Pertemuan 9
Pengujian Perangkat Lunak
TIK : Menjelaskan konsep dasar dan metode pengujian perangkat lunak.

1. TESTING (PENGUJIAN PERANGKAT LUNAK)

Pengujian perangkat lunak merupakan elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean. Pentingnya pengujian perangkat lunak dan implikasinya yang mengacu pada kualitas perangkat lunak tidak dapat terlalu ditekan karena melibatkan sederetan aktivitas produksi di mana peluang terjadinya kesalahan manusia sangat besar dan arena ketidakmampuan manusia untuk melakukan dan berkomunikasi dengan sempurna maka pengembangan perangkat lunak diiringi dengan aktivitas jaminan kualitas.

Meningkatnya visibilitas (kemampuan) perangkat lunak sebagai suatu elemen sistem dan "biaya" yang muncul akibat kegagalan perangkat lunak, memotivasi dilakukannya perencanaan yang baik melalui pengujian yang teliti. Pada dasarnya, pengujian merupakan satu langkah dalam proses rekayasa perangkat lunak yang dapat dianggap sebagai hal yang merusak daripada membangun.

Sejumlah aturan yang berfungsi sebagai sasaran pengujian pada perangkat lunak adalah:

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 itu berlawanan dengan pandangan yang biasanya dipegang yang menyatakan bahwa pengujian yang berhasil adalah pengujian yang tidak ada kesalahan yang ditemukan. Data yang dikumpulkan pada saat pengujian dilakukan memberikan indikasi yang baik mengenai reliabilitas perangkat lunak dan beberapa menunjukkan kualitas perangkat lunak secara keseluruhan, tetapi ada satu hal yang tidak dapat dilakukan oleh pengujian, yaitu pengujian tidak dapat memperlihatkan tidak adanya cacat, pengujian hanya dapat memperlihatkan bahwa ada kesalahan perangkat lunak.

Sebelum mengaplikasikan metode untuk mendesain test case yang efektif, perekayasa perangkat lunak harus memahami prinsip dasar yang menuntun pengujian perangkat lunak, yaitu:

a. Semua pengujian harus dapat ditelusuri sampai ke persyaratan pelanggan, maksudnya mengungkap kesalahan dari cacat yang menyebabkan program gagal.
b. Pengujian harus direncanakan lama sebelum pengujian itu mulai, maksudnya semua pengujian dapat direncanakan dan dirancang sebelum semua kode dijalankan.
c. Prinsip Pareto berlaku untuk pengujian perangkat lunak, maksudnya dari 80% kesalahan yang ditemukan selama pengujian dapat ditelusuri sampai 20% dari semua modul program.
d. Pengujian harus mulai "dari yang kecil" dan berkembang ke pengujian "yang besar", Selagi pengujian berlangsung maju, pengujian mengubah focus dalam

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

e. Pengujian yang mendalam tidak mungkin karena tidak mungkin mengeksekusi setiap kombinasi jalur skema pengujian dikarenakan jumlah jalur permutasi untuk program menengah pun sangat besar.
f. Untuk menjadi paling efektif, pengujian harus dilakukan oleh pihak ketiga yang independent

Dalam lingkungan yang ideal, perekayasa perangkat lunak mendesain suatu program computer, sebuah sistem atau produk dengan testabilitas dalam pikirannya. Hal ini memungkinkan individu yang berurusan dengan pengujian mendesain test case yang efektif secara lebih mudah. Testabilitas adalah seberapa mudah sebuah program computer dapat diuji. Karena sangat sulit, perlu diketahui apa yang dapat dilakukan untuk membuatnya menjadi lebih mudah. Procedural dan menggunakannya sebagai pedoman untuk menetapkan basis set dari jalur eksekusi.

Sasaran utama desain test case adalah untuk mendapatkan serangkaian pengujian yang memiliki kemungkinan tertinggi di dalam pengungkapan kesalahan pada perangkat lunak. Untuk mencapai sasaran tersebut, digunakan 4 kategori yang berbeda dari tehnik desain test case: *Pengujian white-box*, *pengujian black-box*, *Integrasi Bottom-Up* dan *Integrasi Top-Down*.

Pengujian white-box berfokus pada struktur control program. Test case dilakukan untuk memastikan bahwa semua statemen pada program telah dieksekusi paling tidak satu kali selama pengujian dan bahwa semua kondisi logis telah diuji. Pengujian basic path, tehnik pengujian white-box, menggunakan grafik (matriks grafiks) untuk melakukan serangkaian pengujian yang independent secara linear yang akan memastikan cakupan.

Pengujian aliran data dan kondisi lebih lanjut menggunakan logika program dan pengujian loop menyempurnakan tehnik white-box yang lain dengan memberikan sebuah prosedur untuk menguji loop dari tingkat kompleksitas yang bervariasi. Pengujian black-box didesain untuk mengungkap kesalahan pada persyaratan fungsional tanpa mengabaikan kerja internal dari suatu program.

Tehnik pengujian black-box berfokus pada domain informasi dari perangkat lunak, dengan melakukan test case dengan menpartisi domain input dari suatu program dengan cara yang memberikan cakupan pengujian yang mendalam.

Metode pengujian graph-based mengeksplorasi hubungan antara dan tingkah laku objek-objek program. Partisi ekivalensi membagi domain input ke dalam kelas data yang mungkin untuk melakukan fungsi perangkat lunak tertentu. Analisis nilai batas memeriksaa kemampuan program untuk menangani data pada batas yang dapat diterima.

Metode pengujian yang terspesialisasi meliputi sejumlah luas kemampuan perangkat lunak dan area aplikasi. GUI, arsitektur client/ server, dokumentasi dan fasilitas help dan sistem real time masing-masing membutuhkan pedoman dan tehnik khusus untuk pengujian perangkat lunak.

Integrasi Top-Down adalah pendekatan incremental dengan menggerakkan ke bawah melalui hirarki control, dimulai dengan control utama. Strategi intergrasi top-down memeriksa control mayor atau keputusan pada saat awal di dalam proses pengujian. Pada struktur program yang difaktorkan dengan baik, penarikan keputusan terjadi pada tingkat hirarki yang lebih tinggi sehingga terjadi lebih dulu.

Strategi top-down kelihatannya tidak sangat rumit, tetapi di dalam praktenya banyak menimbulkan masalah logistic. Biasanya masalah ini terjadi jika dibutuhkan pemrosesan di dalam hirarki pada tingkat rendah untuk menguji secara memadai tingkat yang lebih tinggi.

Pengujian Integrasi Bottom-up memulai konstruksi dan pengujian dengan modul atomic (modul pada tingkat paling rendah pada struktur program). Karena modul diintegrasikan dari bawah ke atas, maka pemrosesan yang diperlukan untuk modul subordinate ke suatu tuingkat yang diberikan akan selalu tersedia dan kebutuhan akan stub dapat dieliminasi. Strategi integrasi bottom-up dapat diimplementasi dengan langkah-langkah:

a. Modul tingkat rendah digabung ke dalam cluster (build) yang melakukan subfungsi perangkat lunak spesifik.
b. *Driver* (program control untuk pengujian) ditulis untuk mengkoordinasi input dan output test case
c. *Cluster* diuji
d. *Driver* diganti dan *cluster* digabungkan dengan menggerakkannya ke atas di dalam struktur program.

2. IMPLEMENTASI ENTEPRISE SISTEM

Enterprise system adalah sistem berbasis software untuk membantu pengelolaan sistem informasi pada suatu organisasi dengan skala besar. Skala besar berarti volume transaksi yang besar, concern terhadap kualitas informasi yang tinggi, mengintegrasikan berbagai proses bisnis, lintas bidang (horisontal) maupun lintas strata (vertikal). Contoh dari ES adalah ERP (*Enterprise Resource Planning*) atau *e-Business* secara umum, *e-Government*, dan *ingrated software* lainnya.

Mengimplementasikan ES tidak mudah, atau setidaknya memilki strategi yang berbeda dengan sistem lain yang terbatas ruang lingkupnya, penggunanya dan tidak terpadu. Implementasi di sini bermakna bahwa software telah dapat digunakan dan bisa memberikan value bagi penggunanya sesuai tujuan pemanfaatan software tsb. Implementasi ini bisa dilakukan secara internal organisasi (oleh divisi IT/MIS) atau dengan pihak eksternal dalam kerangka proyek dan terikat legalitas berbentuk kontrak. Implementator sebagai pihak eksternal yang melakukan implementasi dan klien sebagai organisasi yang diimplementasikan softwarenya.

Implementasi ES berbeda dengan implementasi software berskala kecil atau yang penggunanya tunggal seperti MS Word, Database Rental VCD atau website, meskipun produknya sama-sama software yang berjalan di atas server dan membutuhkan konektivitas. Tentu nanti ada strategi yang berbeda, metode pemilihan bahan yang berbeda, tahapan yang berbeda, standar-standar tertentu, dst. Demikian pula dalam konteks software, bisa dipilah berdasar cakupan penggunaannya, bisa dilihat juga dari jenisnya (generik dan customized), yang masing-masing punya strategi implementasi yang berbeda. SE berkaitan dengan pengelolaan sistem informasi, yang tidak hanya bicara teknologi saja, tapi berkaitan dengan proses bisnis, struktur organisasi dan manusianya.

Pola pikir ”developer” adalah menganggap suatu problem bisa selesai dengan solusi berbasis software yang baik dan tepat. Tapi apakah cukup seperti itu? Dalam membangun solusi, ya itu cukup, tapi belum tentu menjamin kesuksesan implementasi. Pola pikir developer cenderung berfokus pada analisis dan development tidak pada implementasinya. Padahal sukses tidaknya proyek software, baik buruknya reputasi

implementator, seringkali orang luar melihat pada keberhasilan implementasinya dan value yang didapatkan klien. ES untuk organisasi dengan puluhan divisi, ribuan orang, puluhan kepentingan, dan mungkin ratusan konflik. Apalagi jika software yang kita implementasikan bukan sekedar supporting tools tapi adalah core dari bisnis itu sendiri (konsep e-business). Cara implementasi dengan pola pikir seperti ini hanya akan menghasilkan solusi dan software yang bagus, tapi tidak optimal dan memberikan value untuk organisasi tsb, atau bahkan malah tidak pernah akan digunakan.

Implementator tidak bisa memposisikan diri sebagai project manager pada sebuah proyek yang berkaitan langsung dengan proses bisnis internal klien. Seorang project manager harus mampu mengelola semua resource berkaitan dengan proyek. Kadang kita tidak menyadari bahwa sebagaian besar resource dari proyek software justru berada di sisi organisasi klien. Sementara, project manager seharusnya memiliki akses ke seluruh resource tersebut, karena jika tidak, itu bukan project manager namanya.

Dalam kasus ini, maka project manager seharusnya justru berada di sisi klien, bukan implementator. Akan sia-sia jika aktivitas project planning, project controlling dsb sepenuhnya dilakukan oleh implementator, sementara klien hanya ”tahu beres” saja. Pada akhirnya aktivitas-aktivitas project management tsb hanya akan menghasilkan berkas-berkas dan dokumen administratif saja, yang pada kenyataannya tidak pernah dilaksanakan.

Peran yang paling pas untuk implementator adalah sebagai konsultan. Tugas utama dari konsultan adalah memberikan informasi, mendampingi, memfasilitasi dan menjadi motor ”behind the screen”. Tentu saja jika kontraknya melibatkan pengadaan software, konsultan juga akan melakukan development atau implementasi secara teknis, namun implementasi keseluruhannya harus dipimpin oleh klien sendiri melalui project manager. Jika klien tidak memiliki pengetahuan yang cukup untuk mengelola proyek software, itulah tugas konsultan untuk mendampinginya, sehingga proses project planning, control, evaluation, dst sepenuhnya akan berasal dari ide-ide, komitmen dan effort dari klien sendiri.

Tugas konsultan adalah memfasilitasi dan mengarahkannya. Model seperti ini yang kemudian memunculkan teknik JAD (*Joint Application Design*), yang intinya adalah melibatkan dan kolaborasi seluruh stakeholder proyek. salah satu fase dalam implementasi sistem adalah fase transisi, yang pasti akan menuntut perubahan baik kecil maupun besar. Adanya sistem baru, mau tidak mau akan merubah proses bisnis. Perubahan proses bisnis berarti perubahan cara kerja, alur kerja dan bahkan budaya kerja. Perubahan ini menyangkut aspek people dan proses bisnis, sehingga dikenal konsep *change management*.

Dalam eksekusinya, change ini harus dipimpin dan dimanage oleh leader di internal organisasi. Yang jelas seorang konsultan tidak hanya dituntut memiliki pengetahuan tentang software engineering dan hal-hal teknis, dan juga tidak cukup ditambah dengan pengalaman dan keterampilan project management, namun konsep dan bestpractice tentang change management, communication skill yang excellent sangat diperlukan.

JAD (*Joint Application Development/Design*) sebagai salah satu teknik manajemen dalam mengimplementasikan sebuah sistem informasi (SI) dalam konteks proyek. porsi terbesar dan terumit dari proses implementasi SI adalah justru pada proses transisinya, karena terkait banyak aspek tidak hanya di sisi teknologi tapi harus memahami sisi sosial, manajerial dan SDM.

3. IMPLEMENTASI SI

Masalah terbesar dari implementasi SI adalah untuk mengetahui kebutuhan dari *user*, apalagi dengan karakter proyek :

- Sistem yang melibatkan multi-organisasi/divisi (penggunanya dari beberapa *role* dan divisi)
- Bisnis proses yang kompleks
- Kebutuhan yang sangat spesifik dan *customized*.

Dengan karakter proyek yang semacam ini, tidak cukup bagi seorang *system analyst* (SA) menentukan kebutuhan hanya dengan teknik wawancara, observasi ataupun kuesioner. Banyak kasus ditemui, bahwa pada akhirnya apa yang kita dapatkan dari proses analisa kebutuhan di awal proyek, tidak match dengan kebutuhan sesungguhnya dari pengguna sistem, sehingga sistem akhirnya tidak dapat digunakan dengan baik. Masalah lain adalah di sisi waktu. Teknik-teknik seperti itu seringkali sangat *time consuming*, sangat membutuhkan waktu yang lama. Sering juga tim *developer* dihadapkan situasi bahwa tidak semua *stakeholder* proyek memiliki kepedulian yang sama dengan yang lain. Seorang manajer tidak mengetahui kebutuhan detail dari staf-staf operasional, sementara itu staf operasional mungkin juga tidak memahami sepenuhnya *spirit*, *goal* dari SI. JAD merupakan sebuah teknik yang berfokus pada keterlibatan dan komitmen pengguna dalam menentukan kebutuhan dan merancang (desain) aplikasi. JAD biasanya dilakukan dalam bentuk tim yang merupakan gabungan dari seluruh stakeholder proyek, yang bekerja dalam bentuk *workshop-workshop* atau forum diskusi.

Kenapa *workshop* ? karena teknik JAD ini bukanlah sekedar rapat-rapat, yang biasa dilakukan dalam sebuah proyek dan melibatkan seluruh stakeholder proyek. JAD adalah tim yang nantinya akan membuat rancangan dan mengawasi, memonitor bersama jalannya proyek.

Siapa yang perlu terlibat ?

Secara garis besar yang perlu terlibat adalah :

1. **Sponsor**. Sponsor ini berarti *project owner*, memiliki kedudukan yang cukup tinggi dalam organisasi dan sebagai pengambil keputusan tertinggi dalam pengelolaan sistem informasi. Satu hal yang penting dilakukan oleh seorang *project owner* adalah komitmen yang kuat akan implementasi SI yang dilakukan. *Without the executive sponsor's commitment, people do not show up for workshops on time or sometimes at all. Schedules change and projects are delayed. In short, without an executive sponsor, there is no project!*
2. **Business Users**. *Business User* ini terdiri dari 2 jenis, yaitu *real end user* dan *representative end user*. *Real end user* adalah person yang melakukan pekerjaan real di lapangan. Dalam kasus, ini adalah operator-operator. Sedangkan *representative end user* adalah person yang mengetahui seharusnya bisnis proses itu dilakukan, memahami spirit dan goal dari sistem yang dikelolanya. Biasanya ini adalah kepala bagian, manajer, atau operator senior.
3. **System Analyst (Tim Developer)**. Person/tim ini yang akan in-charge dari sisi teknologi dan proses *engineering*nya.
4. **System Experts**. Tidak semua referensi mencantumkan peran ini. Perannya lebih seperti konsultan yang memahami seluk beluk bisnis proses dari sisi konseptual dan berbasis pengalaman.

5. **Facilitator**. Seorang fasilitator berfungsi sebagai moderator dan mengarahkan setiap aktivitas JAD yang melibatkan banyak pihak, untuk menjadi efektif. Seorang fasilitator harus memiliki kecakapan yang baik dalam berkomunikasi, memberikan stimulus-stimulus dan trik-trik agar diskusi bisa berjalan dengan baik.

Tentu saja, setelah penyusunan tim JAD, diperlukan strategi yang tepat dalam melakukan workshop-workshop, sehingga proses dilakukan lebih efektif. Yang jelas, teknik ini sudah terbuktif efektif dalam menyelesaikan masalah-masalah implementasi SI. Berikut contoh kesimpulan studi kasus oleh Standish grouph report:

a. Parameter Keberhasilan

Success Criteria	Points	DMV	CONFIRM	HYATT	ITAMARATI
1. keterlibatan pemakai	19	NO (0)	NO (0)	YES (19)	YES (19)
2. dukungan manajemen eksekutif	16	NO (0)	YES (16)	YES (16)	YES (16)
3. kebutuhan yg jelas	15	NO (0)	NO (0)	YES (15)	NO (0)
4. perencanaan yg sesuai	11	NO (0)	NO (0)	YES (11)	YES (11)
5. harapan yg realistis	10	YES(10)	YES (10)	YES(10)	YES (10)
6. proyek terkecil	9	NO (0)	NO (0)	YES (9)	YES (9)
7. staff yg kompeten	8	NO (0)	NO (0)	YES (8)	YES (8)
8. pemilik	6	NO (0)	NO (0)	YES (6)	YES (6)
9. visi & sasaan ygjelas	3	NO (0)	NO (0)	YES (3)	YES (3)
10. kerja keras, staff dipusatkan	3	NO (0)	YES (3)	YES (3)	YES (3)
TOTAL	100	10	29	100	85

b. Sukses / gagalnya proyek

Project Success Factors	% of Responses
1. keterlibatan pemakai	15.9%
2. dukungan manajemen eksekutif	13.9%
3. kebutuhan yg jelas	13.0%
4. perencanaaan yg sesuai	9.6%
5. harapan yg realistis	8.2%
6. proyek terkecil	7.7%
7. staff yg kompeten	7.2%
8. pemilik	5.3%
9. visi dan sasaran yg jelas	2.9%
10.kerja keras, staff dipusatkan	2.4%
Lainnya	13.9%

c. Project Challenged Factors

Project Challenged Factors	% of Responses
1. tidak ada masukkan dari pemakai	12.8%

2. kebutuhan & spesifikasi tg tdk sempurna	12.3%
3. mengubah kebutuhan dan spesifikasi	11.8%
4. tidak ada dukungan dr manajemen eksekutif	7.5%
5. ketidakmampuan teknologi	7.0%
6. tidak ada sumber daya	6.4%
7. harapan yg tdk realistis	5.9%
8.sasaran tdk jelas	5.3%
9. batasan waktu tdk realistis	4.3%
10. teknologi baru	3.7%
Lainnya	23.0%

d. Project Impaired Factors

Project Impaired Factors	**% of Responses**
1. kebutuhan tdk lengkap	13.1%
2. tidak ada masukan/keterlibatan dr pemakai	12.4%
3. tidak ada sumber daya	10.6%
4. harapan yg tdk realistis	9.9%
5. tidak ada dukungan dr manajemen eksekutif	9.3%
6. perubahan kebutuhan dan spesifikasi	8.7%
7. tidak ada perencanaan	8.1%
8. tidak diperlukan sama sekali	7.5%
9. tidak ada manajemen IT	6.2%
10. buta teknologi	4.3%
Lainnya	9.9%

4. STRATEGI PENGUJIAN PL

Strategi uji coba PL memudahkan para perancang untuk menentukan keberhasilan system yg telah dikerjakan. Hal yg harus diperhatikan adalah langkah-langkah perencanaan dan pelaksanaan harus direncanakan dengan baik dan berapa lama waktu, upaya dan sumber daya yg diperlukan. Strategi uji coba mempunyai karakteristik sbb :

a. Pengujian mulai pada tingkat modul yg paling bawah, dilanjutkan dgn modul di atasnya kemudian hasilnya dipadukan.
b. Teknik pengujian yang berbeda mungkin menghasilakn sedikit perbedaan (dalam hal waktu)
c. Pengujian dilakukan oleh pengembang perangkat lunak dan (untuk proyek yang besar) suatu kelompok pengujian yang independen.
d. Pengujian dan debugging merupakan aktivitas yang berbeda, tetapi debugging termasuk dalam strategi pengujian.

Pengujian PL adalah satu elemen dari topik yang lebih luas yang sering diacu sebagai *verifikasi dan validasi (V& V). Verifikasi* merupakan kumpulan aktifitas yg menjamin penerapan PL benar-benar sesuai dgn fungsinya dan *Validasi* merupakan kumpulan

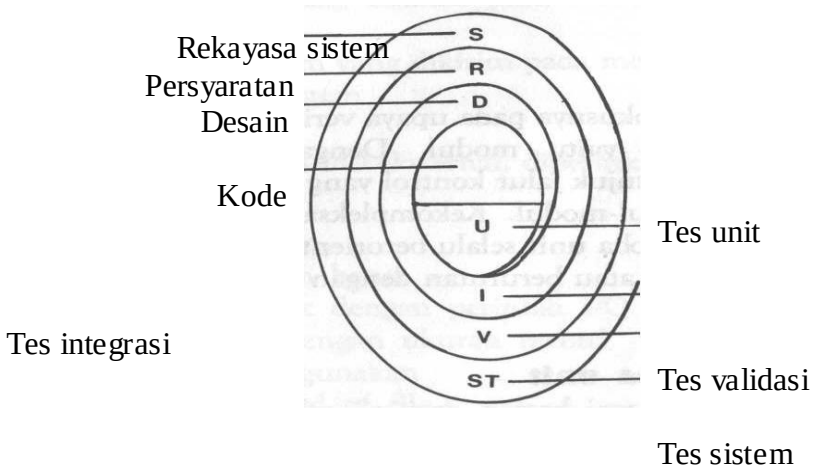
aktivitas yang berbeda yang memastikan bahwa PL yang dibangun dapat memenuhi keperluan pelanggan. Dengan kata lain :

Verifikasi : " Apakah kita membuat produk dgn benar?"

Validasi : " Apakah kita membuat benar-benar suatu produk?"

Definisi dari V&V meliputi berbagai aktivitas yang kita rujuk sebagai jaminan kualias PL (SQA).

Pengujian merupakan salah satu tugas yg ada dlm arus siklus pengembangan system yg dapat digambarkan dalam bentuk spiral :



Gambar. Strategi Uji Coba

4.1. PENGUJIAN UNIT

Unit testing (uji coba unit) fokusnya pada usaha verifikasi pada unit terkecil dari desain PL, yakni modul. Uji coba unit selalu berorientasi pada white box testing dan dapat dikerjakan paralel ayau beruntun dengan modul lainnya.

a. Pertimbangan Pengujian Unit

Interface diuji cobakan untuk menjamin informasi yg masuk atau yg ke luar dari unit program telah tepat atau sesuai dgn yg diharapkan. Yg pertama diuji coba adalah interface karena diperlukan untuk jalannya informasi atau data antar modul.

Myers mengusulkan checklist untuk pengujian interface:

- Apakahjumlah parameter input sama dengan jumlah argumen?
- Apakah antara atribut dan parameter argumen sudah cocok?
- Apakah antara sistem satuan parameter dan argumen sudah cocok?
- Apakah jumlah argumen yang ditransmisikan ke modul yang dipanggil sama dengan jumlah parameter?
- Apakah atribut dari argumen yang ditransmisikan ke modul yang dipanggil sama dengan atribut parameter?
- Apakah sistem unit dari argumen yang ditransmisikan ke modul yang dipanggil sama dengan sistem satuan parameter?
- Apakah jumlah atribut dari urutan argumen ke fungsi-fungsi built-in sudah benar?
- Adakah referensi ke parameter yang tidak sesuai dengan pain entri yang ada?
- Apakah argumen input-only diubah?

- Apakah definisi variabel global konsisten dengan modul?
- Apakah batasan yang dilalui merupakan argumen?

Bila sebuah modul melakukan I/O ekstemal, maka pengujian interface tambahan harus dilakukan.

- Atribut file sudah benar?
- Pemyataan OPEN/CLOSE sudah benar?
- Spesifikasi format sudah cocok dengan pernyataan I/O?
- Ukuran buffer sudah cocok dengan ukuran rekaman?
- File dibuka sebelum penggunaan?
- Apakah kondisi End-of-File ditangani?
- Kesalahan I/O ditangani?
- Adakah kesalahan tekstual di dalam informasi output?

Kesalahan yang umum di dalam komputasi adalah:

- kesalah-pahaman atau prosedur aritmatik yang tidak benar
- operasi mode yang tercampur
- inisialisasi yang tidak benar
- inakurasi ketelitian
- representasi simbolis yang tidak benar dari sebuah persamaan.

Test case harus mengungkap kesalahan seperti

- perbandingan tipe data yang berbeda
- preseden atau operator logika yang tidak benar
- pengharapan akan persamaan bila precision error membuat persamaan yang tidak mungkin
- perbandingan atau variabel yang tidak benar
- penghentian loop yang tidak ada atau tidak teratur
- kegagalan untuk keluar pada saat terjadi iterasi divergen
- variabel loop yang dimodifikasi secara tidak teratur.

b. Prosedur Pengujian Unit

Program sumber telah dikembangkan, ditunjang kembali dan diverifikasi untuk sintaksnya, maka perancangan test case dimulai. Peninjauan kembali perancangan informasi akan menyediakan petunjuk untuk menentukan test case. Karena modul bukan program yg berdiri sendiri maka driver (pengendali) dan atau stub PL harus dikembangkan untuk pengujian unit. **Driver** adalah program yg menerima data untuk test case dan menyalurkan ke modul yg diuji dan mencetak hasilnya. **Stub** melayani pemindahan modul yg akan dipanggil untuk diuji.

5. PENGUJIAN INTEGRASI

Pengujian terintegrasi adl teknik yg sistematis untuk penyusunan struktur program, pada saat bersamaan dikerjakan uji coba untuk memeriksa kesalahan yg nantinya digabungkan dengan interface. Metode pengujiannya meliputi *Top down integration* dan *buttom up integration*

Top down integration merupakan pendekatan inkremental untuk penyusunan struktur program. Modul dipadukan dgn bergerak ke bawah melalui kontrol hirarki dimulai dari modul utama. Modul subordinat ke modul kontrol utama digabungkan ke dalam struktur baik menurut depth first atau breadth first. Sedangkan Proses integrasi meliputi :

- modul utama digunakan sebagai test driver dan stub yg menggantikan seluruh modul yg secara langsung berada di bawah modul kontrol utama.

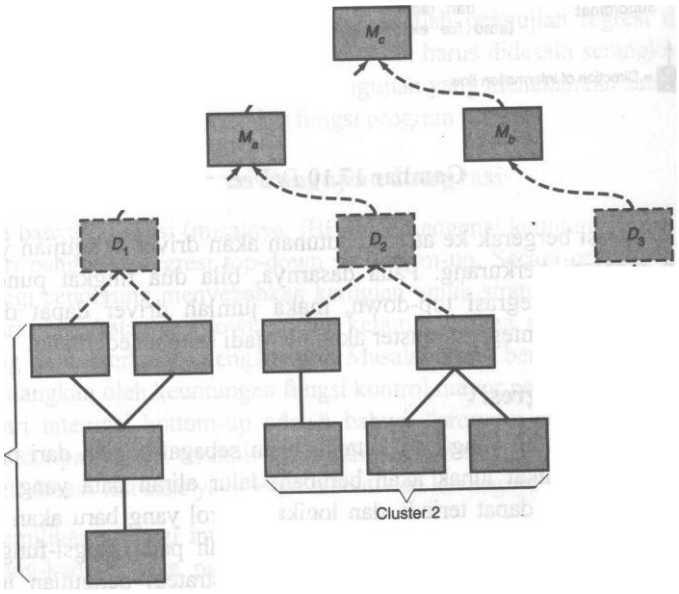
- Tergantung pada pendekatan perpaduan yg dipilih (depth / breadth)
- Uji coba dilakukan selama masing-masing modul dipadukan
- Pada penyelesaian masing-masing uji coba stub yg lain dipindahkan dgn modul sebenarnya.
- Uji coba regression yaitu pengulangan pengujian untuk mencari kesalahan lain yg mungkin muncul.

Pengujian buttom up dinyatakan dgn penyusunan yg dimulai dan diujicobakan dgn atomic modul (yi modul tingkat paling bawah pd struktur program). Karena modul dipadukan dari bawah ke atas, proses yg diperlukan untuk modul subordinat yg selalu diberikan harus ada dan diperlukan untuk stub yg akan dihilangkan. Adapun Strategi pengujiannya yaitu:

a. Modul tingkat bawah digabungkan ke dalam cluster yg memperlihatkan subfungsi PL
b. Driver (program kontrol pengujian) ditulis untuk mengatur input test case dan output
c. Cluster diuji
d. Driver diganti dan cluster yg dikombinasikan dipindahkan ke atas pada struktur program

fgd

Cluster 1



Gambar Buttom Up Integration

6. UJI COBA VALIDASI

Setelah semua kesalahan diperbaiki maka langkah selanjutnya adalah validasi terting. Pengujian validasi dikatakan berhasil bila fungsi yg ada pada PL sesuai dgn yg

diharapkan pemakai. Validasi PL merupakan kumpulan seri uji coba black box yg menunjukkan sesuai dgn yg diperlukan.Kemungkinan kondisi setelah pengujian:

1. Karakteristik performansi fungsi sesuai dgn spesifikasi dan dapat diterima.
2. Penyimpangan dari spesifikasi ditemukan dan dibuatkan daftar penyimpangan.

Pengujian BETA dan ALPHA

Apabila PL dibuat untuk pelanggan maka dapat dilakukan aceeptance test sehingga memungkinkan pelanggan untuk memvalidasi seluruh keperluan. Test ini dilakukan karena memungkinkan pelanggan menemukan kesalahan yg lebih rinci dan membiasakan pelanggan memahami PL yg telah dibuat.

Pengujian Alpha

Dilakukan pada sisi pengembang oleh seorang pelanggan. PL digunakan pada setting yg natural dgn pengembang "yg memandang" melalui bahu pemakai dan merekam semua kesalahan dan masalah pemakaian.

Pengujian Beta

Dilakukan pada satu atau lebih pelanggan oleh pemakai akhir PL dalam lingkungan yg sebenarnya, pengembang biasanya tidak ada pada pengujian ini. Pelanggan merekan semua masalah (real atau imajiner) yg ditemui selama pengujian dan melaporkan pada pengembang pada interval waktu tertentu.

7. UJI COBA SISTEM

Pada akhirnya PL digabungkan dgn elemen system lainnya dan rentetan perpaduan system dan validasi tes dilakukan. Jika uji coba gagal atau di luar skope dari proses daur siklus pengembangan system, langkah yg diambil selama perancangan dan pengujian dapat diperbaiki. Keberhasilan perpaduan PL dan system yg besar merupakan kuncinya. Sistem testing merupakan rentetan pengujian yg berbeda-beda dgn tujuan utama mengerjakan keseluruhan elemen system yg dikembangkan, yaitu :

a. Recovery Testing

Adalah system testing yg memaksa PL mengalami kegagalan dalam bermacam-macam cara dan memeriksa apakah perbaikan dilakukan dgn tepat.

b. Security Testing

Adalah pengujian yg akan melalukan verifikasi dari mekanisme perlindungan yg akan dibuat oleh system, melindungi dari hal-hal yg mungkin terjadi.

c. Strees Testing

Dirancang untuk menghadapi situasi yg tidak normal pada saat program diuji. Testing ini dilakukan oleh system untuk kondisi seperti volume data yg tidak normal (melebihi atau kurang dari batasan) atau frkkuensi.

Daftar Pustaka

1. Presman, Rouger S, *Software Enigineering*, 4th Edition, Mc. Graw Hill,1997.
2. Sommerville,Ian, *Software Engineering*, 7th Edition, Addison Wesley, 2004.
3. Kendall & Kendall, *Systems Analysis and Design*, 6th Edition, Prentice Hall,2006.