Penjaminan Mutu
Perangkat Lunak

**Buku** Penjaminan Mutu Perangkat lunak ini terdiri atas tujuh Bab. Pada bab 1 menjelaskan tentang tujuan daan dasar-dasar perangkat lunak, kerusakan, biaya dan siklus hidup software.Bab2. Menjelaskan Proses dan teknik dalam melakukan pengujian, Bab 3. Menjelaskan metode pengujian dengan **white-box**, Bab 4. Menjelaskan tentang metode pengujian *Clean-room*, Bab 5. Menjelaskan metode pengujian dengan *Black-box* sedangkan pada bab 6 menjelaskan tentang implementasi pengujian dan pada Bab 7 menjelaskan Proyek Perangkat Lunak. Pandangan penulis , bahwa buku ini memiliki kelebihan dari buku lainnya yaitu buku ini sesuai dengan Rencana Pembelajaran Semester (RPS) matakuliah Penjaminan Mutu Perangkat Lunak pada IIB Darmajaya, sehingga kiranya dapat lebih membantu mahasiswa dalam memahami materi yang disampaikan oleh dosen pengampu. Buku ini juga dilengkapi dengan contoh soal serta rangkuman pada setiap bab nya sehingga dapat membantu pemahaman materi setiap bab nya.

DJ Press

ISBN 978-623-94028-3-8

9 786239 492038

Penjaminan Mutu Perangkat Lunak

Isnandar Agus, M.Kom | TM Zaini, M.Kom | Muhammad Fauzan Azima, M.T.I.

DARMAJAYA

# Penjaminan Mutu
# Perangkat Lunak

**Isnandar Agus, M.Kom.**
**TM Zaini, M.Kom.**
**Muhammad Fauzan Azima, M.T.I.**

Kampus Merdeka
INDONESIA JAYA

# Penjaminan Mutu Perangkat Lunak

**Isnandar Agus, M.Kom.**
**TM Zaini, M.Kom.**
**Muhammad Fauzan Azima, M.T.I.**

DarmaJaya Press

# PRAKATA

Puji Syukur kehadirat ALLAH Subhanahu Wataala atas segala nikmat dan Rahmat Nya sehingga buku ajar ini terselesaikan.

Buku ajar ini merupakan salah satu bentuk kegiatan peningkatan mutu pembelajaran yang diharapkan dapat memenuhi standar pembelajaran yang bermutu.

Penjaminan Mutu Perangkat Lunak merupakan salah satu materi yang diberikan kepada kalangan mahasiswa, dosen dan umum yang berminat mempelajari dan memperdalam tentang materi ini. Selain itu dengan adanya buku ini di harapkan mahasiswa lebih dapat memahami materi. Buku ini juga dilengkapi dengan soal-soal latihan yang ada di setiap bab guna melatih mahasiswa mengembangkan pengetahuannya.

Pada kesempatan ini kami mengucapkan terimakasih kepada pihak-pihak yang telah membantu sehingga buku ajar ini dapat diselesaikan. Kami menyadari buku ini terdapat kekurangan untuk itu kritik dan saran kami harapkan dari pembaca untuk kesempurnaan buku ini di masa mendatang.

Bandar Lampung, Februari 2021

Penyusun.

# DAFTAR ISI


**PRAKATA** ..... **v**
**DAFTAR ISI** ..... **vi**
**DAFTAR GAMBAR** ..... **ix**
**DAFTAR TABEL** ..... **x**

**BAB 1 KONSEP PENGUJIAN** ..... **1**
1.1. Tujuan Pengujian Perangkat Lunak ..... 2
1.2. Dasar-dasar Pengujian Perangkat Lunak ..... 2
1.3. Kerusakan Perangkat Lunak ..... 6
1.4. Biaya Pengujian ..... 7
1.5. Biaya Penyeimbang ..... 10
1.6. Siklus Hidup Sofware Secara Umum ..... 10
Rangkuman ..... 12
Latihan 1 ..... 13

**BAB 2 PROSES DAN TEKNIK PENGUJIAN** ..... **14**
2.1. Proses Pengujian ..... 15
2.2. Teknik Pengujian ..... 18
2.2.1. Pengujian Alur (Path) ..... 18
2.2.2. Desain Test Case ..... 19
2.2.3. Pengujian Integrasi ..... 21
2.2.4. Pengujian Interface ..... 23
Rangkuman ..... 24
Latihan 2 ..... 25

**BAB 3 PENGUJIAN WHITE-BOX** ..... **26**
3.1. Execution Testing ..... 28
3.2. Cyclomatic Complexity ..... 36
3.3. Graph Matrix ..... 43
3.4. Loop Testing ..... 45
Rangkuman ..... 53
Latihan 3 ..... 53

**BAB 4 PENGUJIAN CLEAN ROOM** ..... **54**
4.1. Pengujian Statistik ..... 54
4.2. Metode Clean Room ..... 55
Rangkuman ..... 57
Latihan 4 ..... 57

**BAB 5 PENGUJIAN BLACK-BOX** ..... **58**
5.1. Keuntungan dan Kerugian Black-Box ..... 59
5.2. Pengujian Modul Black-Box ..... 60
5.3. Data-based Black-Box Test Generation ..... 62
Rangkuman ..... 64
Latihan 5 ..... 64

**BAB 6 IMPLEMENTASI** ..... **65**
6.1. Menguji Sistem ..... 66
6.2. Menyiapkan Rencana Konversi ..... 67
6.3. Menginstall Database ..... 70
6.4. Melatih Para Pengguna ..... 70
6.5. Beralih Ke Sistem Baru ..... 71
Rangkuman ..... 72
Latihan 6 ..... 72

**BAB 7 PROYEK PERANGKAT LUNAK** ..... **73**
7.1. Manusia dalam pengembangan Perangkat Lunak ..... 74
7.2. Kebutuhan Produk Perangkat Lunak ..... 78
7.3. Kualitas dan Proses Perangkat Lunak ..... 79
7.4. Proyek Perangkat Lunak ..... 81
Rangkuman ..... 82
Latihan 7 ..... 83

GLOSARIUM ..... 84
DAFTAR PUSTAKA ..... 88
INDEX ..... 90
TENTANG PENULIS ..... 92

# 1 KONSEP PENGUJIAN

| | | |
|---|---|---|
| ***Tujuan intruksional umum*** | : | ***Memahami konsep dan ilmu pengujian perangkat lunak dan mengerti macam perangkat lunak*** |
| ***Tujuan instruksional khusus*** | : | • ***Memahami dan mengerti definisi pengujian perangkat lunak***<br>• ***Memahami tujuan pengujian perangkat lunak dan bagaimana perangkat lunak di uji.***<br>• ***Mengerti perbedaan pengujian perangkat lunak*** |

Sekali kode (*coding*) dibuat, pengujian (testing) *program* dimulai. Proses pengujian berfokus pada logika internal perangkat lunak, memastikan bahwa semua pernyataan sudah di uji, dan pada eksternal fungsional yaitu mengarahkan pengujian untuk menemukan kesalahan-kesalahan (*error*) dan memastikan bahwa *input* yang dibatasi akan memberikan hasil aktual (nyata) yang sesuai dengan hasil yang dibutuhkan.

Desain definisi pengujian perangkat lunak, menurut Myers proses menjalankan program dengan maksud menemukan kesalahan . Sedangkan menurut **IEEE (1990)**, pengujian yaitu : a). Proses sistem operasi atau komponen menurut kondisi tertentu, pengamatan atau pencatatan hasil dan mengevaluasi beberapa

Sebelum melakukan testabilitas, kiranya seorang penguji baiknya mengenal terlebih dahulu karakteristik atribut-atribut testabilitas, diantaranya yaitu :

**a. *Operability***

"Semakin baik perangkat lunak (*software*) berkerja, akan membuat *software* dites dengan lebih efisien."

Sistem mempunyai *bug* baru (*bug* menambahkan biaya tak langsung pada proses pengujian, dengan adanya analisa dan pelaporan). Tidak ada *bug* yang menghentikan eksekusi pengujian dimana produk aka berubah dalam tahap fungsional (memungkinkan pengembangan dan pengujian dilakukan secara simultan).

**b. *Obserability***

"Apa yang Anda lihat, adalah apa yang Anda tes."

Hasil dari setiap keluaran (*ouput*) harus menunjukkan hasil dari masukan (*input*). Kondisi sistem variabel dapat dilihat/di*query* selama eksekusi berlangsung. Kondisi dan variabel sistem lama juga dapat dilihat atau di*query dan* semua faktor yang mempengaruhi keluaran dapat dilihat adapun keluaran (*output*) yang salah dapat dengan mudah diidentifikasikan sedangkan kesalahan internal secara otomatis dideteksi dalam mekanisme pengujian secara menyeluruh. Kesalahan yang terjadi secara internal seharusnya otomatis dilaporkan serta *source code* sebaiknya dapat diakses guna memudahkan dalam perawatan (*maintenance*) dan pengembangan (*development*).

**c. *Controlability***

"Dengan semakin baik kita dapat mengendalikan *software*, semakin banyak pengujian memungkinkan dapat di lakukannya aktivitas secara otomatisasi dan dioptimalisasi."

Semua kemungkinan keluaran (*ouput*) dihasilkan dari berbagai kombinasi masukan dimana semua kode dieksekusi dari beberapa kombinasi masukan. Kondisi dan spesifikasi perangkat keras (*hardware*), perangkat lunak (*software*) serta variabel dapat

Disain mudah dimengerti dan dipahami dengan baik. Keterkaitan antara internal, eksternal dan *share* komponen dipahami dengan baik. Perubahan disain dikomunikasikan dan dokumentasi teknis dapat dengan mudah diakses dimana dokumentasi secara teknis dapat diorganisasikan dengan baik .

Dokumentasi teknis berisi spesifikasi dan rincian secara detil yang diperlukan dapat disajikan serta pendokumentasian secara teknis dapat memberikan informasi dan data tersaji secara akurat dan dapat di pertanggung jawabkan. Atribut-atribut di testabilitas yang disarankan oleh Bach dapat digunakan oleh perekayasa *software* untuk mengembangkan suatu konfigurasi *software* (seperti program, data dan dokumen) yang akan dapat membantu proses pengujian.

Tahapan dalam pengujian yang baik mempunyai kemungkinan yang tinggi dalam menentukan *error*. Untuk mencapai tujuan ini, *tester* harus mengerti sistem perangkat lunak yang berjalan dan berusaha untuk mengembangkan gambaran dalam benaknya tentang bagaimana kira-kira aplikasi perangkat lunak tersebut akan dapat mengalami kegagalan (*fail*). Idealnya, kelas-kelas dari *failure* dicari. Suatu pengujian yang baik seharusnya tidak tumpang tindih (*redundant*). Waktu dan sumber daya pengujian terbatas. Tak ada satupun titik dalam pelaksanaan pengujian mempunyai tujuan yang sama dengan pengujian yang lainnya. Dlam hal ini perlu ditekankan bahwa setiap pengujian harus mempunyai tujuan yang berbeda.

Dalam kegiatan pengujian yang baik dapat memberikan hasil yang terbaik [5]. Dalam suatu tim pengujian yang memiliki keterbatasan terhadap intensi, waktu, sumber daya yang sama, akan melakukan eksekusi hanya pada subset dari pengujian ini. Dalam kasus tertentu, aktivitas yang dilakukan dalam pengujian mempunyai kemungkinan tertinggi dalam memperoleh kelas *error* yang seharusnya dapat dimanfaatkan.

**k. Kontrol Sumber dan Versi (*Source and Version Control*)**

Program yang telah kadaluwarsa mungkin akan dapat digunakan lagi bila ada revisi untuk memperbaikinya.

**l. Dokumentasi (*Documentation*)**

Pengguna tak dapat melihat operasi yang telah dideskripsikan dalam dokumen panduan.

**m. Kesalahan Pengujian (*Testing Errors*)**

Tester melakukan kesalahan selama pengujian dan berpikir bahwa sistem aplikasi berjalan tidak benar.

### 1.4. Biaya Pengujian

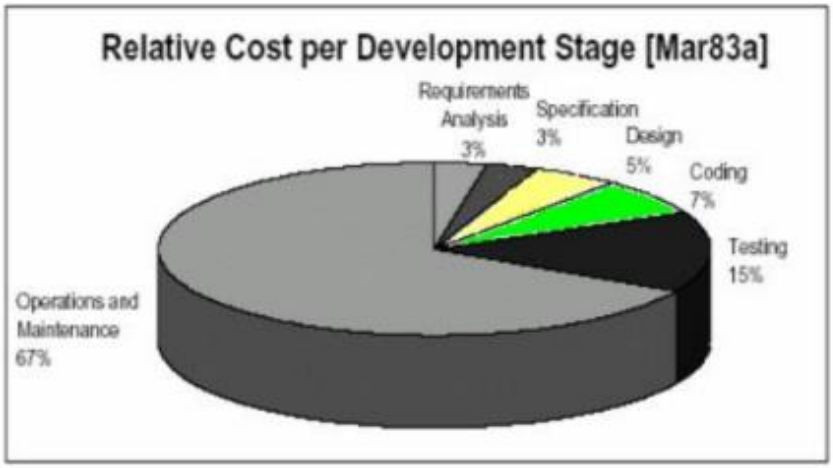
Dalam melakukan pengujian pada umumnya setiap melakukan pengujian secara sengaja atau tidak tentunya akan mengeluarkan biaya. Besarnya biaya yang dikeluarkan dalam pengujian tergantung dari atribut terhadap komponen yang terdapat pada objek yang di lakukan tahapan pengujian. Guna mencegah terjadi besarnya biaya pengujian maka perlunya untuk mengklasifikasikan biaya pencegahan kerusakan (*defect*) terhadap biaya penilaian dan evaluasi kerusakan yang terjadi. Ditunjukkan pada tabel 1 berikut ini.

Tabel 1. Klasifikasi biaya pencegahan kerusakan

| **B I A Y A** | |
|---|---|
| **PENCEGAHAN** | **PENILAIAN & EVALUASI** |
| Pelatihan Staff | Tinjauan disain |
| Analisa Kebutuhan | Inspeksi kode |
| Pembuatan prototipe awal | Glass-box testing |
| Nonaktifkan toleran disain (design fault-tolerant) | Black-box testing |
| Defensive Programming | Pelatihan Penguji |
| Analisa kegunaan | Pengujian Beta (*Beta Testing*) |
| Spesifikasi yang jelas | Otomatisasi pengujian |
| Dokumantasi Internal yang akurat | Pengujian Kegunaan (*Usability Testing*) |
| Evaluasi terhadap reliabilitas dari alat bantu pengembangan (sebelum membelinya) atau komponen lain dari produk yang potensial | Sebelum Menjalankan metode pengujian Kotak (*Pre-release out box testing*) oleh staf pelayanan pelanggan (*customer service*). |

| | |
|---|---|
| Terbuangnya waktu pemasaran | Hilangnya potencial presentasi |
| Terbuangnya promosi | Kegagalan pekanggan karena *software* |
| Biaya langsung dari keterlambatan pengiriman | Terjadinya *failure* dari tugas-tugas yang hanya dapat dilakukan sekali |
| Biaya atas hilangnya kesempatan akibat keterlambatan pengiriman | Biaya penggantian produk |
| | Biaya rekonfigurasi sistem |
| | Biaya pembenahan *software* |
| | Biaya dukungan teknisi |
| | Kecelakaan atau kematian |

Menurut studi dari Martin dan MC Clure [12] menyimpulkan bahwa biaya-biaya relatif pada setiap tahap pengembangan, seperti yang terlihat pada gambar 1 pada grafik di berikut ini.




Gambar 1. Biaya realtif setiap tahap pengembangan

Pada studi ini, pengujian terhitung sebesar 45% dari biaya pengembangan awal. Tahapan pengujian juga merupakan bagian integrasi dari perawatan (*maintenance*) namun juga pada bahasan ini tidak di bedakan secara khusus.

**Rangkuman**

Proses pengujian berfokus pada logika internal perangkat lunak, memastikan bahwa semua pernyataan sudah di uji, dan pada eksternal fungsional yaitu mengarahkan pengujian untuk menemukan kesalahan-kesalahan (*error*) dan memastikan bahwa *input* yang dibatasi akan memberikan hasil aktual (nyata) yang sesuai dengan hasil yang dibutuhkan.

Adapun tujuan dari **pengujian** secara **langsung** adalah untuk (1) mengidentifikasi dan menemukan sejumlah kesalahan yang mungkin terjadi dalam perangkat lunak yang diuji. (2). Setelah proses perangkat lunak dibetulkan maka dilakukan identifikasi kesalahan dan dilakukan pengujian kembali guna menjamin kualitas untuk tingkat penerimaan. (3). Melakukan tahapan pengujian yang lebih efisien dan efektif dengan anggaran dan jadwal yang ditetapkan dan batasan yang juga di tentukan sebelumnya.

Sedangkan tujuan pengujian secara tidak langsung adalah untuk mengumpulkan daftar kesalahan untuk digunakan dalam daftar pencegahan kesalahan (tindakan *corrective* dan *preventive*) secara menyeluruh.

Besarnya biaya yang dikeluarkan dalam pengujian tergantung dari atribut terhadap komponen yang terdapat pada objek yang di lakukan tahapan pengujian. Guna mencegah terjadi besarnya biaya pengujian maka perlunya untuk mengklasifikasikan biaya pencegahan kerusakan (*defect*) terhadap biaya penilaian dan evaluasi kerusakan yang terjadi. Sedangkan karakteristik atribut-atribut testabilitas, diantaranya adalah (1). *Operability* (2). *Obserability* (3). *Controlability* (4). *Decomposability* (5). *Simplicity* (6). *Stability*.

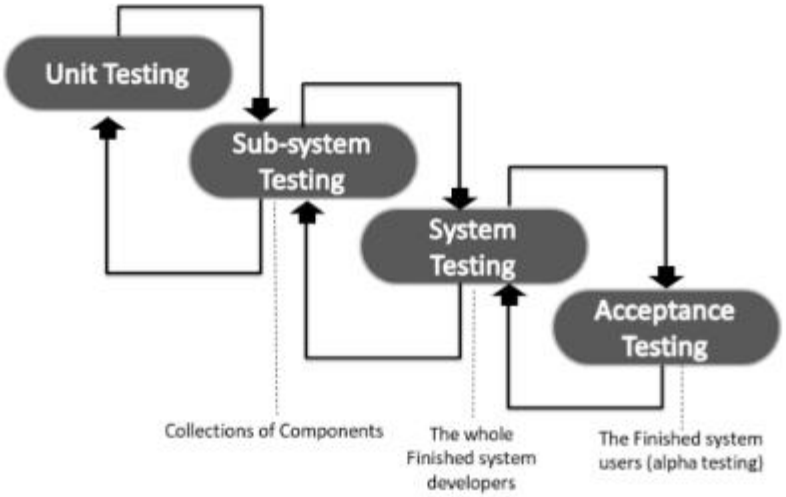
# 2 PROSES DAN TEKNIK PENGUJIAN

| | | |
|---|---|---|
| ***Tujuan intruksional umum*** | : | ***Memahami proses dan teknik pengujian perangkat lunak*** |
| ***Tujuan instruksional khusus*** | : | • ***Memahami dan mengerti proses pengujian perangkat lunak***<br>• ***Memahami teknik pengujian perangkat lunak*** |

Metodologi merupakan sekumpulan tahap atau tugas. Kebanyakan organisasi menggunakan suatu standar untuk pengembangan perangkat lunak (software) yang mendefinisikan suatu model siklus hidup (*life cycle model*), dan dibutuhkan tahap-tahap atau metodologi dalam pelaksanaannya.

Ide pembagian dalam bentuk tahapan digunakan pada semua metodologi perangkat lunak (*software*), dimana setiap tahapan memiliki keluaran berupa produk akhir yang merupakan hasil produk dan menjadi pertanda penyelesaian proses di tiap tahapan tersebut.

Adapun model siklus hidup perangkat lunak pada umumnya terdiri atas beberapa proses tahapan yang diawali dengan tahapan analisa digunakan untuk menentukan feasibilitas dan spesifikasi dari kebutuhan yang ada, kemudian proses berikutnya adalah tahapan disain digunakan untuk menentukan spesifikasi umum dan detil, dan




Gambar 3. Tahapan Pengujian

Adapun hal yang terkait dengan proses pengujian terdiri atas :

a. **Pengujian Kesalahan (*Defect Testing*)**

**T**eknik yang dilakukan untuk menemukan kesalahan dalam program dimana pengujian ini dikatakan sukses jika uji ini membuat program berlaku tidak normal/ tidak seperti seharusnya dan hasil pengujian harus menunjukkan bahwa kesalahan itu ada. Dalam pengujian ini terdiri dari berberapa metode yang digunakan untuk menyelesaikan suatu pengujian. Pengujian teknik ini yang sering dilakukan diantaranya adalah dengan metode *Black-box*, *Equivalence partiontining*, *Structural* (*white-box*) dan pengujian dengan metode *Path*.

b. **Pengujian Integritas (*Integration Testing*)**

Teknik ini digunakan untuk menguji dalam tahapan integritas dan keterhubungan. Adapun metode pengujian ini terdiri dari metode *Top-down*, *Bottom-Up* , *Interface* dan pengujian *Stress*.

c. **Pengujian Kasus (*Test Case*) dan Pengujian Data (*Test Data*)**

**Test Case**, spesifikasi dari masukan (*input*) untuk pengujian dan prediksi *output* dari sistem di tambah pernyataan-pernyataan tentang apa yang di-uji. Pada tahap pengujian ini, kiranya

program guna melihat sejauh mana kesesuaiannya dengan *output* yang di inginkan.

Membandingkan (*compare*) hasil yang diperoleh dengan apa yang telah dilakukan sebelumnya (prediksi awal dan inputan) akan memberikan suatu hasil yang menentukan apakah pengujian cukup dilakukan sampai batasan ini, dan apabila hasil tersebut "meragukan' (tidak sesuai) maka perlu dilakukan pengujian kembali.

## 2.2. Teknik Pengujian

### 2.2.1. Pengujian Alur ( *Path* )

Titik awal *path testing* adalah ***flow graph*** yang menunjukkan simpul (keputusan pada program) dan garis jalur (kendali aliran). Perintah berupa struktur kendali saja yang merupakan simpul pada *flow graph* (perintah ***input, output*** dan assigment (penandaan) tidak masuk dalam *flow graph*. **Tujuan pengujian** ini adalah "untuk memastikan bahwa tiap jalur dalam program dijalankan sedikitnya satu kali dengan menggunakan *test case*.

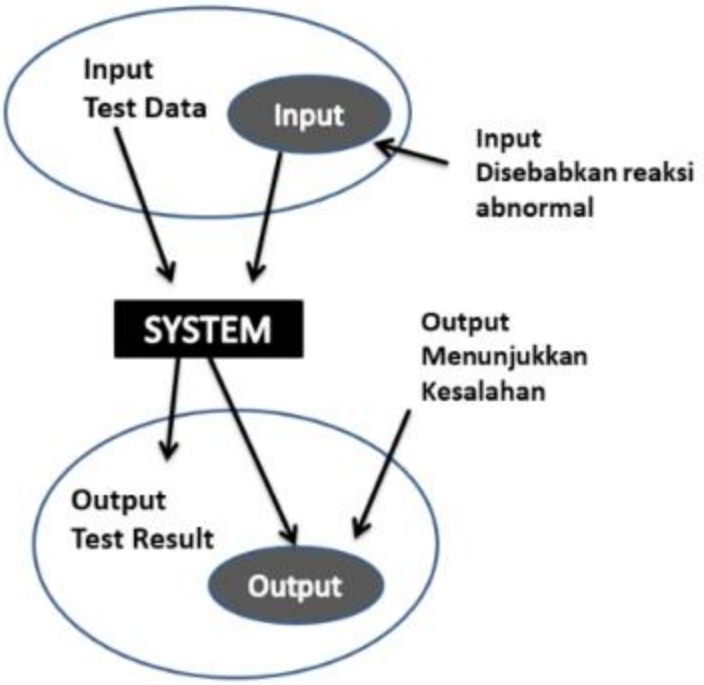
**Graf Aliran program** , Menggambarkan alur kontrol. Setiap cabang ditunjukkan oleh *path* yang terpisah dan *loop* ditunjukkan oleh panah berulang (*arrows looping*) kembali ke loop kondisi titik/simpul (*node*). Metode ini digunakan sebagai basis untuk menghitung ***cyclomatic complexity*** (**CC**), ( CC = jumlah edges – jumlah node + 2 ) dimana CC menyatakan jumlah test untuk menguji *control statements*. Hal ini menggambarkan aliran kendali. Tiap cabang ditunjukkan sebagai alur yang berbeda dan *loop* ditunjukkan dengan panah ***loop*** kembali ke simpul kondisi atas.

yang memiliki kemungkinan tertinggi di dalam pengungkapan kesalahan pada perangkat lunak. Untuk mencapai sasaran tersebut, digunakan berdasarkan atas empat (4) kategori yang berbeda dari tehnik desain *test case*, yaitu :

a). Pengujian *Black-box*

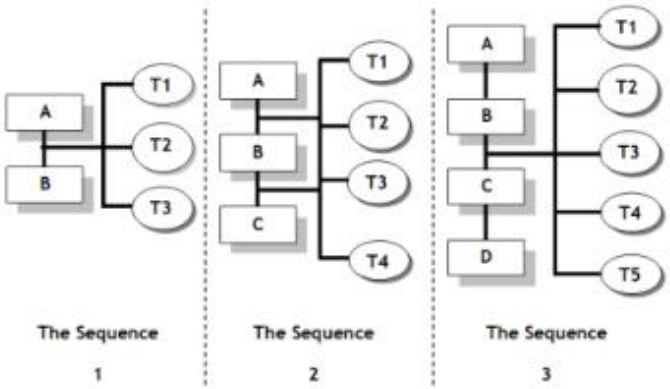
Test case ini bertujuan untuk menunjukkan fungsi perangkat lunak tentang cara beroperasinya, apakah pemasukan (*input*) data keluaran (*ouput*) telah berjalan sebagaimana yang diharapkan dan apakah informasi yang disimpan secara eksternal selalu dijaga kemutakhirannya.

Adapun dalam pengujian ini program yang diuji dianggap kotak hitam (*black-box*) dimana Kasus pengujian (*test case*) berdasarkan spesifikasi sistem sedangkan perencanaan dalam tahapan pengujian dapat dimulai dari awal proses perangkat lunak (*software*).




Gambar 6. Proses Pengujian Black-box

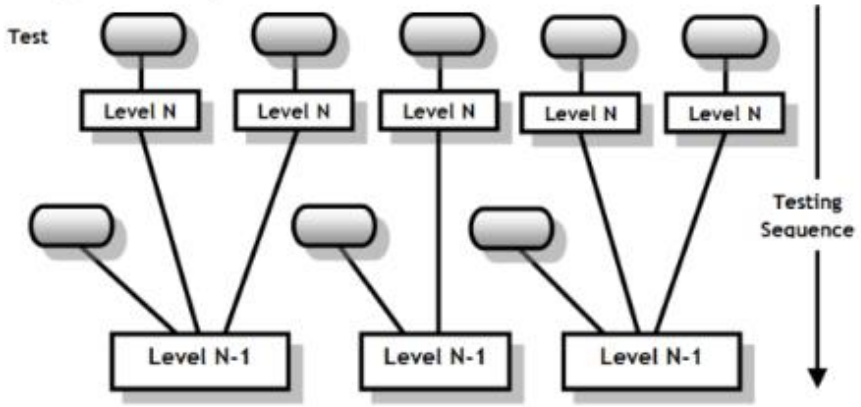
Test Integrasi Secara Bertahap




Gambar 8. Proses Pengujian Integrasi

c). Pengujian Integrasi Bottom-Up

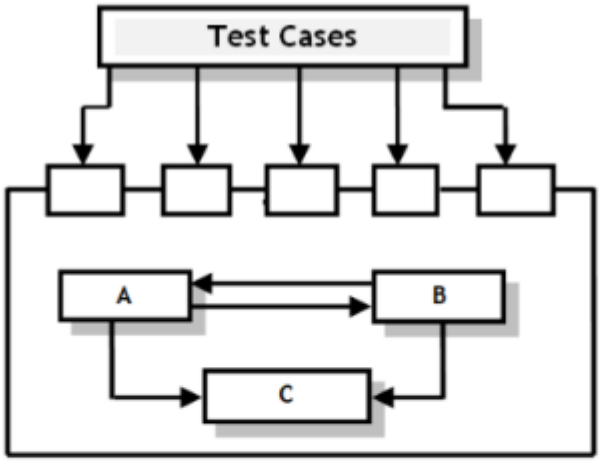
Pengujian Integrasi ini menguji dan menyatukan komponen-komponen mulai dari komponen tingkat yang paling rendah, kemudian bergerak ke tingkat di atasnya hingga semua komponen teruji.




Gambar 9. Proses Pengujian Integrasi Bottom-Up

d). Pengujian Integrasi Top-Down

Mulai dari komponen pada tingkat tertinggi, kemudian menguji komponen-komponen pada tingkat berikutnya.




Gambar 11. Proses Pengujian Interface – Test case

Tipe *Interface* antar Komponen

a). *Parameter Interfaces*
Data dilewatkan dari satu komponen ke komponen yang lain.

b). *Shared memory interfaces*
Pemakaian blok di memori secara bersama. Satu sub-sistem menyimpan data kemudian sub-sistem lain mengambil data tersebut

c). *Procedural interfaces*
Sub-sistem mengandung sekumpulan prosedur yang dipanggil oleh sub-sistem lain.

d). *Message passing interfaces*
Satu sub-sistem meminta ( *request* ) layanan ke sub-sistem lain dengan mengirim pesan (*message*).

**Rangkuman**

Tahapan yang sangat penting dalam pengujian adalah proses pengujian itu sendiri dimana kita harus paham benar dengan tahapan dan gambaran ke depan (prediksi) hasil yang akan didapatkan dari proses itu nantinya. Proses demi proses yang berlangsung tentunya sudah dirancang sedemikian rupa sehingga diharapkan tidak terjadi **"*error process*"**, dalam hal ini juga dituntut kemampuan logika berpikir dan tentunya juga kemampuan analisa

# WHITE-BOX TESTING 3

| | | |
|---|---|---|
| ***Tujuan intruksional umum*** | : | ***Memahami teknik White-Box Testing*** |
| ***Tujuan instruksional khusus*** | : | • ***Memahami dan mengerti proses pengujian white-box***<br>• ***Memahami teknik pengujian white-box*** |

Sering disebut juga dengan istilah *glass-box testing*, metode pengujian ini dilakukan untuk mengetahui isi dari produk, pengujian dapat dilakukan untuk memastikan bahwa semua operasi dapat berjalan sesuai spesifikasi dan semua komponen telah dicoba.

Sebuah metode perancangan kasus pengujian yang menggunakan struktur kontrol dari perancangan prosedural untuk menghasilkan kasus pengujian dapat dihasilkan kasus test yang menjamin semua alur pada sebuah modul sudah dicoba paling tidak sekali percobaan pengujian. Mencoba semua keputusan secara logika baik di sisi pengujian dengan benar (*true*) maupun pengujian dengan kesalahan (*false*) kemudian melakukan eksekusi semua percobaan perulangan (*loop*) dan juga melakukan percobaan pengujian dengan mengeksekusi struktur data internal untuk menjamin validitasnya (keakuratan dan kebenaran).

4. Syarat yang di lakukan dalam menjalankan strategi pengujian *white box*.
5. Mendefinisikan tentang seluruh alur-alur logika yang ada.
6. Membangun dan membuat suatu kasus yang akan di gunakan untuk tahap pengujian.
7. Hasil pengujian yang telah di dapatkan akan di lakukan eveluasi kembali.
8. Pengujian yang di lakukan haruslah secara menyeluruh.

Terdapat beberapa teknik dalam pengujian menggunakan metode *white box testing*. Berikut ini adalah teknik-teknik yang terdapat pada *white box testing*, yaitu : 1). Pengujian *Unit Testing*, ada tiga tahapan yaitu a). *Execution testing* (*statement coverage*, *branch coverage*, *path coverage*, *call coverage*, *loop coverage*, *condition coverage*). b). *Operation Testing* dan c). *Mutation Testing*. Kemudian 2). *Integration Testing*, ada tiga cara yaitu : a). *Top Down* b). *Bottom Up Approach* dan c). *Hybrid Approach*

### 3.1. *Execution Testing*

#### a. *Statement Coverage*

*Statement* dalam bahasa pemrograman tidak lain adalah merupakan sebuah baris kode. Pengujian *statement coverage* merupakan suatu teknik pengujian *white box* yang memastikan bahwa setiap *executable statement* dijalankan setidaknya sebanayak satu kali. Satu statement yang dijalankan hanya merupakan sebagian dari satu pengujian *test case*, sehingga tidak mungkin ada nya *test case* yang dialkukan dengan menjalankan statement yang sama. Dengan demikian hal ini membuktikan bahwa pada *statement coverage* memastikan bahwa semua pernyataan yang dijalankan tanpa efek samping. Sebagai suatu ilustrasi terdapat contoh *flowgraph* yang menggambarkan bahwa pengujian dengan teknik *statement coverage* dapat dilihat pada gambar berikut ini.

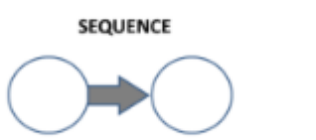
dijalankan dengan memvalidasi cabang yang benar (*true*) ataupun cabang yang salah (*false*). Berdasarkan pada *flowgraph* yang dijabarkan sebelumnya, terdapat enam (6) buah anak panah cabang. Sebagai contoh, suatu jalur eksekusi program melewati node-node a,b,d,h,k, jalur tersebut meninjau 2 dari 6 anak cabang yang ada, sehingga nilai branch yang didapat yaitu sebesar 33% (Jatnika & Irwan, 2010). Adapun rumus untuk menghitung nilai hasil coverage pada teknik pengujian pada *branch coverage* adalah sebagai berikut.

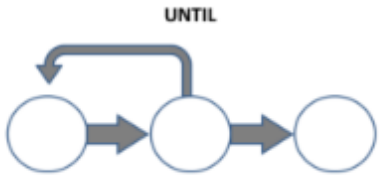
$$\textbf{Branch}\ \text{coverage} = \frac{\text{number of } \textbf{Branch} \text{ exercised}}{\text{total number of } \textbf{Branch}}$$

**c. Notasi Diagram Alir (*Path Graph Notation*)**

Notasi yang digunakan untuk menggambarkan jalur eksekusi adalah notasi diagram alir atau grafik program, yang menggunakan notasi lingkaran (simpul atau *node*) dan anak panah (*link* atau *edge*). Notasi ini menggambarkan aliran *control* logika yang digunakan dalam suatu bahasa pemrograman. Setiap representasi rancangan *procedural* dapat diterjemahkan kedalam *flowgraph*.

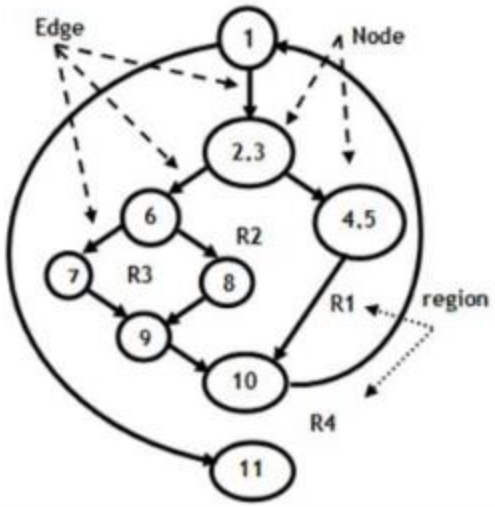
Notasi lingkaran disebut sebagai *flow graph node* yang digunakan untuk menggambarkan statement-statement yaitu satu atau lebih *statement* secara sekuensial yang dikelompokkan. Percabangan seleksi dari satu statement kedua pilihan statement (seleksi). Adapun tahap awal pada metode ini adalah dengan menggambarkan model alur logika (*flow graph*) yaitu :



**Komponen**

- Titik (*Nodes*) merupakan suatu pernyataan (atau sub program) yang akan ditinjau saat eksekusi program.
- Anak Panah (*Edges*) merupakan suatu jalur alur logika untuk menghubungkan satu pernyataan (atau sub program) dengan yang lainnya.
- Titik Cabang (*Branch nodes*) merupakan suatu titik-titik yang memiliki lebih dari satu anak panah keluaran (*output*).
- Anak Panah Cabang (*Brach Edges*) merupakan anak panah yang keluar dari suatu titik cabang.
- Jalur (*Path*) merupakan suatu jalur yang mungkin bergerak dari satu titik ke titik lainnya sejalan dengan keberadaan arah anak panah.
- *Region* merupakan suatu area yang dibatasi oleh *edges* dan *node*, area diluar *graph* juga dihitung sebagai *region*.

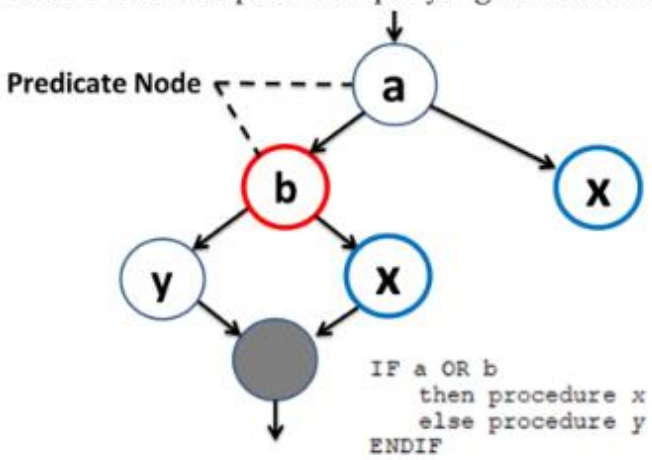



Gambar 14. Model Alur Logika komponen Region

*Test case* adalah suatu bentuk dokumen yang memiliki satu set data tes, prasyarat, hasil yang diharapkan dan *post conditions*, dikembangkan untuk skenario pengujian tertentu guna memverifikasi kepatuhan terhadap persyaratan tertenu (Suhartono, 2016).

```
ocedure: sort
  do while records remain read records;
    if record field 1= 0
       then process record;
         store in buffer;
       elseif record field 2 = 0
         then reset counter;
          else process record;
          store in file;
       endif
    endif
  enddo
end.
```

- *Predicate Node* merupakan simpul yang berisikan kondisi

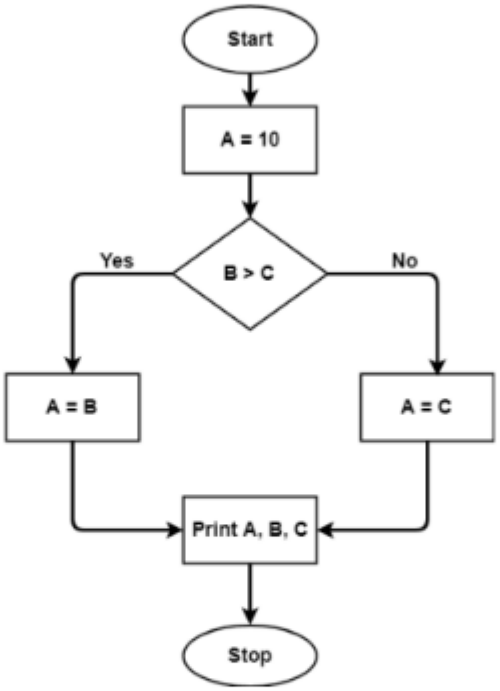



Gambar 16. Kondisional pada *Predicate Node*

Berikut menunjukkan contoh bagian PDL (*Program Design Languange*) dan *flow graph*.

**3.2.** *Cyclomatic Complexity*

*Cyclomatic Complexity* Adalah pengukuran *software* yang memberikan pengukuran kuantitatif dari kompleksitas logika program. Pada konteks metode ini, nilai yang dihitung bagi *cyclomatic complexity* menentukan jumlah jalur-jalur yang independen dalam kumpulan basis suatu program dan memberikan jumlah tes minimal yang harus dilakukan untuk memastikan bahwa semua pernyataan telah dieksekusi sekurangnya satu kali.




Gambar 18. Contoh Alur Logika *Cyclomatic Complexity*

**3.2.** *Cyclomatic Complexity*

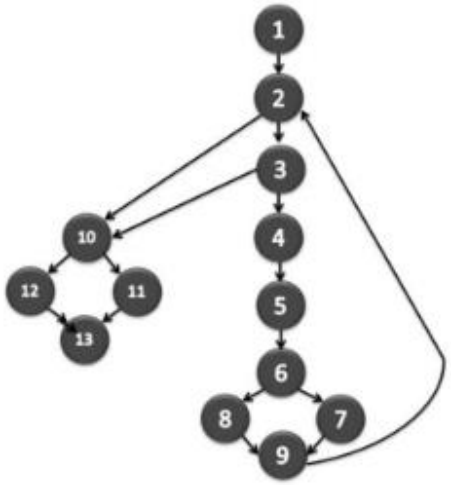
Jalur independen adalah dimana setiap jalur pada program yang memperlihatkan satu kelompok baru dari pernyataan proses atau kondisi baru. Jalur merupakan ukuran kuantitatif kompleksitas secara lojik pada sebuah program. Dimana nilai yang akan dihasilkan dari metode *cyclomatic complexity* akan mendefinisikan sejumlah alur secara independen dalam sebuah program dan juga batasan secara maksimal jumlah kasus pengujian (*test*) yang harus dilakukan

```
PROCEDURE average;

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

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

    TYPE value[1:100] 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.input 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;
    ENDDO
    IF total.valid > 0
       THEN average = sum / total.valid;
       ELSE average = -999;
    ENDIF
END average
```




Gambar 19. *Coding* dan *Flow* Logika program *Test Case*

Adapun contoh pengujian *white box* dengan menggunakan grafik aliran, kita dapat menghitung jumlah jalur *independen* melalui kode dengan menggunakan metrik disebut nomor *cyclomatic* (McCabe, 1976), yang berdasarkan pada teori grafik. Cara termudah untuk menghitung jumlah siklomatik adalah dengan menghitung jumlah *conditional* /predikat (*symbol lambing diamond*) dan kemudian di tambahkan satu (1). Di misalkan terdapat ada lima *conditional*. Oleh karena itu, jumlah *cyclomatic* adalah 6, dan memiliki enam jalur independen melalui kode. Jadi dapat di hitung , yaitu :

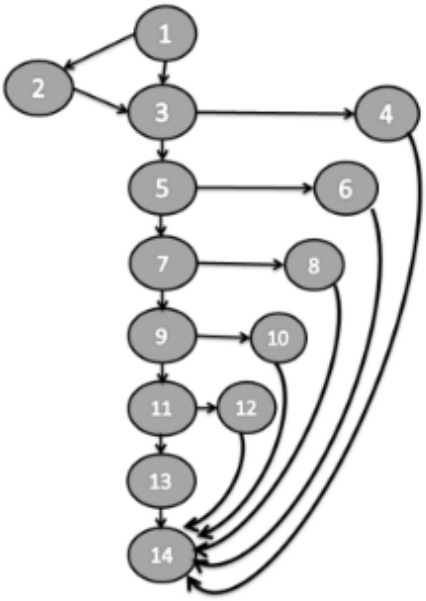
1. 1-2-3-4-5-10 (properti yang dimiliki oleh orang lain, tidak mempunyai uang untuk sewa)
2. 1-2-3-4-6-10 (properti yang dimiliki oleh orang lain, membayar sewa)
3. 1-2-3-10 (properti yang dimiliki oleh pemain)
4. 1-2-7-10 (properti yang tersedia, tidak memiliki cukup uang)
5. 1-2-7-8-10 (properti yang tersedia, punya uang, tidak ingin membelinya)
6. 1-2-7-8-9-10 (properti yang tersedia, punya uang, dan membelinya)

Kasus pengujian ini di gunakan untuk memastikan bahwa setiap jalur yang akan diuji setidaknya sekali dimana jumlah siklomatik adalah batas bawah pada jumlah kasus uji yang akan di tulis. Uji kasus yang ditentukan dengan cara ini adalah yang akan gunakan dalam pengujian *basis patch*. Sedangkan contoh pengujian *white box* dengan menggunakan metode Algoritma, dapat dilakukan dengan tahapan sebagai berikut :

**Step 1**:

Pada prosedur di berikut ini menunjukkan bagaimana laporan algoritma dipetakan ke dalam node grafik, dengan memberikan penomoran angka di sebelah kiri.

Dibawah ini adalah *flowchart* dari contoh program diatas :




Gambar 20. *Coding* node penomoran dan *Flow* program

**Step 2**:

Menentukan kompleksitas *cyclomatic* dari grafik aliran.

V(G) = E – N + 2
= 19 – 14 + 2
= 7

**Keterangan:**

E : Jumlah Busur atau Link

N : Jumlah Simpul

Ini menjelaskan bahwa batas atas pada ukuran basis set. Artinya, memberikan jumlah jalur independen yang perlu kita cari.

**Step 3**:

Menentukan dasar jalur independen

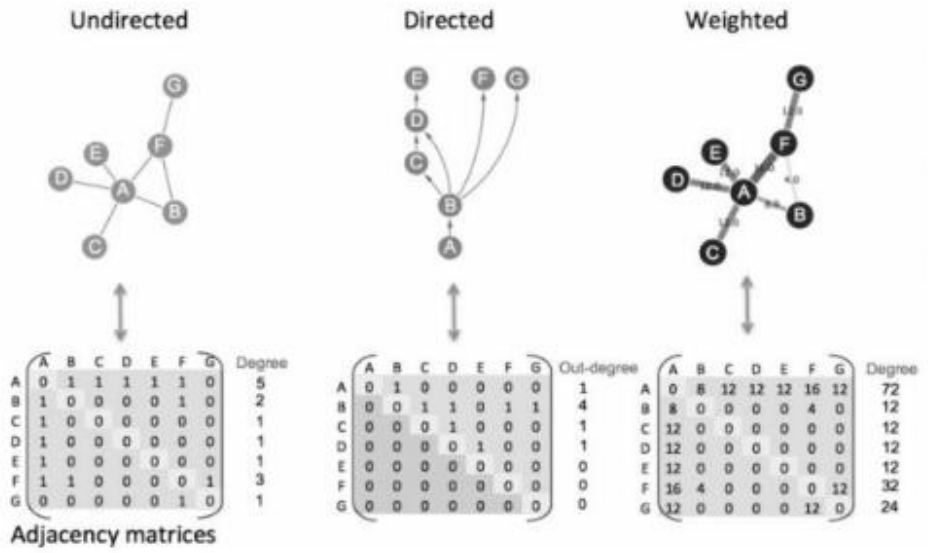
**Path 1**: 1 – 2 – 3 – 5 – 7 – 9 – 11 – 13 – 14

**Path 2**: 1 – 3 – 4 – 14

**Path 3**: 1 – 3 – 5 – 6 – 14

Adapun matrik yang berbentuk segi empat sama sisi, dimana setiap jumlah baris dan kolom sama dengan jumlah titik, dan di identifikasi pada baris dan kolom yang sama dengan identifikasi *node*, serta isi data adalah adanya keberadaan penghubung antar *node* (*edges*). Beberapa properti yang dapat ditambahkan sebagai nilai pembobotan pada penghubung antar node di dalam graph matrix, sebagai berikut:

1. Kemungkinan jalur (*edge*) akan dilalui / dieksekusi.
2. Waktu proses yang diharapkan pada jalur selama proses transfer dilakukan.
3. Memori yang dibutuhkan selama proses transfer dilakukan pada jalur.
4. Sumberdaya (*resources*) yang dibutuhkan selama proses transfer dilakukan pada jalur.



Adapun tahapan alur logika pengujian setiap spesifikasi flow adalah sebagai berikut :

***Simple loop :***

- Skip seluruh loop
- Hanya satu kali melewati loop
- Dua kali melewati loop
- Jika n adalah jumlah pass maksimum yang di perbolehkan, maka lewati loop sebanyak m kali dimana m < n
- Lewati loop sebanyak n – 1, n, n + 1

***Nested loop :***

- Mulai dari loop terdalam. Semua loop lain diset dengan nilai minimum.
- Lakukan *simple loop test* untuk loop terdalam dengan nilai parameter iterasi loop lain diset menjadi minimum. Tambahkan nilai-nilai diluar range.

***Concatenated loop :***

- Jika semua loop tidak saling bergantung, maka lakukan test simple loop untuk masing-masing loop.
- *Jika kedua loop saling berhubungan dan nilai counter pada loop 1 digunakan sebagai nilai awal counter loop 2, maka lakukan test nested loop.*

***Unstructured loop :***

- Jika mungkin, loop ini harus dirancang ulang.

**f. *Condition***

Pengujian bersyarat Ini adalah pengujian dengan melakukan pendekatan untuk merancang kasus uji untuk menjalankan kondisi logis yang terdapat dalam modul program aplikasi. Pengujian ini berfokus untuk mengetes setiap kondisi yang terdapat dalam program, adapun kesalahan yang mungkin terjadi dalam melakukan pengujian ini pada umumnya adalah :

- *Boolean operator error*
- *Boolean variabel error*
- *Boolean parenthesis error*

tidaknya suatu item yang akan digunakan, biasanya dilakukan uji signifikansi koefisien korelasi pada taraf signifikansi 0,05, artinya suatu item dianggap valid jika berkorelasi signifikan terhadap skor total.

Untuk melakukan pengujian secara validitas (contoh) penggunaan aplikasi program SPSS. Teknik pengujian ini sering digunakan dalam pengujian validitas adalah dengan metode korelasi *Bivariate Pearson* (Produk Momen Pearson). Hasil analisa ini adalah dengan cara melakukan korelasi setiap nilai item dengan nilai total. Nilair total adalah dengan penjumlahan secara keseluruhan item. Item pertanyaan yang berkorelasi secara signifikan dengan nilai total menunjukkan item tersebut dapat memberikan dukungan dalam mengungkap apa yang ingin diungkan secara validitas. Jika nilai r hitung ≥ r pada tabel (uji 2 sisi dengan sig. 0,05) maka item pertanyaan berkorelasi secara signifikan terhadap nilai secara total (dinyatakan benar). Adapaun tahapan dalam pengujian validitas ini yaitu :

1. Buat skor total masing-masing variabel (Tabel perhitungan skor).
2. Klik Analyze -> Correlate -> Bivariate (Gambar/Output SPSS).
3. Masukan seluruh item variabel x ke Variabels
4. Cek list Pearson ; Two Tailed ; Flag
5. Klik Ok

Tabel rangkuman hasil uji validitas dari variabel tersebut dapat dilihat sebagai berikut :

| No Soal | r hitung | r tabel | Keterangan |
|---|---|---|---|
| 1 | 0.915 | 0.576 | Valid |
| 2 | 0.904 | 0.576 | Valid |
| 3 | 0.910 | 0.576 | Valid |
| 4 | 0.883 | 0.576 | Valid |
| 5 | 0.922 | 0.576 | Valid |
| 6 | 0.956 | 0.576 | Valid |
| 7 | 0.931 | 0.576 | Valid |
| 8 | 0.963 | 0.576 | Valid |
| 9 | 0.941 | 0.576 | Valid |
| 10 | 0.879 | 0.576 | Valid |
| 11 | 0.920 | 0.576 | Valid |

Tabel 4. Rangkuman Hasil Uji Validitas

Kesepakatan secara umum reliabilitas yang dianggap sudah cukup memuaskan jika ≥ 0.700.

Pengujian reliabilitas instrumen dengan menggunakan rumus Alpha Cronbach (instrumen berbentuk angket dan skala bertingkat). Adapun formula "Alpha Cronbach" sebagai berikut :

$$r_{11} = \left(\frac{n}{n-1}\right)\left(1 - \frac{\sum \sigma_t^2}{\sigma_t^2}\right)$$

Keterangan :

r11 = kehandalan yang di cari
n = jumlah item pertanyaan yang di uji
$\sum \alpha t^2$ = jumlah variasi nilai setiap item
$\alpha t^2$ = total varian

d. Branch testing

Untuk setiap pencabangan C, cabang true dan false dari C dan setiap kondisi di C harus dieksekusi paling sedikit satu kali Domain testing : memerlukan tiga atau empat test untuk sebuah rational expression

**E1 <relational-operator> E2**

Maka diperlukan tiga test dengan nilai E1 lebih kecil, sama dengan dan lebih besar dari E2. **BRO** (*branch and relational operator*) testing : menggunakan strategi conditions constraint

**C1: B1 & B2**

dimana B1 dan B2 adalah variabel Boolean. *Condition constraint* untuk C1 adalah (D1,D2), dimana setiap D1 dan D2 adalah "t" atau "f" . BRO testing mengharuskan constraint set {(t,t),(f,t),(t,f)} termasuk dalam ekseskusi **C1**

**C2: B1 & (E2 = E4)**

dimana B1 adalah ekspresi Boolean dan E2 dan E4 adalah ekspresi aritmatik. ***Condition constraint*** untuk C1 adalah (D1,D2), dimana setiap D1 adalah "t" atau "f" dan D2 adalah >, =, atau <. *Constraint set* untuk **C2** adalah **{(t,=),(f,=),(t,<),(t,>)}**

Memilih alur test program berdasarkan lokasi definisi dan menggunakan variabel dalam program. Asumsi bahwa setiap kalimat dalam program diberi nomor dan tiap fungsi tidak ***mengubah parameternya (variabel global)***Untuk setiap kalinat dengan S sebagai nomornya :

DEF(S) = {X | kalimat S mencakup definisi X}
USE(S) = {X | kalimat S mencakup penggunaan X}

Definisi variabel X pada kalimat S diakatakan hidup pada kalimat S' jika ada alur dari kalimat S ke kalimat S' tanpa definisi lain dari X. Definition-use chain (DU chain) dari variabel X dalam bentuk [X,S,S'] dimana S dan S' adalah nomor kalimat, X dalam DEF(S) dan USE(S') & definisi X dalam kalimat S hidup pada kalimat S' Setiap DU chain harus dijalankan minimal satu kali.

```
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;
end proc;
```

Asumsi : varabel X didefinisikan di kalimat terakhir blok B1, B2, B3, B4, dan B5, dan digunakan di kalimat pertama blok B2, B3, B4, B5, dan B6

Eksekusi alur terpendek dari setiap Bi,
$0<i<=5$, dan
Bj, $1<j<=6$

Gambar 23. Contoh Alur Logika *Branch Testing*

# PENGUJIAN CLEAN-ROOM 4

| | | |
|---|---|---|
| ***Tujuan intruksional umum*** | : | ***Memahami teknik pengujian clean-room*** |
| ***Tujuan instruksional khusus*** | : | • ***Memahami dan mengerti tujuan pengujian clean-room***<br>• ***Memahami dan dapat membuat laporan dengan model pengujian clean-room*** |

Strategi dan taktik pengujian *cleanroom* secara mendasar berbeda dengan pendekatan pengujian konvensional. Metode konvensional menarik serangkaian pengujian kasus (*test case*) untuk mengunakap kesalahan (*error*) yang terjadi dalam pengkodean dan rancangan (*design*). Tujuan metode pengujian *clean-room* ini adalah untuk melakukan validasi persyaratan pada perangkat lunak (*software*) dengan memperlihatkan (menunjukkan laporan) bahwa suatu contoh ditunjukkan secara statistik dari kasus pemakaian aplikasi telah dilakukan dan dieksekusi serta menunjukkan hasil yang sesuai dengan target yang telah di tetapkan.

## 4.1. Pengujian Statistik

Pemakai program komputer jarang merasa perlu dipahami secara detail teknik dari perancangan (design) adapun pola program

probabilitas dari setiap stimulus. Untuk membuat seleksi lebih mudah, probabilitas itu dipetakan ke dalam interval yang diberi nomor antara 1 dan 99 [LIN94].

Guna memunculkan urutan test case pemakaian yangs sesuai dengan format distribusi probabilitas pemakaian, sederetan nomor acak dimunculkan antara angka 1 sampai dengan 99 yang berhubungan dengan interval pada distribusi probabilitas (lihat tabel 5). dengan demikian urutan kasus penggunaannya ditentukan secara acak (*random*) namun demikian sesuai dengan probabilitas kejadian secara stimulus yang telah disesuaikan. Sebagai contoh,bahwa diasumsikan dalam suatu urutan bilangan yang dilakukan secara acak (*random*) adalah sebagai berikut :

Tabel 5. Contoh Interval Distribusi Probabilitas

**13-94-22-24-45-56**
**81-19-31-69-45-9**
**38-21-52-84-86-97**

Dengan memilih stimulus yang sesuai berdasarkan interval distribusi yang diperlihatkan pada tabel 5, di dapat *use case* sebagai berikut

Table 6. Contoh Interval Distribusi di dapat use case

**AD-T-AD-AD-ZS**
**T-AD-AD-AD-Q-AD-AD**
**AD-AD-ZS-T-T-PA**

Tim pengujian yang mengeksekusi *use case* tersebut (dan lainnya) serta memverifikasi tingkah laku perangkat lunak (*software*) terhadap spesifikasi sistem. Waktu untuk melakukan kegiatan pengujian direkam sehingga akan didapatkan waktu interval dapat ditentukan. Dengan menggunakan waktu interval ini tim sertifikasi dapat menghitung *mean-time-failure* (MTF). Bila sederetan panjang pengujian dilakukan tanpa kegagalan,(*failure*) maka *mean-time-failure* (MTF) dapat dikategorikan rendah dan realibilitas perangkat lunak (*software*) dapat diasumsikan tinggi.

Tabel 6. Stimulus Program dan distribusi nilai probabilitas

| Stimulus program | Distribusi | Interval |
|---|---|---|
| Arm/disam (AD) | 50% | 1-49 |
| Zone set (ZS) | 15% | 50-64 |
| Query (Q) | 15% | 64-78 |
| Test(T) | 15% | 79-94 |
| Panic alam (PA) | 5% | 95-99 |

**Rangkuman**

Pengujian menggunakan statistik merupakan pengujian perangkat lunak yang berlaku sama dengan yang dimaksudkan *user* dimana tim pengujian harus dapat menentukan distribusi kemungkinan digunakannya aplikasi perangkat lunak tersebut.

Rekayasa perangkat lunak (*software*) metode *cleanroom* yang digunakan memungkinkan adanya pengembangan secara inkremen pada perangkat lunak (*software*) yangselanjutnya akan mengatur interaksi pemakai.

**Latihan 4**

**Jawablah soal berikut ini dengan singkat dan jelas.**

1. Jelaskan yang dimaksud dengan pengujian *cleanroom*
2. Jelaskan pengujian menggunakan statistik
3. Untuk menentukan *test case*, buatlah suatu kasus pengujian (contoh) yang menggunakan metode pengujian *cleanroom*.

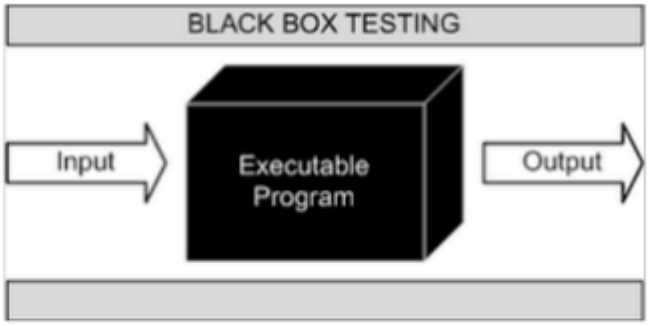
# BLACK-BOX TESTING 5

| | | |
|---|---|---|
| ***Tujuan intruksional umum*** | : | ***Memahami pengujian Black-Box*** |
| ***Tujuan instruksional khusus*** | : | • ***Memahami dan mengerti tujuan pengujian black-box***<br>• ***Memahami dan dapat meneyelesaikan kasus dengan meggunakan pengujian black-box*** |

Pengujian *Black-Box* merupakan metode pengujian yang dilakukan dengan mengamati hasil eksekusi melalui data yang uji dan cek software fungsionalitasnya. Jadi analogi seperti ini yang kita lihat sebagai suatu seperti kotak hitam, kita dapat melihat hanya penampilan luar saja, tanpa mengetahui apa di balik bungkus hitam tersebut. Sama sepertihalnya pengujian pada kotak hitam, melakukan evaluasi hanya pada penampilan eksternal (antarmuka/ interface), fungsional tanpa mengetahui apa yang sebenarnya terjadi dalam pada internalnya/codingan.

Tujuan dilakukannya pengujian metode *black-box* ini adalah untuk mendapatkan kesalahan (*error*)/kegagalan (*fail*) dalam operasi tingkat tinggi, yang mencakup kemampuan dari aplikasi perangkat lunak,(*software*) secara operasional/tata laksana, skenario pemakai.

Adapun proses pengujian *black-box* digambarkan sebagai berikut :




Gambar 24. Model Pengujian *black-box*

Pendekatan pengujian dimana program dianggap sebagai suatu '*black-box*' ("kotak hitam"), yaitu :

- *Program test case* berbasis spesifikasi
- *Test planning* dapat dimulai sejak awal proses pengembangan sistem.
- Menghasilkan himpunan kondisi input yang akan mencoba semua kebutuhan fungsional sebuah program
- Pengujian ini berusaha untuk menemukan adanya kesalahan dalam kategori :
  - Fungsi yang tidak benar atau hilang
  - Kesalahan antarmuka (*interface*)
  - Kesalahan dalam struktur data atau akses database eksternal
  - Kesalahan kinerja
  - Kesalahan inisialisasi dan terminasi

## 5.2. Pengujian Modul *Black-Box*

Pengujian ini juga disebut dengan Partisi Ekivalensi (*Equivalence Partitions*), dimana *Input* data dan *output* hasil terdapat di klas yang berbeda yang sesuai dengan klas inputnya. Masing-masing klas *equivalensi partitions* diproses dimana program akan memproses anggota klas-klas tersebut secara equivalen. Test cases dipilih dari masing-masing paritisi.

equivalence class 1 : data < 1
equivalence class 2 : 1 < data < n
equivalence class 3 : n < data

***Boundary Value Analysis***

Salah satu teknik yang baik untuk menemukan kesalahan adalah boundary value analysis, yaitu pengujian pada data yang berada pada perbatasan (ambang jangkauan ditambah/dikurang 1), sehingga diperoleh 7 test data.

Contoh : suatu program dengan jangkauan data R1...R2, dimana R1=1 dan R2=5000, maka test data yang harus dilakukan meliputi :

Test case 1 : R1 -1 = 0
Test case 2 : R1 = 1
Test case 3 : R1+1 = 2
Test case 4 : R1..R2 = 300
Test case 5 : R2-1 = 4999
Test case 6 : R2 = 5000
Test case 7 : R2+1 = 5001

- ***Functional Testing***

Menguji apakah fungsi dari modul telah sesuai dengan spesifikasi yang diharapkan.

### 5.3. *Data-based Black-Box Test Generation*

Apa yang dimaksud dengan "*error/fault/bug*" terkadang ungkapan ini sering diutarakan tatkala terjadi kendala-kendala (kesalahan) yang terjadi pada perangkat lunak (*software*).

Kesalahan (*Failure*), pada umumnya merupakan manisfestasi (ungkapan) yang di utarakan apabila terjadi kesalahan (dalam dunia *computer-observasi* terhadap ruang lingkup perangkat lunak. Terkadang juga hal tersebut "ada sesuatu yang tidak benar" dalam pengelolaan *software* (terkait dengan hal sudut pandang *user* terhadap kebutuhan yang digunakan) di tunjukkan pada tabel berikut ini.

**Rangkuman**

Pengujian *Black-Box* merupakan metode pengujian yang dilakukan dengan mengamati hasil eksekusi melalui data yang uji dan cek perangkat lunak secara fungsional. Adapun tujuan dilakukannya pengujian metode *black-box* ini adalah untuk mendapatkan kesalahan (*error*)/kegagalan (*fail*) dalam operasi tingkat tinggi, yang mencakup kemampuan dari aplikasi perangkat lunak,(*software*) secara operasional/tata laksana, skenario pemakai.

**Latihan 3**

**Jawablah soal berikut ini dengan singkat dan jelas.**

1. Jelaskan yang dimaksud dengan pengujian black-box.
2. Jelaskan Tujuan Pengujian black-box
3. Jelaskan Keuntungan Pengujian black-box.
4. Jelaskan Kerugian Pengujian black-box
5. Berikan contoh Kasus Pengujian dengan status “error”.

# IMPLEMENTASI 6

| | | |
|---|---|---|
| ***Tujuan intruksional umum*** | : | ***Memahami implementasi pengujian*** |
| ***Tujuan instruksional khusus*** | : | • ***Memahami dan mengerti tujuan implementasi pengujian***<br>• ***Memahami bahwa implementasi pengujian diterima*** |

Pengujian merupakan kemampuan penting yang sering diabaikan dalam kursus akademik dalam pemrograman komputer. Pengujian harus di tunda sampai setelah keseluruhan program selesai dibuat. Terdapat tiga tingkatan pengujian harus dilakukan: ***stub testing***, ***unit testing*** (***program testing***) dan ***system testing***. Hanya karena sebuah program tunggal telah bekerja dengan baik, tidak berarti program tersebut akan bekerja dengan baik dengan program lain (kasus lain).

Set terintegrasi dari program harus diuji terlebih dahulu untuk memastikan bahwa program tersebut dapat diterima dengan baik sebagai ***input*** dan merupakan ***output*** dari program yang lain. Setelah pengujian sistem selesai dan dinilai berhasil, kita dapat melangkah ke implementasi sistem.

Apa yang belum kita lakukan ? sistem baru selalu menggambarkan sebuah perjalanan sebuah bisnis yang dijalankan, berdasarkan itu "**analis**" harus melakukan transisi yang halus dari sitem lama ke sistem baru dan membantu pengguna untuk

Iterasi ini akan dilanjutkan secara simultan sampai pengujian sistem dianggap berhasil.

## 6.2. Menyiapkan Rencana Konversi

Setelah pengujian sistem berhasil (tercapai), kita akan memulai persiapan guna menempatkan sistem yang baru tersebut ke dalam operasi. Dengan menggunakan spesifikasi desain untuk sistem baru, Analis sistem akan mengembangkan sebuah rencana detail konversi. Rencana ini akan mengidentifikasi database yang harus diinstal, pelatihan penguna akhir (***end-user***) dan dokumentasi yang harus dikembangkan, dan sebuah strategi yang dapat mengkonversi sistem lama ke sistem baru.

Banyak organisasi yang meminta agar seluruh rencana proyek dipresentasikan secara formal kepada ***steering body*** (***steering committee***) untuk persetujuan akhir. Rencana konversi dapat melibatkan salah satu dari strategi instalasi yang biasa digunakan; diantaranya :

a. ***Abrupt cut-over***, penanggalan tertentu (umumnya tanggal yang bertepatan dengan satu periode resmi bisnis perbulan, tiga bulan, atau tahun fiskal). *Abrupt cut-over* dirasa perlu jika, misalnya : sebuah peraturan pemerintah atau kebijakan bisnis menjadi efektif pada tanggal tertentu dan sistem tidak dapat digunakan sebelum tanggal tersebut.

b. ***Parallel conversion***, Pada pendekatan ini, baik sistem alam atau sistem baru dioperasikan untuk beberapa periode (berjalan bersamaan), Hal ini dilakukan untuk memastikan seluruh masalah utama pada sistem baru telah diatasi sebelum sistem lama dibuang (tidak dipakai lagi).
   **Perlu diperhatikan**, bahwa menjalankan dua edisi dari sistem yang sama pada computer dapat menyebabkan terjadinya permintaan ***resource*** komputasi yang tidak beralasan (tambahan), namun mungkin dilakukan hanya jika sistem lama sebagian besar dilakukan secara manual.

a. ***System performance***.
Apakah trouhgput dan waktu respon untuk pemprosesan sudah tepat untuk memenuhi beban kerja pemrosesan normal ? Jika tidak, bebrapa program "harus di buat ulang" untuk meningkatkan efisiensi atau hardware pemrosesan harus diganti (up-grade) untuk mengatasi beban kerja tambahan.

b. ***Peak workload processing performance***.
Apakah sistem dapat mengatasi beban kerja selama periode pemrosesan *puncak*? Jika tidak, perbaiki *hardware* dan (atau *software*) untuk meningkatkan efisiensi (jadwal ulang pemrosesan). Jadi, pertimbangkan untuk melakukan tindakan-tindakan pemrosesan yang tidak begitu kritis selama periode nonpuncak.

c. ***Human engineering test***.
Apakah sistem, sama mudahnya untuk di pelajari dan digunakan seperti yang ditunda sampai sistem dipindahkan ke dalam operasi?

d. ***Methods dan procedures test***.
Selama konversi, metode dan prosedur untuk sistem baru akan mengalami pengujian riil pertama. Metode dan prosedur mungkin harus dimodifikasi jika mereka terbukti janggal dan tidak efisien menurut pengguna akhir.

e. ***Backup and recovery testing***.
Seluruh prosedur backup dan recovery harus diuji. Pengujian meliputi mensimulasi masalah kehilangan data dan menguji waktu yang dibutuhkan untuk mengatasi masalah tersebut.Membandingkan data sebelum dan sesudah perbaikan harus dilakukan untuk memastikan bahwa data benar-benar telah diperbaiki. Jangan menunggu samapai muncul masalah pertama untuk menemukan sebuah kesalahan pada prosedur recovery.

g. ***Audit testing***, menyatakan bahwa Pengujian yang dilakukan untuk memastikan sistem bebas dari kesalahan dan siap untuk ditempatkan pada operasi. Tidak semua organisasi

Dengan melakukan tahap dokumentasi yang tepat pada sistem baru, analis sistem akan memberikan bentuk dokumentasi hasil referensi dan juga akan melakukan kegiatan pelatihan yang dibutuhkan untuk pengguna sistem agar kiranya nanti dapat menyesuaikan diri sebelum menggunakan sistem baru dengan tepat. Hasil utama dari tugas ini adalah pelatihan dan dokumentasi pengguna.

Pada tabel 9 diatas adalah merupakan skema khusus untuk petunjuk manual) pelatihan. Sebagai contoh kasus dimana Anda di minta untuk menulis untuk orang lain seperti Anda mengharuskan mereka menulis untuk Anda.(Studi kasus, contoh) "Anda bukanlah pakar bisnis ", janganlah mengharapkan kepada orang lain untuk menjadi seorang ahli teknik dimana situasi ini yang mungkin akan terjadi dan untuk itu buatlah prosedur yang tepat untuk situasi tersebut dimana setiap aktivitas yang terjadi hendaknya dapat di dokumentasikan dengan baik dan benar.

## 6.5. Beralih ke Sistem Baru

Konversi ke sistem baru dari sistem lama adalah merupakan kejadian yang sangat penting. Setelah konversi, kepemilikan sistem secara resmi berpindah dari analis dan programmer kepada pengguna akhir. Ingat bahwa rencana konversi meliputi konversi meliputi strategi instalasi yang detail yang harus diikuti untuk beralih dari sistem informasi produksi yang lama ke sistem informasi produksi yang baru. Pengguna sistem akan memberikan umpan balik (*feed-back*) yang berharga tentang penggunaan actual dari sistem baru tersebut. Mereka akan menjadi sumber dari mayoritas umpan balik yang digunakan untuk mengukur penerimaan sistem.

Pada beberapa kasus, umpan balik tersebut dapat mendorong tindakan untuk mengkoreksi kelemahan-kelemahan yang telah diidentifikasi. Bagaimanapun, umpan balik tersebut akan digunakan untuk membantu "benchmark" proyek sistem baru yang sedang dijalankan.

Input kunci aktivitas ini adalah rencana konversi yang telah dibuat pada tugas fase implementasi sebelumnya.

# PROYEK PERANGKAT LUNAK 7

| | | |
|---|---|---|
| ***Tujuan intruksional umum*** | : | ***Memahami Proyek Perangkat Lunak*** |
| ***Tujuan instruksional khusus*** | : | • ***Memahami dan mengerti tujuan proyek perangkat lunak***<br>• ***Memahami kegiatan di dalam mempertimbangkan pembentukan dalam tim pengujian perangkat lunak*** |

Manajemen proyek perangkat lunak (software) yang efektif berfokus pada urutan (1) *People* , elemen terpentingdari suksesnya proyek. (2) *Problem* (*product*) , software yang dikembangkan (3) *Process* , suatu kerangka kerja dari suatu aktifitas dan kumpulan tugas untuk mengembangkan perangkat lunak (4) *Project* (tambahan), penggabungan semua kerja untuk membuat produk menjadi kenyataan.

Rekayasa Perangkat Lunak (*Software Engineering* /SEI) telah mengembangkan suatu model kematangan kemampuan manajemen manusia (*people management Capability Manurity Model* (PM-CMM) untuk mempertinggi kesiapan organisasi Perangkat Lunak dalam membuat aplikasi yang semakin kompleks sehingga menarik, menumbuhkan, memotivikasi, menyebarkan dan memelihara bakat

beberapa alternatif untuk menentukan sumber daya manusia tersebut :

- n orang mengerjakan tugas fungsional berbeda sebanyak m dengan sedikit kombinasi kerja & koordinasi tanggung jawab manajer proyek.
- n orang mengerjakan tugas fungsionalberbeda sebanyak m (m<n), seorang pemimpin tim *ad hoc* dapat dipilih, koordinasi bertanggung jawab manajer PL.
- n orang diatur di dalam tim, setiap orang mengerjakan >= 1 tugas fungsional, yang bekerja pada sebuah proyek, koordinasi di kontrol oleh tim itu sendiri dan oleh manajer proyek PL (sistem ini paling produktif).

Mantei, mengusulkan tiga hal organisasi dalam tim, yaitu :

1. Demokrasi Terdesentralisasi (DD)
   Tidak memiliki pimpinan permanen dan koordinator dipilih untuk tugas pendek bila tugas berbeda maka pimpinan berbeda. Keputusan diambil oleh konsensus kelompok_dan komunikasi secara horizontal.
2. Terkontrol Terdesentralisasi (CD)
   Tim memiliki pimpinan tertentu dan memiliki pimpinan sekunder untuk sub-sub masalah. Pemecahan masalah merupakan aktivitas dari kelompok dan implementasi pemecahan pada sub-sub kelompok. Komunikasi antar kelompok dan orang bersifat horizontal tetapi komunikasi secara vertikal berjalan bila hirarki kontrol berjalan.
3. Terkontrol Tersentralisasi (CC)
   Pemecahan tingkat puncak dan internal tim oleh pimpinan tim. Komunikasi dilakukan secara vertikal.

Terdapat 7 (tujuh) faktor proyek yang harus dipertimbangkan dalam merencanakan tim *software*, yaitu :

1. Kesulitan pada masalah
2. Ukuran program yang dihasilkan (LOC/Function)
3. Waktu tim (umur)
4. Tingkat dimana dapat dimodularitasi

1. Paradigma tertutup,
   Membentuk hierarki otoritas tradisional (mirip tim CC ) namum kurang inivatif.
2. Paradigma random
   Membentuk tim longgar dan tergantung pada inisiatif individual tim, untuk inovasi sangat baik (unggul) bila unjuk kerja tim teratur.
3. Paradigma terbuka
   Membentuk tim dengan cara tertentu sehingga banyak kontrol, inovasi banyak, cocok untuk masalah yang kompleks tetapi tidak seefisien tim lainnya.
4. Paradigma sinkron
   Mengorganisasikan tim untuk bekerja pada bagian-bagian kecil masalah dengan komunikasi aktif pada tim.
5. *Coordination and Communication issue* (Masalah koordinasi dan Komunikasi).

Proyek perangkat lunak mengalami kesulitan dikarenakan :

- **Skala**, usaha pengembangan yang besar sehingga kesulitan dalam mengkoordinasi anggota tim dan kompleksitas yang semakin besar.
- **Ketidakpstian**, mengakibatkan perubahan terus-menerus pada proyek.
- Interoperabilitas, merupakan cirri dari system dan menyesuaikan dengan batasan sistem.

Dalam menguji sekumpulan teknik koordinasi proyek menurut Kraul dan streeter, terbagi atas :

a. Pendekatan impersonal, formal penyampian dan dokumen rekayasa perangkat lunak (memo, laporan dan lainnya).
b. Prosedure interpersonal, formal aktivitas jaminan kualitas yang diterapkan kepada produk kerja rekayasa perangkat lunak (status pengkajian, perancangan dan inpeksi kode).
c. Prosedure interpersonal, informal pertemuan kelompok untuk menyebarkan informasi dan pemecahan masalah serta pengembangan staf.

- penulisan *source code* harus dilakukan dengan hati-hati dan senantiasa melalui tahap uji,
- dilengkapi dengan dokumen-dokumen pendukung seperti : prinsip pengoperasian, *user's manual*, instruksi instalasi, dokumen pemeliharaan,
- menyiapkan bantuan pelatihan.

**7.3. Kulitas dan Proses Perangkat Lunak**

Beberapa atribut yang merupakan ukuran kualitas perangkat lunak adalah :

a. kegunaan, yaitu pemenuhan terhadap kebutuhan pengguna,
b. keandalan, yaitu kemampuan melaksanakan fungsi yang diinginkan,
c. kejelasan, yaitu penulisan program dilakukan secara jelas dan mudah dimengerti,
d. efisiensi, terutama dalam waktu eksekusi dan penggunaan media penyimpanan (*memory*).

Proses perangkat lunak memberikan suatu kerangka kerja dimana rencana komprehensip bagi pengembangan perangkat lunak yang dapat dibangun dengan :

- sejumlah kumpulan tugas yang berbeda, kemapuan penyanpaian dan jaminan kualitas.
- Aktivitas pelindung, jaminan kualitas perangkat lunak, maanajemen konfigurasi perangkat lunak dan pengukuran.

Adapun model proses , diantaranya :

1. Sekunsial linier, *classic life cycle* (model air terjun / water fall).
2. *Prototype*, perencanaan kilat untuk konstruksi oleh prototype.
3. Rapid Aplication Development (RAD), model sekuntial linier yang menekankan siklus pengembangan yang sangat pendek dengan pendekatan konstruksi berbasis komponen.
4. Inkremntal (pertambahan), menggabungkan elemen-elemen model sekuensial linier dengan filosopi *prototype iterative* khusus untuk penempatan karyawan (*staffing*).

menggunakan *prototype*.. Tugas kerja yang actual bervariasi sehingga dekomposisi proses dimulai pada saat bagaimana menyelesaikan kerja proses secara umum.

### 7.4. Proyek Perangkat Lunak

Profesional industri sering mengacu pada aturan 90-90 yaitu pada saat mendiskusikan proyek perangkat lunak yang sukar maka 90% dari system yang pertama menyerap 90% dari usaha dan waktu yang diberikan. Sedangkan 10% sisanya yang terakhir mengambil 90% lain dari usaha dan waktu yang diberikan. Berdasarkan pernyataan diatas proyek mengalami kesulitan, yaitu :

1. kemajuan mengalami kecacatan.
2. tidak ada cara untuk mengkalibrasi kemajuan karena tidak memperoleh matrik kuantitatif.
3. Rencana proyek belum dirancang untuk mengakokodasi sumber daya yang diperlukan pada akhir sebuah proyek.
4. Jadual yangada tidak realistis dan cacat.

Untuk mengatasi masalah tersebut maka diperlukan waktu pada awal proyek untuk membangun rencana yang realistis guna memonitor rencana proyek selama berjalan dan pada keseluruhan proyek serta mengontrol kualitas serta perubahnnya.

**Upaya Masa Pendistribusian Produk Perangkat Lunak**

- Masa hidup sebuah produk perangkat lunak adalah 1 s/d 3 tahun dalam pengembangan dan 5 s/d 15 tahun dalam pemakaiannya (pemeliharannya).
- Distribusi upaya antara pengembangan dan pemeliharaan bervariasi antara 40/60, 30/70, dan bahkan 10/90.
- Tiga aktivitas pengembangan perangkat lunak adalah : analisa dan perancangan, implementasi dan pengujian.
- Tiga aktivitas pemeliharaan perangkat lunak adalah : peningkatan kemampuan produk, penyesuaian produk dengan lingkungan pemrosesan baru, dan perbaikan.

dinyatakan secara tegas, (2) produk perangkat lunak harus dirancang sedemikian rupa sehingga mampu mengakomodasi paling tidak kepentingan tiga pihak berikut : pelaksana implementasi, pengguna, dan pemelihara produk, (3) format penulisan *source code* harus dilakukan dengan hati-hati dan senantiasa melalui tahap pengujian secara simultan, (4) Adanya dokumen pendukung seperti : prinsip pengoperasian,( *user's manual*), instruksi untuk melakukan instalasi, dokumen pemeliharaan dan jaminan keamanan, serta (5) menyiapkan kegiatan untuk pelatihan.

**Latihan 7**

**Jawablah soal berikut ini dengan singkat dan jelas.**

6. Jelaskan tahapan Manajemen proyek perangkat lunak (*software*) yang efektif.
7. Jelaskan Model kematangan manajemen manusia
8. Jelaskan Organisasi dalam tim menurut Mantei
9. Jelaskan Pengujian teknik koordinasi proyek menurut Kraul dan streeter.
10. Jelaskan Kualitas dan Proses peranglat lunak
11. Jelaskan hal apa saja yang perlu di perhatikan dalam pengembangan proyek perangkat lunak.
12. Jelaskan dan berikan contoh model yang menggunakan RAD
13. Jelaskan dan berikan contoh model *prototype*.
14. Jelaskan dan berikan contoh model *waterfall*
15. Jelaskan paradigma menurut Constantine.

# GLOSARIUM

| | |
|---|---|
| *Algoritma* | Urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis. |
| Analisa | Suatu usaha dalam mengamati secara detail pada suatu hal dengan cara menguraikan komponen-komponen pembentuknya. |
| *Arrays* | Struktur data yang menyimpan sekumpulan elemen yang bertipe sama. |
| *Atribut* | Karakteristik atau ciri yang membedakan antara entitas satu dengan entitas yang lainnya. |
| *Black-box Testing* | Pengujian yang didasarkan pada detail aplikasi seperti tampilan aplikasi, fungsi-fungsi yang ada pada aplikasi, dan kesesuaian alur fungsi. |
| *Branch Coverage* | Merupakan suatu metode teknik pengujian *white box* yang memastikan bahwa setiap cabang di eksekusi minimal satu kali. |
| *Branch Nodes* | Suatu titik-titik yang memiliki lebih dari satu anak panah keluaran (*output*). |
| *Coding* | Salah satu tindakan dari langkah-langkah pemrograman dengan menuliskan kode atau skrip dalam bahasa pemrograman. |
| *Conditional* | Pengujian dengan melakukan pendekatan untuk merancang kasus uji untuk menjalankan kondisi logis yang terdapat dalam modul program aplikasi. |

| | |
|---|---|
| *Nodes* | Suatu pernyataan (atau sub program) yang akan ditinjau saat eksekusi program. |
| *Output* | Data yang telah diproses menjadi bentuk yang dapat digunakan. |
| *Path* | Suatu jalur yang mungkin bergerak dari satu titik ke titik lainnya sejalan dengan keberadaan arah anak panah. |
| PDL | Notasi bahasa yang menggunakan kosakata dari satu bahasa |
| Pengujian | Percobaan untuk mengetahui mutu sesuatu. |
| Perangkat Lunak | Program dan data yang tidak dapat disentuh secara fisik dan tersimpan dalam *hardware* yang dapat menjalankan komputer dan mengendalikan hardware. |
| Program | Serangkaian instruksi yang ditulis untuk melakukan suatu fungsi spesifik pada komputer. |
| *Programmer* | Seseorang yang bekerja membuat program komputer. |
| Prosedur | Instruksi yang dibutuhkan oleh pengguna dalam memproses informasi. Sekumpulan perintah yang merupakan bagian dari program yang lebih besar yang berfungsi mengerjakan suatu tugas tertentu. |
| Proses | Perubahan atau transformasi input menjadi output. |
| *Region* | Suatu area yang dibatasi oleh *edges* dan *node*, area diluar *graph* juga dihitung sebagai *region*. |
| Sistem | Kumpulan dari elemen-elemen yang saling berinteraksi untuk mencapai tujuan tertentu. |
| SPSS | Sebuah software pengolah data statistik atau yang digunakan untuk analisis statistik interaktif, atau batch. |

# DAFTAR PUSTAKA


E. W. Kastango, "Understanding Pharmacy Cleanroom Design Requirement," PPP, New Jersey, 2014.

Ian Sommerville, 2004, software engineering, 7th edition.

Jeffry L Whitten, Lonnie D, Bentley, Kevin. Dittman, 2004, System Analysis and Design Methods, 6th edition, Mc Graw Hill, New York.

K. Mikhail, "Cleanrooms at Pharmaceutical Production - Bachelor Thesis," Mikkeli University Applied Sciences, Findand, 2010. [16]E. W. Kastango, "Understanding Pharmacy Cleanroom Design Requirement," PPP, New Jersey, 2014.

Pressman, R.S, 2010, "Rekayasa Perangkat Lunak Buku 2", Andi, Yogyakarta.

S. P. R. Ranade, "Design and Development of Cost Effective Clean Rooms For Pharmaceutical Units," in Second International Conference on Emerging Trends in engineering (SICETE) , Jaysingpur, 2013.

https://www.dosenpendidikan.co.id/white-box-testing/,diakses tanggal 20-01-2021, 06.00 wib.

https://media.neliti.com/media/publications/234457-uji-validitas-dan-reabilitas-terhadap-im-fb26ecef.pdf, diakses tanggal 15-01-2021, 08.00 wib

M, Komaruddin MZ, "Pengujian Perangkat Lunak Metode BLACK-BOX Berbasis Equivalence Partitions Pada Aplikasi Sistem Informasi Sekolah" Jurnal Mikrotik Edisi Bulan Februari 2016,

# INDEX


## A

*Algoritma*, 40, 84
Analisa, 7, 80, 84
*arrays*, 6
*Atribut*, 5, 84

## B

*Black-box Testing*, 23, 84
**Branch coverage**, 29, 30, 84
*Branch nodes*, 32, 84

## C

*Coding*, 8, 38, 42, 84
**Concatenated loop**, 46
*conditional*, 40
*Counter*, 85
*Cyclomatic Complexity*, 36, 85

## D

*Database*, 70, 85

## E

*Edges*, 32, 85
*error*, 1, 5, 6, 12, 15, 24, 46, 47, 54, 58, 62, 63, 64
*Error*, 6, 59, 85
*Execution testing*, 28

## F

***flow graph***, 18, 27, 30, 34, 37, 39
*Flowchart*, 85
**Functional Testing**, 62, 85

## G

*Graph Matrix*, 43, 45, 53, 85

## H

**Human engineering test**, 69

## I

*Input*, 60, 63, 71, 85
*Interface*, 6, 16, 23, 24, 25, 85
*Interval*, 56, 57, 85

## L

*Line coverage*, 27, 53
*Loop*, 45, 53, 85