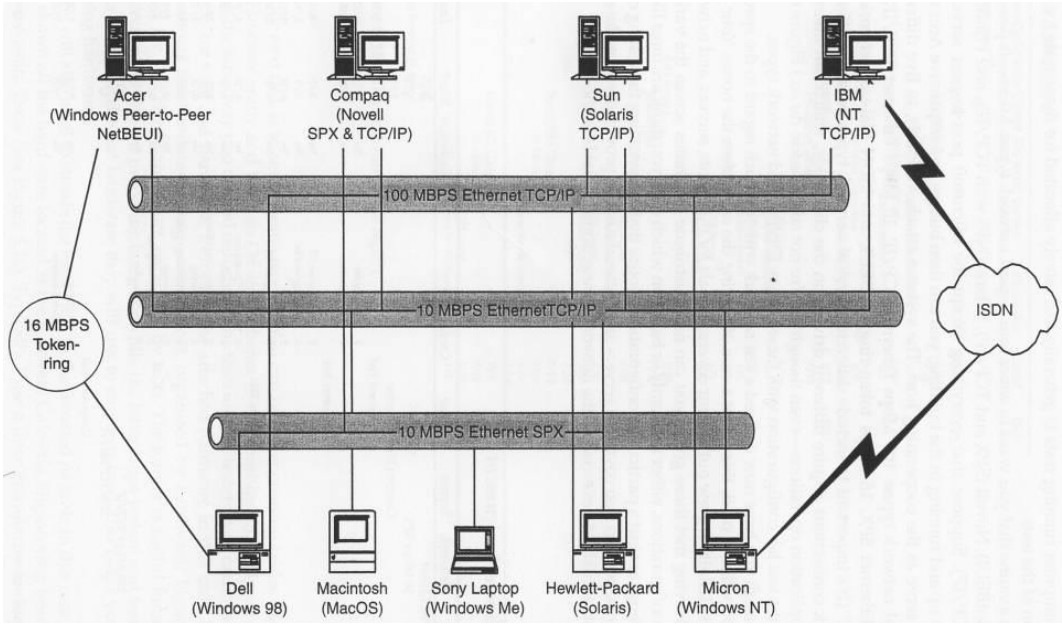
Location	TesterName	Start	End	Phase	CycleSuite	
JG Home						
	John Goldstein	8/21	9/3			
		8/30	9/3	System Test	3	Documentation/Packaging
SC Engr						
	John Goldstein	8/9	8/20			
		8/16	8/20	System Test	1	Documentation/Packaging
	Liz-Tze Woo	1/1	12/31			
		7/19	7/21	Component Test	1	Edit Engine
		7/22	7/23	Component Test	1	User Interface
		7/26	7/28	Component Test	2	Edit Engine
		7/29	7/30	Component Test	2	User Interface
		8/2	8/4	Integration Test	1	Edit Engine-UI
		8/9	8/10	Integration Test	2	Edit Engine-File
		8/11	8/12	Integration Test	2	File-UI
		8/16	8/18	Integration Test	3	Edit Engine-UI
		8/16	8/18	System Test	1	Performance
		8/16	8/20	System Test	1	Install/Configure
		8/23	8/24	System Test	2	Error Handling/Recovery
		8/25	8/27	System Test	2	File Sharing
		8/30	8/31	System Test	3	Error Handling/Recovery
		8/30	9/1	System Test	3	Performance
		8/30	9/3	System Test	3	Install/Configure
		9/1	9/3	System Test	3	File Sharing
	Liz Campbell	1/1	12/31			
		7/26	7/27	Component Test	2	File
		7/28	7/28	Component Test	2	Tools
		8/2	8/3	Component Test	3	File
		8/4	8/4	Component Test	3	Tools
		8/16	8/20	System Test	1	Install/Configure
		8/30	9/3	System Test	3	Install/Configure
SC SM						
	Sales/Marketing	1/1	12/31			
		8/23	9/3	System Test	2	Beta
STC Lab						
	STC	1/1	12/31			
		8/23	8/29	System Test	2	Compatibility
		8/30	9/3	System Test	3	Compatibility

Saturday, December 22, 2001 — Page 1 of 1

Gambar. Laporan Lokasi Pengujian dan Tester



Gambar. Form Query untuk mencari lokasi, tests, dan tester



Gambar. Test Hardware dan Jaringan

Hardware Assignment

Client	*Network*	*Server*	*Phase*	*Cycle*	*Suite*	*Exclusive*	*Start*	*End*
Dell W98 Client							7/15	
	10 Mbps SPX						7/19	
		Compaq Novell Server					1/1	
			Component Test					
				1	Edit Engine	Yes	7/19	7/21
				2	User Interface	Yes	7/29	7/30
			Integration Test					
				1	Edit Engine-UI	Yes	8/2	8/4
				2	Edit Engine-File	Yes	8/9	8/10
				2	File-UI	Yes	8/11	8/12
				3	Edit Engine-UI	Yes	8/16	8/18
			System Test					
				1	Install/Configure	Yes	8/16	8/20
				1	Performance	Yes	8/16	8/18
				2	Error Handling/Recovery	Yes	8/23	8/24
				2	File Sharing	Yes	8/25	8/27
				3	Error Handling/Recovery	No	8/30	8/31
				3	File Sharing	No	9/1	9/3
				3	Install/Configure	No	8/30	9/3
				3	Performance	No	8/30	9/1
	100 Mbps TCP/IP						7/19	
		IBM NT Server					1/1	
			Component Test					
				1	Edit Engine	Yes	7/19	7/21
				2	User Interface	Yes	7/29	7/30

Sunday, December 23, 2001 *Page 1 of 14*

Gambar. Penggunaan Hardware

Test Hardware

Phase Cycle Suite Client Network Server	Exclusive	Start	End
Component Test			
1			
Edit Engine		7/19	7/21
Dell W98 Client	Yes	7/15	
10 Mbps SPX		7/19	
Compaq Novell Server		1/1	
100 Mbps TCP/IP		7/19	
IBM NT Server		1/1	
Sun Solaris Server		1/1	
16 Mbps Token-Ring		8/2	
Acer MS Server		8/13	
User Interface		7/22	7/23
HP Solaris Client	Yes	7/14	
10 Mbps TCP/IP		7/19	
Acer MS Server		8/13	
Compaq Novell Server		1/1	
IBM NT Server		1/1	
Sun Solaris Server		1/1	
100 Mbps TCP/IP		7/19	
IBM NT Server		1/1	
Sun Solaris Server		1/1	
2			
Edit Engine		7/26	7/28
Mac Client	Yes	7/12	
10 Mbps TCP/IP		7/19	
Acer MS Server		8/13	
Compaq Novell Server		1/1	
IBM NT Server		1/1	
Sun Solaris Server		1/1	
File		7/26	7/27
Sony WMe Client	Yes	7/19	
10 Mbps SPX		7/19	
Compaq Novell Server		1/1	
Tools		7/28	7/28
Micron NT Client	Yes	7/6	
10 Mbps TCP/IP		7/19	
Acer MS Server		8/13	
Compaq Novell Server		1/1	

Gambar. Pengujian Hardware

Table 6.1 Planned Releases for Target Platforms

REVISION IDENTIFIERS	RELEASE DATE	PHASE	CYCLE
C.1.Mac C.1.W95 C.1.W98 C.1.WNT C.1.Sol	7/19	Component	1
C.2.Mac C.2.W95 C.2.W98 C.2.WNT C.2.Sol	7/26	Component	2
I.1.Mac I.1.W95 I.1.W98 I.1.WNT I.1.Sol	8/2	Component Integration	3 1
I.2.Mac I.2.W95 I.2.W98 I.2.WNT I.2.Sol	8/9	Integration	2
S.1.Mac S.1.W95 S.1.W98 S.1.WNT S.1.Sol	8/16	Integration System	3 1
S.2.Mac S.2.W95 S.2.W98 S.2.WNT S.2.Sol	8/23	System	2
S.3.Mac S.3.W95 S.3.W98 S.3.WNT S.3.Sol	8/30	System	3

Gambar. Planed Releases for target platforms

SW Name	Release ID	Released
SpeedyWriter	C.1.Mac	7/19
SpeedyWriter	C.1.Sol	7/20
SpeedyWriter	C.1.W95	7/19
SpeedyWriter	C.1.W98	7/19
SpeedyWriter	C.1.WNT	7/19
SpeedyWriter	C.2.Mac	7/26
SpeedyWriter	C.2.Sol	7/30
SpeedyWriter	C.2.WNT	7/28
SpeedyWriter	I.1.Mac	8/2
SpeedyWriter	I.1.W95	8/3
SpeedyWriter	I.1.W98	8/1
SpeedyWriter	I.1.WNT	8/1
SpeedyWriter	I.2.Sol	8/9
SpeedyWriter	I.2.W95	8/10
SpeedyWriter	I.2.W98	8/11
SpeedyWriter	I.2.WNT	8/8
SpeedyWriter	S.1.Mac	8/16
SpeedyWriter	S.1.Sol	8/17
SpeedyWriter	S.1.W95	8/17
SpeedyWriter	S.1.WNT	8/17
SpeedyWriter	S.2.Mac	8/23
SpeedyWriter	S.2.Sol	8/23
SpeedyWriter	S.2.W95	8/25
SpeedyWriter	S.2.W98	8/26
SpeedyWriter	S.3.Mac	8/30
SpeedyWriter	S.3.W95	8/31
SpeedyWriter	S.3.W98	9/1
SpeedyWriter	S.3.WNT	8/30

Tested Configurations

Phase	*Cycle*	*Suite*	*Client*	*Software*	*Release ID*	*Released*	*Installed*
Component Test							
	1						
		Edit Engine: Starts on 7/19 and ends on 7/21					
			Dell W98 Client (Available 7/15)				
				SpeedyWriter	C.1.W98	7/19	7/19
		User Interface: Starts on 7/22 and ends on 7/23					
			HP Solaris Client (Available 7/15)				
				SpeedyWriter	C.1.Sol	7/20	7/21
	2						
		Edit Engine: Starts on 7/26 and ends on 7/28					
			Mac Client (Available 7/12)				
				SpeedyWriter	C.2.Mac	7/26	7/26
		File: Starts on 7/26 and ends on 7/27					
			Sony WMe Client (Available 7/19)				
				SpeedyWriter	C.1.WMe	7/19	7/19
		Tools: Starts on 7/28 and ends on 7/28					
			Micron NT Client (Available 7/6)				
				SpeedyWriter	C.1.WNT	7/19	7/19
		User Interface: Starts on 7/29 and ends on 7/30					
			Dell W98 Client (Available 7/15)				
				SpeedyWriter	C.1.W98	7/19	7/19
	3						
		File: Starts on 8/2 and ends on 8/3					
			HP Solaris Client (Available 7/15)				
				SpeedyWriter	C.2.Sol	7/30	7/30
		Tools: Starts on 8/4 and ends on 8/4					
			Mac Client (Available 7/12)				
				SpeedyWriter	I.1.Mac	8/2	8/2
Integration Test							
	1						
		Edit Engine-UI: Starts on 8/2 and ends on 8/4					

Gambar. Tested Configration

E. Basis Data Pengelola Perubahan Sederhana (Simple Change Management Database)

Bagaimanapun sempurnanya suatu perencanaan, langkah penanganan yang hati-hati, kefektifan kerjasama dengan tim penguji, rekan kerja, dan manajer tetap saja akan menghadapi tanggapan yang tidak diharapkan.

Perubahan tidak dapat dihindari dan tidak seluruh perubahan dapat direncanakan sebelumnya.

Perubahan seringkali merupakan manifestasi dari proses pembelajaran.

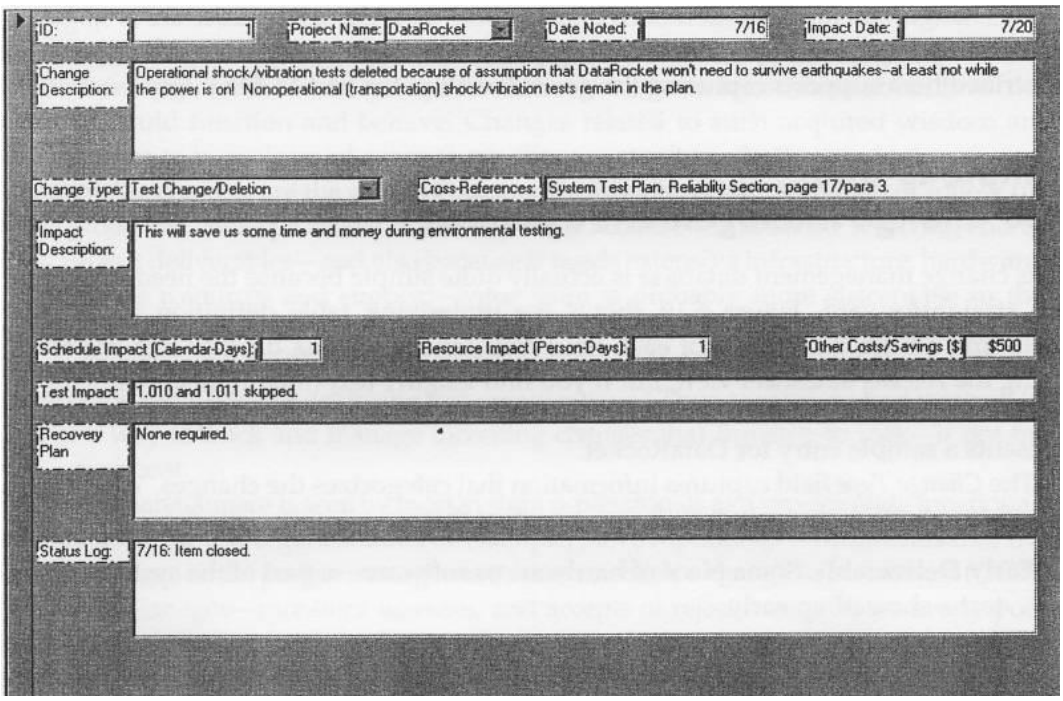
Pengelolaan perubahan dapat dilakukan secara formal dengan **CCB** (*change control board*) atau secara informal dengan *simple change management database.*

Menggunakan Data Pengelolan Perubahan.
Dengan mengumpulkan data mengenai perubahan, pengaruhnya dan perencanaan pemulihan (recovery plan) akan dapat memenuhi beberapa komunikasi yang penting dan tanggung jawab perencanaan.

Dengan data ini maka komunikasi dengan manajemen proyek dapat dilakukan secara terstruktur.

Field Name	Data Type	Description
ID	AutoNumber	Unique identifier for the change.
Project Name	Text	Name of project on which the change occurred.
Date Noted	Date/Time	Date on which the change was first noted.
Impact Date	Date/Time	Date on which the change will first affect testing.
Change Description	Memo	Short summary of what changed.
Change Type	Text	Type of change that is affecting testing.
Cross-References	Memo	List of the specific test documents that are affected by the change, with page and paragraph numbers.
Impact Description	Memo	Short description of what this change will mean for testing.
Schedule Impact	Number	Number of calendar-days delay or advance as a result of this change.
Resource Impact	Number	Number of person-days lost or saved as a result of this change.
Other Costs/Savings	Currency	Other costs/savings associated with this change.
Test Impact	Memo	Short list of the test case IDs affected by this change.
Recovery Plan	Memo	Short description of how we intend to accommodate this change, including any contingency plans identified in the test or project plan.
Status Log	Memo	Log of progress toward recovering from this problem.

Gambar. Tabel Sederhana untuk mencatat perubahan dalam pengujian



Gambar. Tabel Sederhana untuk mencatat perubahan dalam pengujian

PERTEMUAN X

PENGUJIAN DAN IMPLEMENTASI SISTEM

PENGELOLAAN LABORATORIUM PENGUJIAN

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 10 (Sepuluh)	Waktu :

Modul	10 (Sepuluh)
Topik	PENGELOLAAN LABORATORIUM PENGUJIAN
Sub Topik	-
Materi	• Test Laboratory • Kebutuhan Akan Laboratorium Pengujian • Memilih dan Merencanakan Laboratorium Pengujian • Persediaan dalam Laboratorium Pengujian • Masalah Keamanan dan Penelusuran • Pengelolaan Peralatan dan Konfigurasi • Menajga Kebersihan Lingkungan Laboratorium • Faktor Manusia • Keamanan dalam Laboratorium • Kerusakan Peralatan Laboratorium • Produktivitas dalam Laboratorium Pengujian
Tujuan	Mahasiswa dapat menerangkan pengelolaan laboratorium pengujian

A. Test Laboratory

Laboratorium pengujian adalah tempat dimana suatu proses pengujian dilakukan.

Penggunaan kata "**laboratorium**" disini untuk menekankan bahwa pelaku pengujian harus mengendalikan seluruh proses percobaan pengujian dan melakukan pengukuran serta mencatat hasilnya.

Laboratorium pengujian merupakan laboratorium rekayasa teknik (*engineering laboratory*), bukan merupakan laboratorium penelitian.

Laboratorium pengujian yang baik harus terstruktur dan diorganisasikan secara rapi.

Sebuah laboratorium pengujian merupakan suatu lokasi yang ada secara fisik.

Apabila tersedia lebih dari satu lokasi laboratorium pengujian, maka akan sangat berguna bila tiap lokasi laboratorium tersebut diberi nama.

Bahkan, apabila pengujian dilakukan secara terdistribusi maka dimungkinkan untuk digunakan beberapa laboratorium yang lokasinya tersebar di berbagai negara.

Dalam topik ini akan dibahas proses perencanaan, pembangunan, dan pengoperasian laboratorium pengujian.

B. Kebutuhan Akan Laboratorium Pengujian

Tidak semua organisasi pengujian membutuhkan laboratorium pengujian. Beberapa organisasi pengujian membutuhkan sebuah laboratorium hanya pada waktu tertentu saja; sedangkan ada pula organisasi yang tidak dapat melakukan pengujian tanpa tersedianya laboratorium.

Tetapi oleh karena pengadaan dan perawatan sebuah laboratorium pengujian memerlukan biaya yang mahal, maka dalam mengevaluasi akan kebutuhan suatu laboratorium pengujian harus dilakukan secara cermat.

Berikut ini adlh pertanyaan2 yg dpt menolong utk menentukan apkh memang benar2 diperlukan keberadaan laboratorium pengujian dlm suatu organisasi:

1. Apakah dalam pengujian diperlukan alat uji yang besar seperti bilik khusus (*chamber*)? Apakah alat uji yang digunakan tidak mudah untuk dipindah-pindahkan dan membutuhkan lokasi permanen? Contoh: Osiloskop.

2. Apakah lingkungan dengan kondisi khusus dibutuhkan ? Jika test platform memerlukan kondisi lingkungan yang ketat (seperti untuk server) atau kebutuhan akan tegangan yang tidak lazim (seperti sentral telepon), maka diperlukan paling tidak sebuah ruang server, jika tidak sebuah lab pengujian yang terpisah.

3. Apakah keamanan menjadi masalah? Sebagai contoh, ketika dilakukan pengujian perangkat keras dan perangkat lunak baru dan kerahasiaan terhadap pesaing harus dijaga. Sesuatu yang rahasia akan tetap menjadi rahasia hanya jika orang-orang tertentu saja yang mengehatui hal itu. Lebih jauh lagi, jika dilakukan pengujian kompatibilitas dengan berbagaimacam paket perangkat lunak maka CD-ROM dan disk adalah merupakan barang berharga, seperti juga CPU, tape drive, hard drive, dan sebangsanya. Menjaga akses yang terbatas terhadap barang-barang tersebut (khususnya dalam lemari yang terkunci) akan menghemat ribuan dollar terhadap kejadian kehilangan setiap tahunnya.

4. Apakah dibutuhkan pencegahan agar pihak yang bukan pelaku pengujian tidak dapat mengakses fasilitas dalam lingkungan pengujian? Kadang-kadang terjadi seseorang dari tim lain tidak dapat melakukan sesuatu karena tidak tersedianya komputer yang dapat digunakan sehingga ia mencoba menggunakan perangkat keras yang sesunguhnya sedang digunakan untuk proses testing. Dan ia lupa untuk mengembalikannya ke keadaan semula yang kemudian dapat mengacaukan proses tetsing yang dijalankan. Apabila dalam lingkungan kerja terdapat beberapa orang seperti ini maka dibutuhkan sebuah test lab yang formal.

5. Apakah diperlukan fasilitas pengujian dalam jangka waktu yang lama? Dapatkah beberapa proyek sekaligus menanggung biaya operasianal laboratorium sehingga akan menekan beban biaya bagi organisasi secara keseluruhan? Dapatkan laboratorium pengujian dibuat sebagai *profit center* (penghasil keuntungan) dengan menjual alat bantu dan jasa pengujian ke pihak luar?

C. Memilih dan Merencanakan Laboratorium Pengujian.

1. **Ukuran luas (*Size*)**

   Apakah kemungkinan lokasi laboratorium cukup luas untuk memenuhi kebutuhan sebuah *test lab*?

   Bila bentuk ruangan tidak persegi panjang / bujur sangkar dan cukup luas, maka ukuran harus diperhatikan hal-hal yang berhubungan dengan tata letak – harus diperhatikan kebutuhan ruang kerja sesungguhnya.

   Juga harus diperhatikan tingginya langit-langit, karena mungkin ada rak peralatan yang cukup tinggi.

   Selain itu harus diperhatikan pula masalah pintu. Segala sesuatu yang ada di dalam lab harus dapat masuk dan keluar dengan mudah, sehingga lokasi pintu, ukuran, dan arah buka/tutup daun pintu harus diperhatikan.

2. **Pencahayaan (*Lighting*)**

   Jendela dapat memberikan kesan estetika dan kenyamanan bagi orang yang berada dalam ruangan. Tetapi kadang2 jendela juga menjadi masalah saat sinar matahari menembus masuk langsung dari jendela. Sinar matahari ini selain menimbulkan panas, sinar ini akan mengakibatkan tampilan pada layar monitor sulit untuk dilihat dan panas yang ditimbulkannya dapat merusak berbagai peralatan.

   Seluruh jendela dalam lab harus dibuat agak gelap dan dilengkapi dengan tirai. Juga harus dipertimbangkan ketinggian dari jendela. Apabila jendela berada pada lantai dasar atau dengan dengan lantai dasar maka akan dapat menimbulkan masalah keamanan.

   Apabila digunakan penerangan buatan, maka lampu neon paling umum digunakan dan merupakan yang terbaik. Lampu pijar hanya dapat menerangi area yang terbatas, walaupun lampu pijar tidak menimbulkan masalah sakit kepala atau gangguan pada mata. Lokasi penempatan lampu dan jendela harus diperhatikan untuk memberikan kenyamanan dalam bekerja (mambaca dokumen dan melihat tampilan monitor).

3. **Tata letak (*Layout*)**

   Ketika kita memilih sebuah ruangan lab, harus diingat jenis meja dan rak peralatan yang akan digunakan. Ruangan berbentuk persegi sangat tepat bila seluruh peralatan diletakan dalam rak atau bila peralatan dapat dipindah-pindahkan secara bebas. Tetapi apabila lab akan menggunakan meja maka ruangan luas berbentuk persegi panjang akan lebih tepat.

Dalam ruangan harus disediakan area bebas untuk memudakan pekerja bergerak secara bebas dalam ruangan.

4. **Climate Control**
   Lab harus memiliki penyejuk udara yang cukup atau pemanas ruangan (apabila berada di lokasi dengan 4 musim) untuk memberikan suhu dan kelembaban udara yang stabil bagi peralatan dan pekerja.

   Pengaturan kondisi udara ini harus aktif selama 24 jam.

   Lebih baik lagi apabila dilengkap dengan sensor suhu sehingga bila suhu atau kelembaban udara berada diluar batas toleransi yang diijinkan maka peralatan akan mati secara otomatis.

   Sensor suhu dan kelembaban udara harus ditempatkan pada lokasi yang sesuai.

5. **Fire Safety and prevention**
   Setiap tempat kerja harus dilengkapi dengan detektor asap atau api, baik yang dicatu dengan batere maupun yang dicatu dengan listrik tetapi di-*back up* dengan batere. Detektor ini harus diuji fungsi secara teratur.

   Untuk mencegah kebakaran maka lab harus memiliki pemadam kebakaran portabel yang aman terhadap listrik dan efektif.

   Jika di dalam lab terdapat peralatan yang besar seperti komputer mainframe atau sentral peralatan komunikasi, maka diperlukan peralatan pengendalian kebakaran yang canggih yang aman baik bagi peralatan maupun karyawan.

   Peralatan pendeteksi dan penanggulangan kebakaran ini harus mudah diakses.

6. **Listrik (Power)**
   Bebagai jenis sumber daya listrik harus disiapkan dalam lab. Contoh: 120 VAC, 240 VAC, 480 VAC, 48 VDC, dan sebagainya.

   Sumber tegangan listrik ini harus tersedia di tempat yang tepat.

   Sebagai tambahan, daya listrik yang masuk harus stabil, sesuai dengan spesifikasi sesuai standar dan tidak boleh terputus, bebas dari *spikes*, *surges*, *sags*, *brownouts*, dan *blackouts* yang mengganggu jaringan listrik.

   Tentukan tataletak soket stop kontak pada lokasi2 yang sesuai dengan jumlah yang memadai.

7. **Medan Statis (Static)**
   Jika akan digunakan karpet sebagai pelapis lantai ruangan lab, maka harus diperhatikan masalah medan listrik statis yang dapat ditimbulkan oleh bahan karpet tersebut.

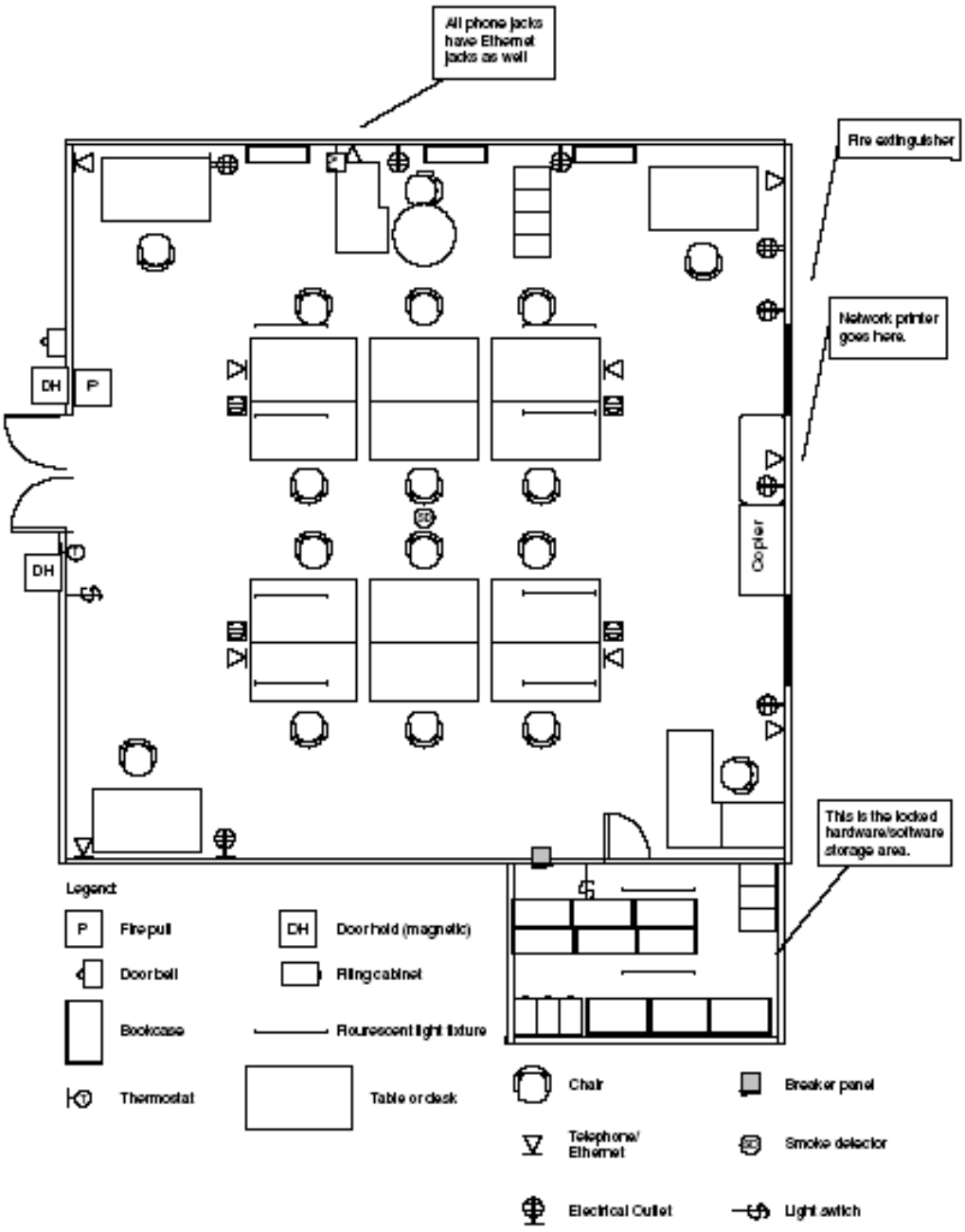
   Perlu dipilih bahan yang tidak menimbulkan medan listrik statis yang dapat merusak komponen2 elektronik.

   Ruangan harus dilengkapi dengan alas kaki khusus anti statik dan sistem pertahanan yang baik untuk mencegah timbulnya medan statik ini.
8. **Fasilitas**
   Fasilitas untuk karyawan laboratorium, seperti: toilet, tangga, elevator, dan sebagainya (termasuk sarana akses bagi penyandang cacat) adalah penting tidak hanya karena alasan legal saja, tetapi akan mempengaruhi produktifitas dari tim.

   Fasilitas saluran komunikasi: jaringan telepon (PTSN, BRI ISDN, PRI ISDN, T1, OC3 dan sebagainya), jaringan komputer (ethernet cat3, cat 5, token ring, serat optik, atau ATM) dan koneksi2 lainnya.

   Untuk memudahkan pengelolaan kabel jaringan disarankan lab menggunakan lantai yang dinaikkan (*raised floor*).

   Bila dibutuhkan, bisa disiapkan pula koneksi untuk saluran air dan pembuangannya. Kadang2 diperlukan air mengalir untuk pendinginan.



Gambar. Contoh tata letak sebuah laboratorium pengujian.

D. Persediaan dalam Laboratorium Pengujian

Barang-barang dan peralatan-peralatan yang diperlukan dalam suatu laboratorium pengujian akan berbeda-beda tergantung dari jenis pengujian yang akan dilakukan.

Faktor2 yang mempengaruhinya adalah: jenis sistem yang akan diuji.

Pentingnya akan suatu kebutuhan tergantung sekali terhadap jenis pengujian yang akan dilakukan.

Kebutuhan laboratorium pengujian perangkat keras juga akan berbeda dengan kebutuhan laboratorium yang digunakan untuk pengujian perangkat lunak.

Pokok bahasan berikut ini akan membahas kebutuhan-kebutuhan yang diperlukan untuk melengkapi suatu laboratorium pengujian.

Perangkat lunak : Sistem Operasi, aplikasi, alat bantu pengujian dan perangkat lunak pelengkap/pendukung lainnya (utilities).

Perangkat Keras : Pc Cards, Monitor/video cards, printer dan scanner, modem/network cards, data storage, surge protector / UPS units, cables, Jaringan Komputer.

Bahan Habis Pakai : Computer Media, Alat tulis/kantor, printer supplies.

Perabotan : Mesin fotocopy dan printer, mesin penghancur dokumen, meja kerja (bench), meja kantor, kursi tanpa sandaran, kursi kerja, alas mouse, alas anti statis.

Peralatan : Instrumen Khusus (Osiloskop, Multimeter), Perangkat L

E. Masalah Keamanan dan Penelusuran

Keamanan fisik perangkat keras dan perangkat lunak: pencegahan terhadap terjadinya kerusakan, pencurian, dan kehilangan lainnya.

Ada satu orang yang bertanggung jawab setiap saat terhadap suatu barang yang ada dalam laboratorium.

Menerapkan konsep *due dilligence* terhadap anggota tes tim

Membertimbangkan kebutuhan akan asuransi terhadap aset laboratorium pengujian

Perhatian terhadap hak milik intelektual

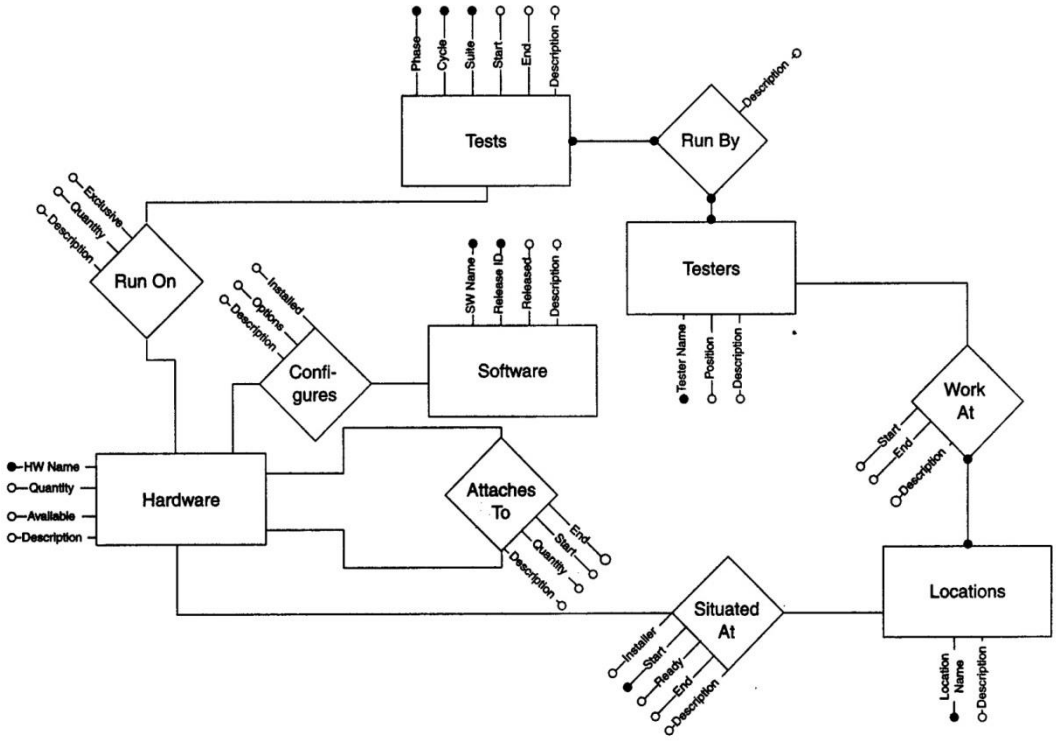
Sebelum meninggalkan ruangan maka meja kerja arus dibersihkan dan mengunci laci & lemari

F. Pengelolaan Peralatan dan Konfigurasi

Pendekatan umum: mengunakan label asset.

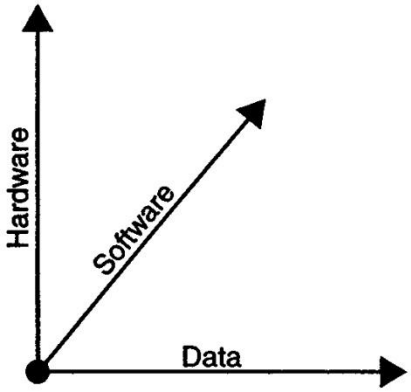
Konfigurasi perangkat keras dan perangkat lunak: **basis data logistic.**

Basis data logistik tidak hanya digunakan untuk mencatat **konfigurasi perangkat lunak** dan **perangkat keras** saja, tetap digunakan juga untuk meyimpan informasi tentang **konfigurasi data.**



Gambar. ER Diagram Basis Data Logistik

Diluar konfigurasi perangkat keras dan perangkat lunak, dimensi lainnya yang penting dalam sebuah konfigurasi test platform adalah data.



Gambar . Konfigurasi Data

Termasuk dalam dimensi data adalah basis data dan file konfigurasi (Windows Registry atau Unix cron tables) seperti test input, output, dan log files.

Data dapat mempengaruhi tingkah laku sistem sama seperti hardware dan software.

Tantangan lainnya dalam pengelolaan data pada proses testing adalah data selalu berubah selama pelaksanaan proses testing.

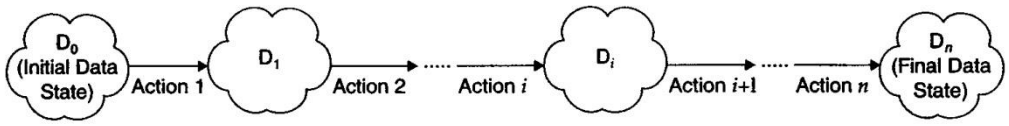


Figure 7.6 Data configuration changes resulting from setup and test actions.

G. Menjaga Kebersihan lingkungan Laboratorium

Seperti yang telah ditekankan sebelumnya, adalah penting untuk mengetahui secara tepat kondisi atau konfigurasi pada masing-masing sistem pengujian setiap saat selama proses pengujian berlangsung.

Bila tidak, maka bug akan tidak dapat dihasilkan kembali dan laporan bug menjadi tidak akurat.

Meskipun hal ini terjadi, namun pada kenyataannya kebanyakan orang menganggap remeh terhadap pentingnya pengelolaan konfigurasi dalam lingkungan pengujian.

Pada proses pencarian kesalahan, kebanyakan *programmer* ingin untuk dapat mengubah konfigurasi sistem dengan cepat dan segera melakukan percobaan; tanpa

mendokumentasikan setiap langkah perubahan yang dilakukan yang dapat memperlambat proses pengujian.

Namun disayangkan, tindakan ini tidak selaras dengan tujuan untuk memelihara konfigurasi dari platform pengujian yang bertentangan dengan keinginan untuk memperbaiki kesalahan dengan cepat.

Dari sudut panjang pengujian, solusi yang mungkin paling baik untuk masalah ini adalah mencegah dilakukaknnya setiap aktifitas penelusuran dan perbaikan kesalahan pada platformpengujian. Tim pengembang seharusnya memiliki laboratorium pengembangan tersendiri yang terpisah dari fasilitas pengujian yang akan menhindari keperluan untuk bekerja dalam laboratorium pengujian.

Dalam prakteknya, kadang-kadang dapat dijumpai adanya bug yang hanya muncul pada lingkungan pengujian karena lingkungan pengujian dikonfigurasikan semirip mungkin dengan konfigurasi yang nantinya akan digunakan oleh pengguna.

Dalam prakteknya tidak ada solusi yang mudah untuk hal ini. Di salah satu pihak, bila proses debugging harus dilakukan di luar test lab maka dapat berdampak terhadap waktu pelaksanaan pengujian yang menjadi tertunda. Di pihak lainnya, bila tim pengembang dibiarkan melakukan *debugging* pada *test platform* maka akan dapat timbul kekacauan pada *test platform* itu sendiri yang menimbulkan masalah pada proses *testing.*

Sehingga bila Anda menghadapi situasi dimana Anda harus berbagai sumber daya dengan tim pengembang maka salah satu cara yang dilakukan untuk menghidari kekacauan konfigurasi dari platform pengujian adalah dengan melakukan back-up sebelum sistem digunakan oleh tim pengembang/programmer.

Backup konfigurasi ini dapat disimpan dalam suatu kepustakaan konfigurasi pada suatu media memori eksternal seperti tape atau *network drive.*

Mungkin tim pengujian membutuhkan konfigurasi D_0 sementara pihak pengembang menginingkan konfigurasi D_i dan D_j. Semua konfigurasi ini dapat disimpan dalam kepustakaan konfigurasi.

H. Faktor Manusia

Meskipun fokus bahasan pada topik ini adalah lebih kepada material/bahan-bahan dalam sebuah laboratorium dan segala informasi yang berhubungan dengan material/bahan-

bahan tersebut, namun perlu juga diingat orang-orang yang bekerja dalam laoratorium pengujian tersebut.

Ada 3 hal yang harus diperhatikan berkaitan dengan faktor manusia yang bekerja dalam sebuah laboratorium, yaitu :

1. Keamanan dalam laboratorium.
2. Kerusakan pada peralatan laboratorium
3. Produktivitas dalam laboratorium

I. Keamanan Dalam Labotarium

Laboratorium pengujian perangkat lunak pada umumnya adalah merupakan laboratorium biasa dimana tidak ada resiko terhadap bahaya kimia, listrik, radiasi, atau mekanik.

Namun demikian, setiap orang yang bekerja dalam laboratorium harus memahami aspek-aspek keamanan dasar dan melaksanakannya sebagai kebiasaan.

Segala tindakan yang dilakukan tanpa dipikirkan dan kecerobohan dapat membahayakan, melukai diri sendiri atau orang lain, atau merusak peralatan, sistem, dan fasilitas dalam laboratorium.

Sebagai contoh: bila seseorang bekerja dengan perangkat keras komputer, maka sebaiknya menggunakan jas laboratorium. Atau apabila dianggap jas lab terlalu berlebihan maka paling tidak orang tersebut bekerja dengan tidak menggunakan baju yang sangat longgar dan dimohon untuk menyelipkan atau mengikat setiap bagian yang dapat terlepas & menyangkut seperti dasi, bandana (seperti dasi pramuka), dan syal.

Rambut dan perhiasan juga dapat menyebabkan masalah yang serupa. Jenggot, rambut panjang, kumis yang panjang dalam kondisi tertentu dapat tersangkut pada bagian komputer yang bergerak (mis. fan).

Rambut yang basah atau lembab dapat menimbulkan hubung singkat pada rangkaian elektronik yang sedang menyala.

Kalung yang longgar, cincin, dan gelang dapat tersangkut pada bagian2 komputer yang menonjol. Selain itu perhiasan-perhiasan yang lain seperti anting-anting telinga dan hidung, serta perhiasan-perhiasan lainnya yang menempel pada bagian-bagian tubuh yang tidak terlindung dapat tersangkut yang akan menimbulkan rasa sakit atau bahkan mengakibatkan luka yang berbahaya.

Bagian2 komputer yang tajam juga harus diwaspadai karena dapat melukai tubuh.

Mungkin dapat dipertimbangkan penggunaan sarung tangan saat bekerja dengan perangkat keras, tetapi sarung tangan kadang2 juga menyulitkan apabila kita sedang bekerja dengan komponen2 yang kecil, seperti IC/chip, jumper, dan mur/sekrup.

Pelindung mata kadang2 juga diperlukan apabila kita bekerja dengan komponen2 perangkat keras.

Detektor asap, alaram kebakaran, dan peralatan pemadam kebakaran harus selalu tersedia dalam lab.

Para pekerja juga harus mengetahui jalur evakuasi saat terjadi keadaan darurat.

Untuk menghindari terjadinya cedera pada anggota2 tubuh seperti tangan, pergelangan tangan, dan tulang belakang maka tata letak papan ketik dan *mouse* harus diletakan sedemikian rupa sehingga memberi kenyamanan pada orang yang bekerja dan menghindari terjadinya cedera pada tubuh.

J. Kerusakan Peralatan Laboratorium

Pekerja dalam laboratorium juga dapat mengakibatkan kerusakan pada peralatan. Contoh: menumpahkan cairan pada papan ketik, jari yang berminyak dapat menyebabkan touchpad menjadi tidak sensitif.

Pada umumnya organisasi melarang karyawan bekerja sambil makan atau minum di dalam laboratorium.

Keberadaan magnet dalam lab juga dapat merusak media magnetik, seperti disket dan kartu magnetic.

Klip kertas, pin logam, dan obyek logam kecil lainnya dapat terjatuh dan tersangkut dalam komputer yang akan mengakibatkan terjadinya hubung singkat dan merusak computer.

Guncangan, getaran, dan terjatuh juga menjadi penyebab kerusakan peralatan.

Pergunakanlah peralatan yang sesuai saat bekerja dengan perangkat keras.

Pikirlah dua kali sebelum mengerjakan sesuatu. Bekerjalah seteliti mungkin.

Gunakan gelang dan alas anti statis dengan sistem pentanahan yang baik bila bekerja dengan komponen-komponen elektronik (IC / micro chip).

Hati-hatilah saat membersihkan peralatan, gunakanlah cairan pembersih dan peralatan yang sesuai. Lakukanlah dengan hati-hati.

K. Produktivitas Dalam Laboratorium Pengujian

Penataan tata letak dalam laboratorium menjadi faktor yang penting bagi produktivitas pekerja.

Peralatan dan workstation harus ditata sedemikian rupa sehingga meminimumkan pergerakan pekerja dalam laboratorium.

Keberadaan radio, TV, CD, tape, dan sarana hiburan lainnya mungkin juga dapat mengakibatkan penurunan produktivitas kerja oleh karena mengganggu konsentrasi pekerja.

Produktivitas dapat ditingkatkan dengan memberikan dua buah workstation untuk masing-masing pekerja.
Satu *workstation* digunakan untuk mengakses alat bantu pengujian seberti *bug tracking database, test tracking spreadsheet, system configuration tracking database,* dan sumber-sumber informasi serta sarana pelaporan lainnya yang diperlukan untuk melaksanakan pengujian dan pelaporannya.

Sedangkan *workstation* yang lainnya digunakan untuk melakukan pengujian.

PERTEMUAN XI

PENGUJIAN DAN IMPLEMENTASI SISTEM

PENGELOLAAN TIM PENGUJIAN

MODUL KULIAH

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 11 (Sebelas) Waktu :

Modul	11 (Sebelas)
Topik	Pengelolaan Tim Pengujian
Sub Topik	-
Materi	• Staffing and Managing a Test Team • Jumlah Personel • Keahlian • Jabatan • Model Organisasi • Hiring Tester • Test Specialist and Content Spesialist • Kualifikasi yang tidak tepat • Hal hal yang membangkitkan Motivasi • Hal hal yang dapat menurunkan Motivasi Team • Pemanfaatan pekerja Temporer
Tujuan	Mahasiswa dapat menerangkan dan membenuk pengelolaan tim pengujian

A. Staffing and Managing a Test Team

Most Basic Principle for Managing Staff :

1. Hiring (Memberikan pekerjaan dalam mengembangkan diri)
2. Motivating
3. Retaining (mempertahankan/ memelihara)

What kind of people make good Test Engineers ?:

1. Professional Pessimism
2. Balanced Curiosity
3. Focus: No Space Cadets
4. Avoid the Aspiring Hero
5. Shun the Sloth
6. Reject Casper Milquetoast

Defining the Test Team

1. Size
2. Skills :
    - ✓ Technology
    - ✓ Application Domain
    - ✓ Testing
3. Education and training
4. Positions, experience and goals

B. Jumlah Personel

Ada dua cara untuk menentukan jumlah anggota suatu test team, yaitu :

1. Berdasarkan **tugas/*task*** yang harus dikerjakan.
2. Berdasarkan rasio antara ***tester*** dan ***developer.***

Rasio yang ideal antara tester dan developer 1:1, umumnya berkisar 1:2 s.d. 1:5

C. Keahlian

Dalam beberapa kasus ***testing*** diperlukan tenaga yang memiliki suatu keahlian khusus.

Harus dapat membaca dan belajar secara cepat.
Dapat menulis laporan dengan baik.

Dalam beberapa kasus diperlukan tenaga yang menguasai statistik dan matematika.

D. Jabatan

1. ***Test engineer*** : *programmer,mechanical engineer*, dan *electrical engineer.*
2. ***Test Technician.***
3. ***Test Manager***
4. Dalam beberapa kasus mungkin diperlukan juga ***release management engineer*** dan ***configuration management engineer***

E. Model Organisasi

Skill-based Model: sangat direkomen-dasikan pada sistem2 yang rumit dan berteknologi canggih.

Project-bases Organization : pembagian tugas berdasarkan proyek yang dikerjakaan, bukan berdasarkan spesialisasinya.

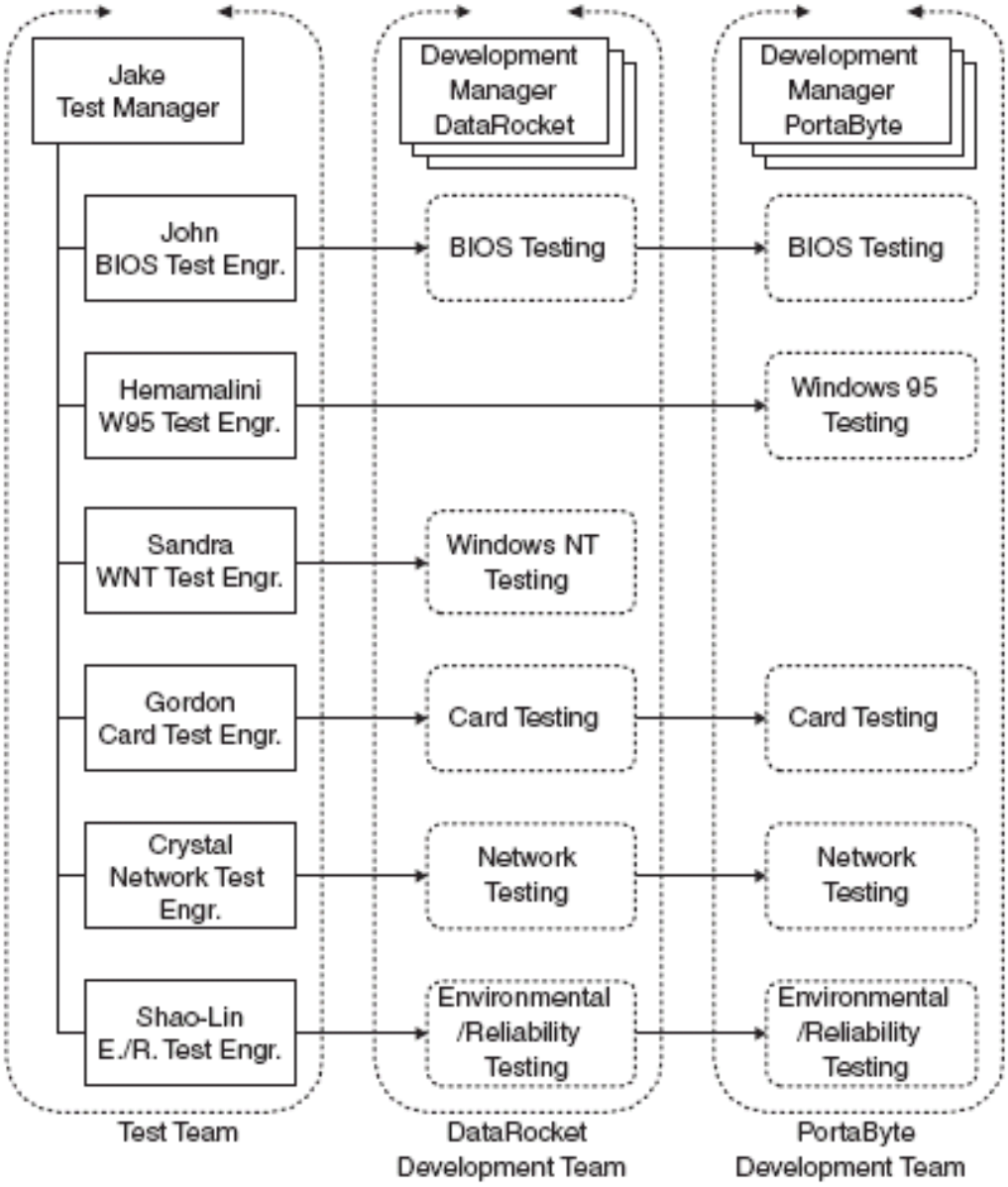


Figure 8.4 A skills-based test organization.

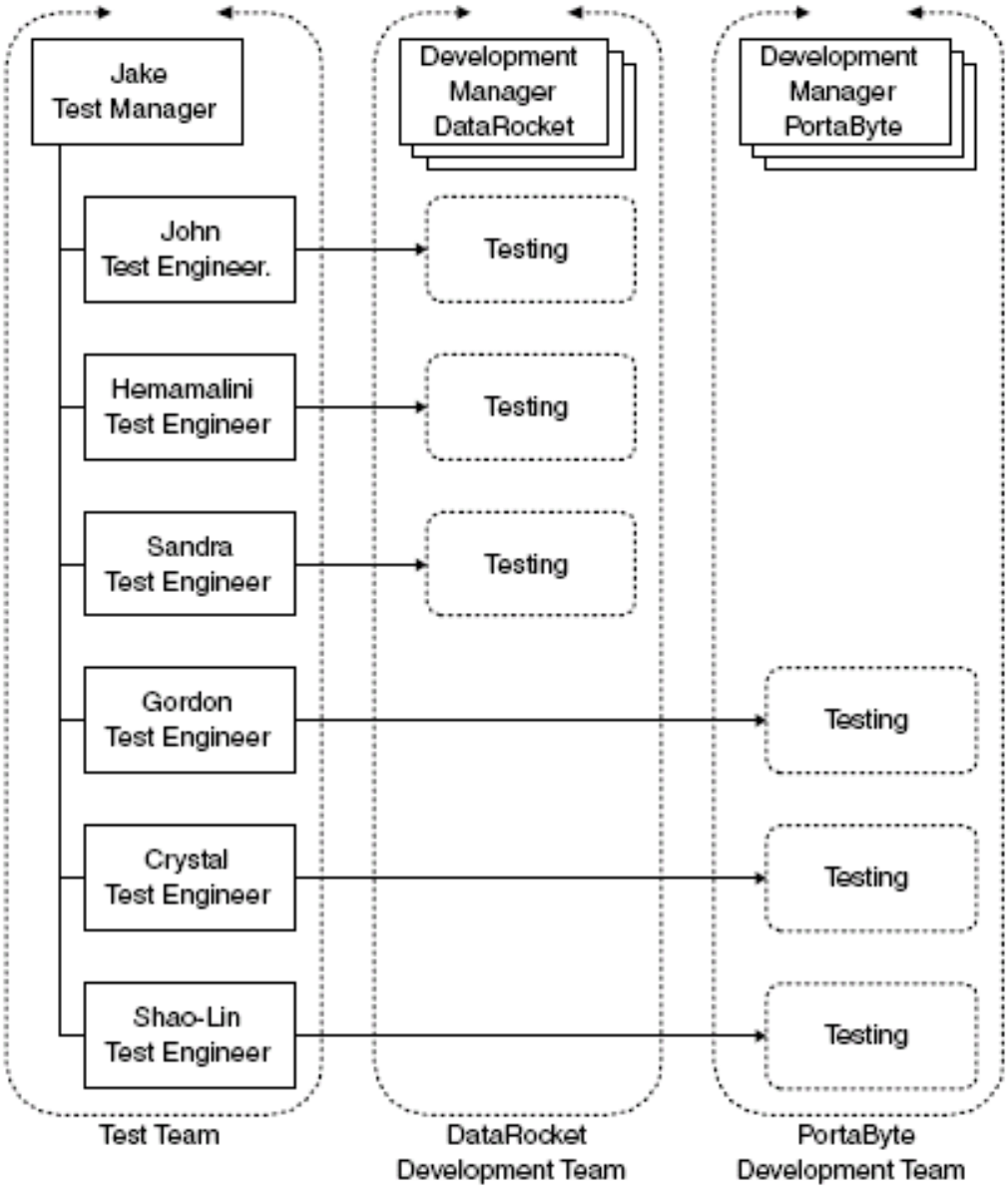


Figure 8.5 A project-based test organization.

F. Hiring Tester

1. Defining the Job
2. Gathering and Screening Resumes
3. On-site Interviews
4. Making the Hiring Decision
5. Avoiding and Undoing Hiring Mistakes
6. Bringing the New Tester on board

Adapun Kualifikasinya adalah :

1. Memiliki latar belakang teknik.
2. Memiliki komitment pada bidang testing sebagai spesialisasi.

3. Selalu memiliki rasa ingin tahu.
4. Dapat berkonsentrasi dengan baik dan terfokus.
5. Kemampuan analisa yang baik.
6. Kreatif.

G. Test Specialist & Content Spesialist

Test Specialists : orang yang meniti karir khusus pada bidang testing, menduduki posisi sebagai *test engineer*

Content Specialists : orang yang mengerti produk perusahaan dan teknologinya secara menyeluruh, dapat melakukan *functional testing* secara baik tetapi mungkin tidak begitu menguasai proses testing secara terinci

H. Kualifikasi yang tidak tepat

1. Orang yang ingin mendapatkan jabatan/ posisi yang *prestige* dalam organisasi.
2. Tidak memiliki dorongan untuk mengerjakan tugasnya.
3. Orang yang memiliki sifat menjatuhkan orang lain

I. Hal Hal yang membangkitkan Motivasi

1. Bertindak bijaksana pada anggota team yang melakukan kesalahan.
2. Pemberian upah yang standar sesuai dengan tanggung jawab dan keahliannya.
3. Jenjang jabatan yang jelas.
4. Adanya kompensasi untuk *over-time.*
5. Pemberian pelatihan, kesempatan mengikuti seminar, konferensi, dan rekreasi.

J. Hal Hal yang dapat Menurunkan Motivasi Team

1. Pemberian bonus berdasarkan ketepatan pencapaian jadwal yang telah ditentukan.
2. Pemberian bonus berdasarkan jumlah bugs yang ditemukan.
3. Pemberian “pizza” (*junk-food*) sebagai menu makan malam.
4. Mengkonfrontasi *test team* dengan *development team* atas problem yang ditemukan.

K. Pemanfaatan Pekerja Temporer

Test technician yang memiliki peran yang penting tetapi bukan merupakan peran yang kritis dalam implementasi. Diantaranya :

- ✓ Konsultan
- ✓ Traniner
- ✓ Tenaga Ahli
- ✓ Kontraktor

PERTEMUAN XII

PENGUJIAN DAN IMPLEMENTASI SISTEM

TANTANGAN ORGANISASI BAGI MANAJER PENGUJIAN

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 12 (Dua belas) Waktu :

Modul	12 (Dua Belas)
Topik	Tantangan Organisasi Bagi Manajer Pengujian
Sub Topik	-
Materi	• Jabatan yang diberikan pada test manager • Definisi yang diberikan oleh standar IEEE • Organisasi Test Group • Organisasi testing sebagai sebuah sumberdaya proyek pengembangan. • Organisasi testing yang indepeden • Menambah fungsi fungsi lain dalam organisasi • Kelompok Pengembangan layanan. • Bekerja dengan manajer lain • Hal hal yang mempercepat proses testing • Hal hal yang memperlambat proses testing • Tanda tanda pembubaran unit test • Mempresentasikan hasil pengujian.
Tujuan	Mahasiswa dapat menerangkan **tantangan organisasi bagi manajer pengujian**

A. Jabatan yang diberikan pada Test Manager

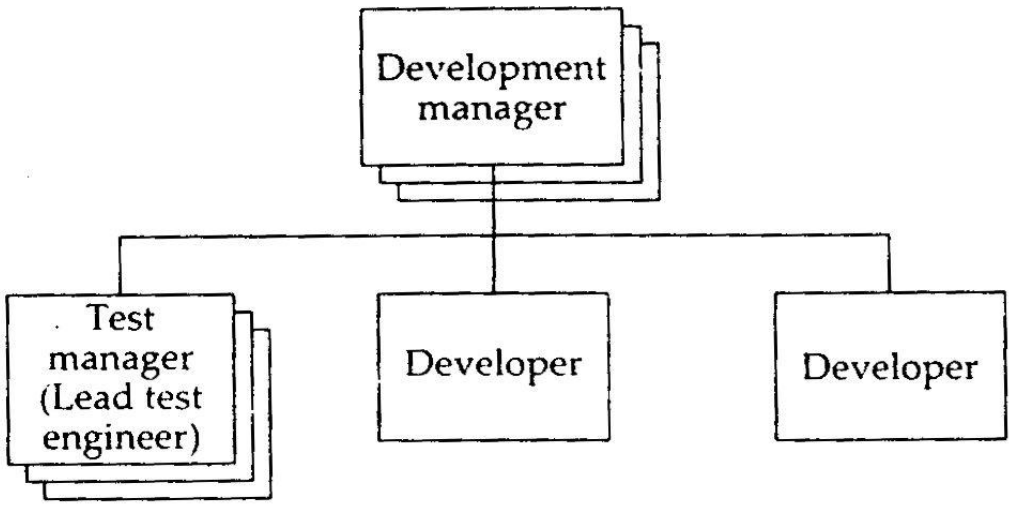
1. **Quality Control Manager**: bertanggung jawab melakukan pengetesan untuk mencari kesalahan.
2. **Quality Assurance Manager**: tidak hanya bertanggung jawab untuk menemukan kesalahan tetapi juga mencegah timbulnya kesalahan serta mengusahakan peningkatan kualitas produk secara terus-menerus.

B. Definisi yang diberikan oleh standar IEEE 610.12-1990

1. Sebuah pola yang direncanakan dan sistematis dari seluruh kegiatan yang diperlukan untuk menyakinkan bahwa sebuah produk telah sesuai dengan spesifikasi teknik yang telah ditentukan.
2. Sejumlah aktivitas yang dirancang untuk mengevaluasi proses pengembangan /pembuatan suatu produk.

Quality Control didefinisikan : Sejumlah aktifitas yang dirancang untuk mengevaluasi kualitas dari suatu produk yang dikembangkan/dibuat.

C. Organisasi Test Group



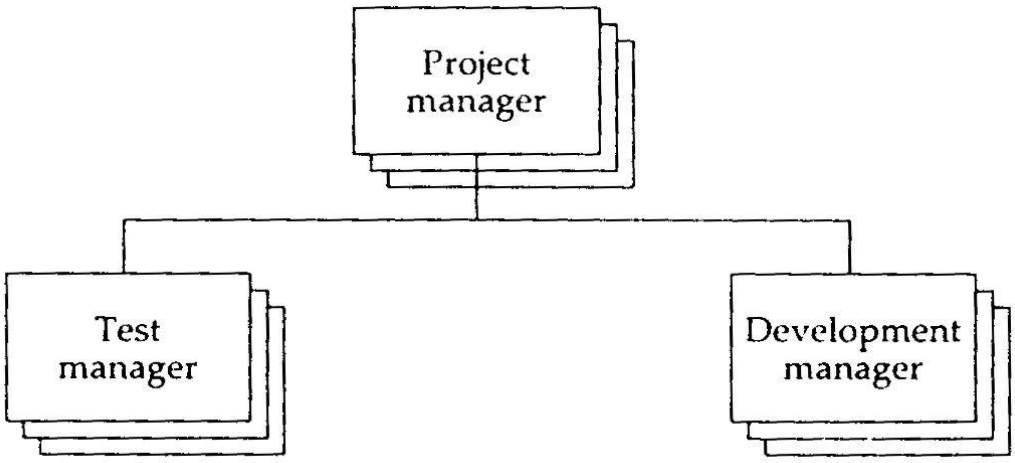
Gambar. Model Dasar Organisasi Testing.

- ✓ *Testing Manager* memberi laporan kepada *Development Manager*.
- ✓ *Test Engineer* dipimpin oleh seorang pemimpin (*Lead Test Engineer*) yang akan mengkomunikasi-kan hasil testing langsung pada *Development Manager*.
- ✓ Model ini cocok bila kita bekerja dalam suatu kelompok dengan jumlah personel yang sedikit (jumlahnya hanya belasan).

Kesalahan pada Model Dasar Organisasi Testing adalah :

1. Test group menjadi tidak independen
2. Proses testing tidak mendapatkan akses terhadap sumber daya yang dibutuhkannya.
3. Sumber daya yang dimilikinya tidak hanya digunakan untuk testing.
4. Peranan testing menjadi tidak jelas, karena hanya digunakan sebagai saran dalam pengembangan sistem, bahkan mungkin pelaku testing akan bekerja sebagai *developer* juga.

D. Organisasi Testing Sebagai Sebuah Sumberdaya Proyek Pengembangan.



Test Manager dan *Development Manager* keduanya memberi laporan kepada *Project Manager*.

Struktur organisasi testing ini bukan merupakan solusi yang sempurna tetapi merupakan perbaikan dari struktur organisasi testing sebelumnya.

Dalam struktur ini test group masih tidak independen seutuhnya, karena Test Manager memberikan jawaban kpd. Project Manager.

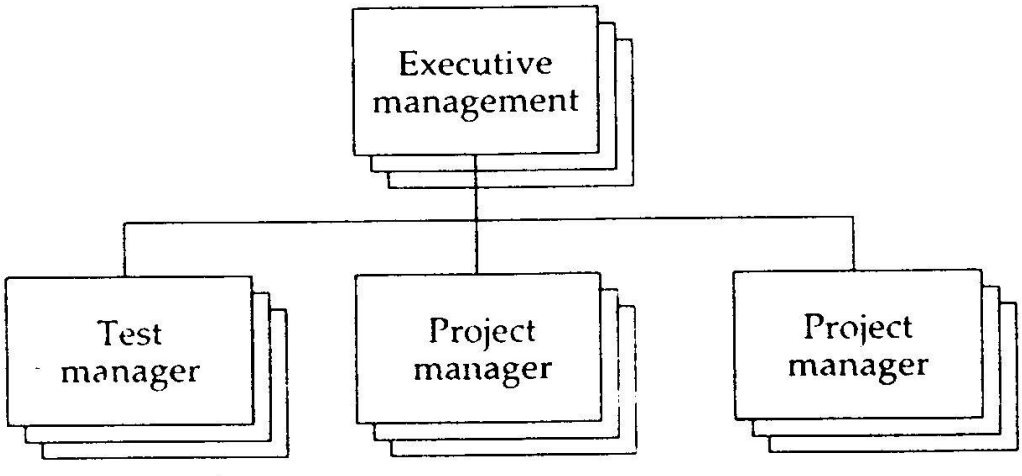
Kelebihannya, dalam struktur ini keterlibatan Project Manager dalam proses pengembangan sistem menjadi berkurang.

Bagian pengembangan dan bagian testing memiliki pegawai dan budget yang terpisah.

Unit testing dalam organisasi menjadi lebih berperanan karena seluruh laporan kesalahan langsung diberikan pada project management.

Kerugian : apabila dalam pelaksanaan suatu proyek ternyata tidak sesuai dengan jadwal, maka test group harus memberikan rasa simpati pada bagian pengembangan dan harus membantunya (melakukan kerja lembur) untuk menyelesaikan proyek tersebut agar tepat waktu.

E. Organisasi Testing yang independen



Merupakan model organisasi testing yang paling baik.Team tes dalam hal ini benar2 independen.

Perhatian dan tujuan manajemen adalah untuk mempromosikan keunggulan perusahaan, oleh karena itu manajemen akan menerima laporan status tes dengan pikiran yang terbuka.

Masalah2 yang berkaitan dg pengaruh, alokasi dana, dan sumberdaya manusia diminimumkan.

F. Menambah Fungsi fungsi lain dalam organisasi

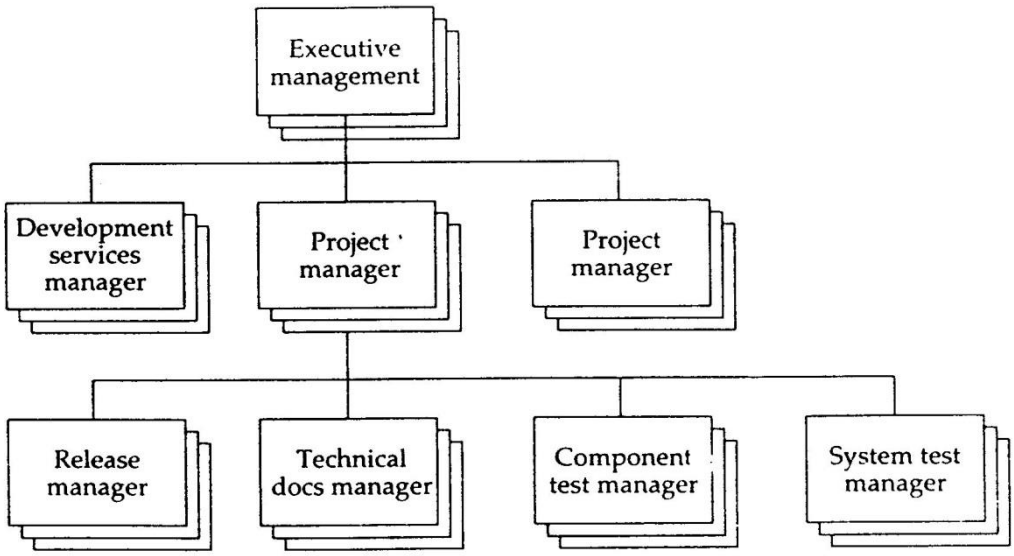
Pada umumnya Manajer Testing tidak melakukan tugas2 berikut ini :

1. Configuration Mgt. & Release Mgt
2. Customer Support.
3. Administrasi sistem dan penanggunalang bencana.
4. Manajemen Kualitas
5. Manajemen Laboratorium Pengembangan.
6. Membuat data utk. Pelatihan dan prosedur operasi.

Tetapi ada beberapa faktor yang mengharuskan bagian testing menambahkan fungsi2 lain tersebut :

1. Tim tes memiliki sebagian tanggung jawab thd pengembangan produk atau apabila test team secara keseluruhan terintegrasi dengan tim pengembangan.
2. Pihak manajemen memberikan tanggung jawab tambahan pada tim tes karena dianggap tim memiliki kemampuan yg lebih.
3. Tugas khusus diberikan pada anggota tim tertentu yang memiliki keahlian yang relevan.
4. Anggota tim tes ingin mendapatkan tambahan tanggung jawab sebagai bagian dari pengembangan karir.

G. Kelompok Pengembangan Layanan



H. Bekerja dengan Manajer lain

Proyek Tes Harus dikelola pada tiga arah yaitu :

1. **Inward**: menentukan tim tes, merekrut anggota, menentukan struktur org., memantau dan memotivasi anggota tim.
2. **Upward**: menyimpulkan status dari proses tes dan mengeskalasi problem yang penting, menentukan harapan yg hrs dicapai, memberikan tanggapan secara cepat, dan berpartisipasi dlm pertemuan manajemen.

3. **Outward**: mengkomunikasikan hasil tes, mengklarifikasi laporan masalah, mendiskusikan kebutuhan tes, dana, & layanan yang diperlukan pada pihak pengelola.

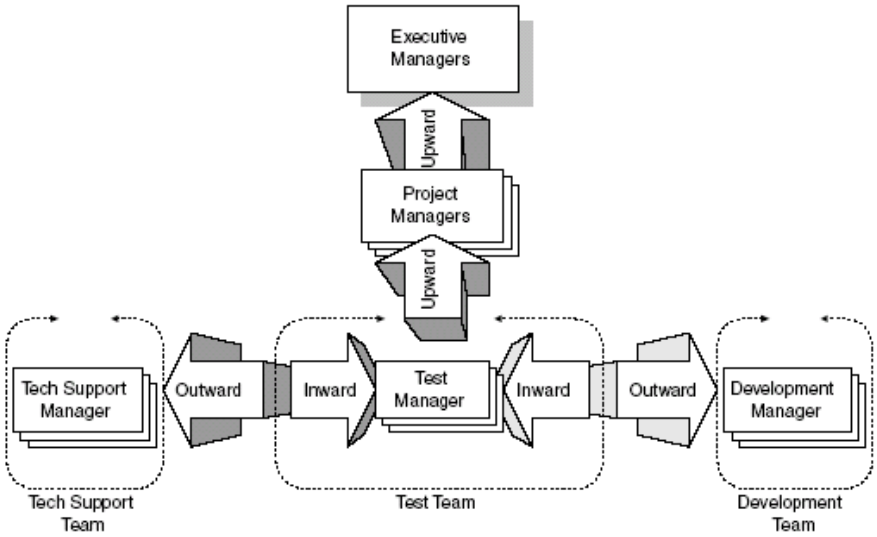
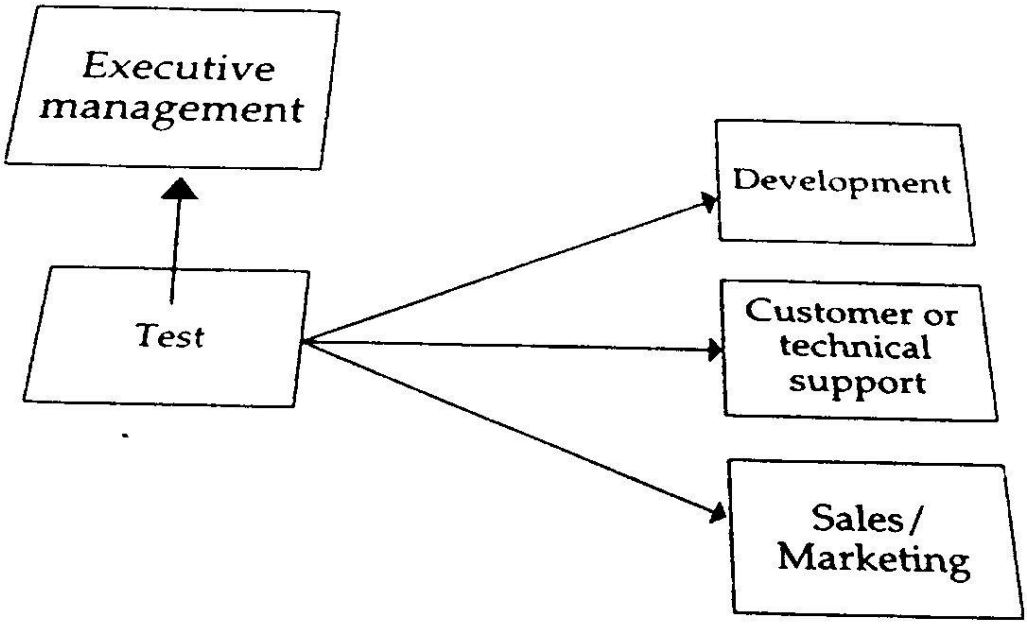


Figure 9.5 The three directions of test management.

Gambar. Working with other managers.



Gambar. Arah dalam Pengelolaan Tes

I. Hal hal yang Mempercepat Proses Testing

Test dilakukan sepenuhnya secara teliti, dengan memperhatikan 3 faktor :

1. Waktu : disiapkan secara benar, penemuan bug sedini mungkin, seluruh tim hrs. dilibatkan sejak awal.
2. Organisasi : keterbukaan komunikasi antara unit tes dengan tim lainnya.
3. Kebudayaan Perusahaan: unit tes merupakan jalan untuk mengurangi resiko dan filosofi manajemen bisnis diterapkan pada seluruh proyek.

J. Hal hal yang Memperlambat proses testing

1. Use of the Test Lab for Debugging
2. Buggy Deliverables
3. Violations of Test Plan Entry Criteria
4. Violations of Test Plan Exit Criteria
5. Scope Creep, Crawl, and Redefinition
6. Test Suite or Phase Cancellation
7. Tester Mistakes

K. Tanda tanda pembubaran unit test

1. Pengikutsertaan karyawan/manajer dalam ujian penilaian kecakapan karyawan.
2. Penurunan pendapatan perusahaan.
3. Masuknya akuntan atau konsultan yang berkerja sama dengan bagian personalia.
4. Adanya rumor tentang daftar karyawan yang akan diberhentikan.

L. Mempresentasikan hasil pengujian

1. Good Ways to Deliver Bad News
2. Institutionalizing a Test Dashboard
3. The Importance of Accuracy and Audience

PERTEMUAN XIII

PENGUJIAN DAN IMPLEMENTASI SISTEM

PENDISTRIBUSIAN PROYEK PENGUJIAN

UNIVERSITAS BINA SARANA INFORMATIKA

Mata Kuliah	**PENGUJIAN DAN IMPLEMENTASI SISTEM**
Semester	
Kelas	Sistem Informasi
Dosen	Achmad Baroqah M P M.Kom

Pertemuan : 13 (Tiga Belas) Waktu :

Modul	13 (Tiga Belas)
Topik	Pendistribusian Proyek Pengujian
Sub Topik	-
Materi	• Pendistribusian Proyek Testing • Pemilihan Rekanan • Pemasok / Vendor • Third-Party Testers • Kantor Pemasaran • Perencanaan Sebuah Proyek Testing terdistribusi • Menyusun, Mengkoordinasi dan membagi program testing • Pemetaan Kasus Testing • Pengelolaan Testing yang terdistribusi
Tujuan	Mahasiswa dapat menerapkan pendistribusian proyek pengujian apabila diperlukan.

A. Pendistribusian Proyek Testing

Dalam suatu proyek tes kadang2 ada pihak2 dan orang2 di luar organisasi testing yang harus turut terlibat. Bahkan kadang2 dalam suatu proyek tes harus melibatkan pihak2 atau orang2 dari luar organisasi.

Dapat dilakukan dengan 4 cara :

1. Melibatkan vendor : perusahaan yang menyediakan produk yang akan dintegrasikan pada produk yang sedang dikembangkan.
2. Melibatkan organisasi testing independen.
3. Melibatkan kantor pemasaran (terutama kantor di luar negeri untuk melakukan testing lokalisasi)
4. Melibatkan pemakai/pelanggan (beta testing)

Terdapat 3 tahap untuk melakukan pendistribusian proses pengujian :

1. Pemilihan rekanan, berdasarkan kebutuhan akan testing yang bersifat khusus.
2. Perencanaan
3. Pengelolaan proses testing yang dilakukan secara ekster-nal seperti bila kita melaku-kannya secara internal

B. Pemilihan Rekanan

Ada dua alasan untuk memilih rekanan yaitu :

1. Memanfaatkan kemampuan/ kelebihan yang dimiliki rekanan.
2. Untuk mengurangi beban pekerjaan

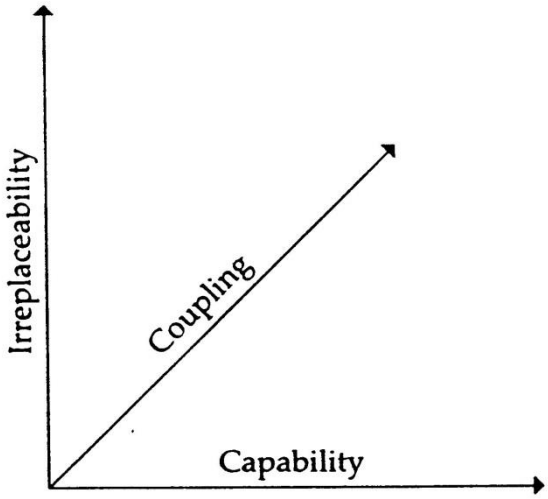
Pihak pihak yang dapat dijadikan rekanan dalam proyek testing yaitu :

1. Vendor/Pemasok
2. Third-Party Testers
3. Sales Office

C. Pemasok / Vendor

Ada 3 faktor utama utk melibatkan vendor dlm proses testing :

1. Irreplaceability (Tak dapat digantikan)
2. Coupling
3. Capability (Kemampuan)



D. Third-Party Testers

Sumber daya testing dari pihak ketiga adalah setiap organisasi yang menawarkan jasa testing kepada pelanggannya. Jasa ini dpt diberikan secara off site, on site, atau keduanya.

Berdasarkan caranya mengelola proyek testing, organisasi pihak ketiga yang menawarkan jasa testing dapat dibedakan menjadi 2, yaitu

1. Temporary Agency (Agen Sementara) : menyediakan ahli testing.
2. Perusahaan Testing Pihak Ketiga : membuat perencanaan suatu proyek testing dan pengelolaannya.

Keuntungan Pemanfaatan Third-Party Tester

1. Memiliki keahlian dalam bidang pengelolaan proyek testing dan memiliki pengalaman untuk melakukan testing secara teknis.
2. Dapat menyelesaikan proyek testing lebih cepat.

E. Kantor Pemasaran

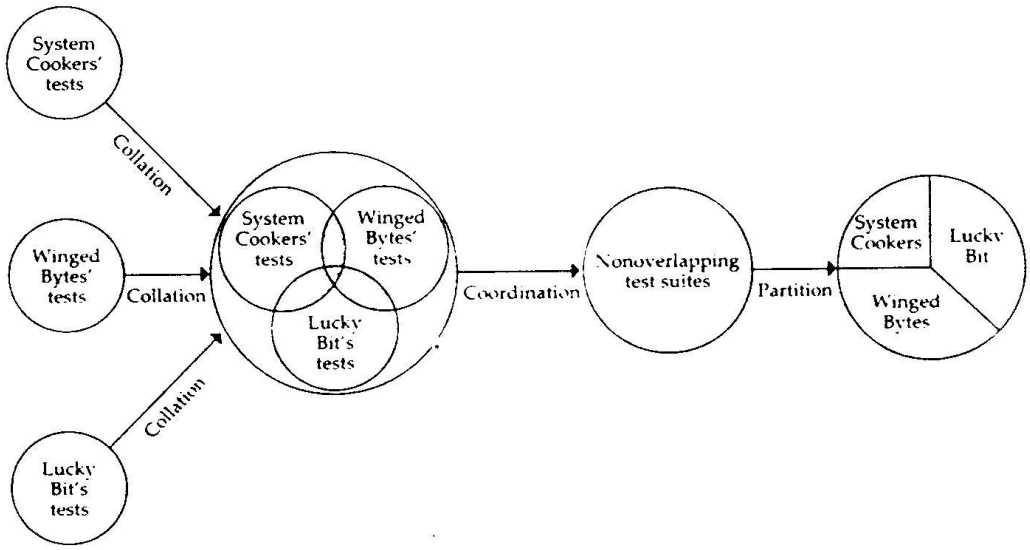
Kantor pemasaran yg tersebar di seluruh dunia dpt dimanfaat-kan untuk melakukan testing yang berhubungan dengan lokalisasi suatu system.

Kelemahannya: tidak dapat melakukan proses testing yang rumit atau yg bersifat terlalu teknis.

F. Perencanaan Sebuah Proyek Testing yang terdistribusi

1. Menilai Kemampuan
2. Biaya: biaya tetap dan biaya waktu dan material
3. Menyusun, mengkoordinasi, dan membagi program testing.
4. Pengelolaan Logistik
5. Pemetaan Kasus Testing

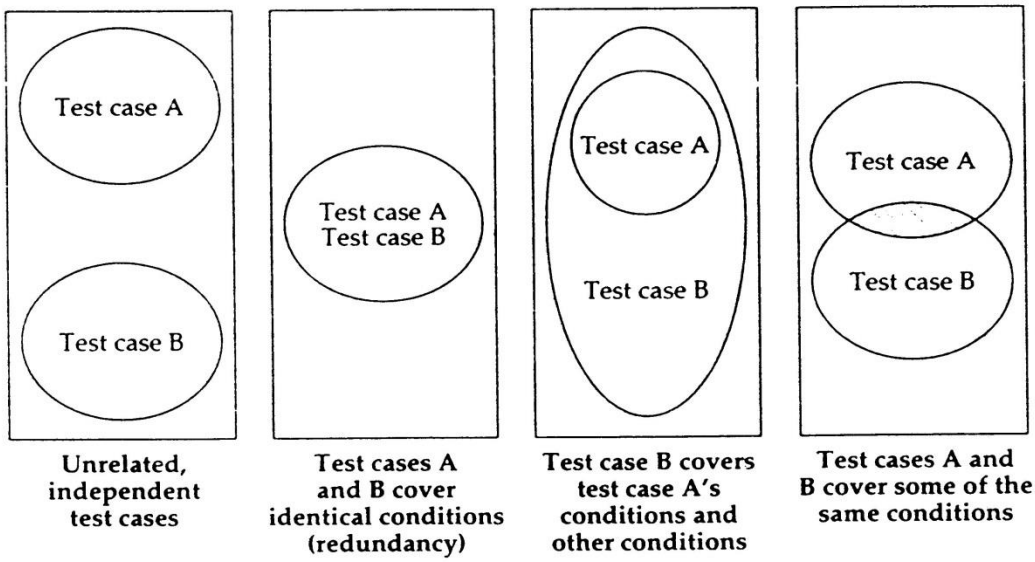
G. Menyusun , Mengkoordinasi dan Membagi Program Testing



.

H. Pemetaan Kasus Testing

Memetakan kasus testing yang dilakukan oleh pihak ketiga pada pola yang sesuai dengan kebutuhan. Dilakukan pada tahap koordinasi proyek testing.



I. Pengelolaan Testing yang Terdistribusi

1. Memantau pelaksanaan testing.
2. Mengkomunikasikan status testing.
3. Menangani Pertimbangan2 Politis

4. Peka terhadap terjadinya Pertentangan kebiasaan/ kebudayaan.
5. Menanamkan dan menjaga unsur kepercayaan

REFERENSI

1. Black, Red. ***Managing the Testing Process: Practical Tools and Techniques for Managing Software and Hardware Testing, 3rd Edition***. Wiley Publishing, Inc., Indianapolis, Indiana. 2009. (Buku Utama)
2. Sommerville, Ian. ***Software Engineering 10th Edition***. Pearson. 2015.
3. Pressman, R; Maxim, B. ***Software Engineering: A Practitioner's Approach 8th Edition***. McGraw-Hill Education. 2014
4. www.rexblackconsulting.com