BAB VI
REGRESSION TESTING

A. Test Plans

Test Plan adalah dokumen yang berisi definisi tujuan dan sasaran pengujian dalam lingkup iterasi (atau proyek), item-item yang menjadi target pengujian, pendekatan yang akan diambil, sumber daya yang dibutuhkan dan poin untuk diproduksi. Dengan kata lain *test plan* dapat disebut sebagai perencanaan atau skenario untuk melakukan *testing* yang akan dilakukan baik oleh *expert* atau pengguna umum. Tujuan membuat *test plan* secara umum adalah untuk memudahkan *developer* ~~untuk~~ dalam melakukan *testing*, agar *testing* yang dilakukan menjadi jelas sehingga hasilnya lebih berguna dan efisien.

Pembuatan *test plan* dapat dibuat dengan mengikuti *template* pembuatan *test plan*, namun tidak selalu harus mengacu kepada *template test*. Berikut adalah penjelasan mengenai *template test plan* yang dikeluarkan oleh IEEE 829 (*Institute of Electrical and Electronics Engineers*) :

1. *Test Plan Identifier*

   Test Plan Identifier adalah bagian untuk menjelaskan secara singkat mengenai objek yang akan diuji. Bisa berupa penjelasan narasi atau berbentuk tabel dengan kategori – kategori tertentu. Informasi yang dijelaskan dapat berupa sekilas mengenai subjek *testing*, nama orang yang bertanggung jawab terhadap *testing*, penyusun *test plan*, tanggal dibuat *test plan*, ~~dan~~ tanggal revisi,dll.

2. *Introduction*

   Pada bagian *introduction* dibuat untuk menjelaskan secara narasi, mengenai *testing* yang akan dilakukan terhadap suatu objek. Bagian *introduction* dapat dibuat lebih rinci dengan menambahkan sub-bab apabila perlu untuk dibuat. Contoh subbab yang dapat dibuat antara lain :

   1) *Purpose* : untuk menjelaskan tujuan *testing* secara spesifik.
   2) *Background* : latar belakang mengapa *testing* dilakukan.
   3) *Scope* : Sejauh mana *testing* dilakukan.
   4) *Definition and Acronyms* : penjelasan mengenai singkatan dan isitlah yang ada di dalam dokumen *test plan*.

3. *Test Items*

Bagian *test item* menjelaskan mengenai daftar komponen-komponen dalam objek *testing* yang akan di uji satu per satu.

4. *Features to be tested*

Penjelasan dan daftar daftar fitur yang akan ditest ~~di~~ pada saat pelaksanaan *testing* dimulai.

5. *Features Not to be Tested*

Menjelaskan mengenai fitur-fitur apa saja yang ada di dalam objek *testing*. Namun, fitur tersebut tidak akan di *test* pada saat pelaksanaan *testing*, dan disertakan penjelasan singkat mengapa fitur tersebut tidak di uji pada saat *testing*.

6. *Approach/Test Strategy*

Bagian *Approach* adalah bagian yang digunakan untuk memberi deskirpsi mengenai cara yang dilakukan untuk melaksanakan *testing*, dan disertakan dengan penjelasan mengenai cara ~~yang digunakan~~. Penggunaannya

7. *Item Pass/Fail Criteria*

Berisi tentang kriteria-kriteria yang harus dipenuhi sebelum berlanjut ke fase berikutnya. Kriteria yang dimaksud merangkup kriteria yang benar serta kriteria yang salah.

8. *Suspension Criteria*

Berisi tentang spesifikasi kriteria-kriteria yang dapat digunakan untuk menghentikan sementara kegiatan *testing*, dan *testing* tersebut dapat dilanjutkan di waktu lain.

9. *Test Deliverables*

Merupakan daftar dokumen-dokumen yang akan dihasilkan setelah *testing* selesai dilakukan.

10. *Testing Task*

Menjelaskan kegiatan *testing* beserta dengan pihak yang akan melaksanakan kegiatan tersebut.

11. Enviromental Needs

Spesifikasi dan perincian segala sesuatu yang dibutuhkan dan digunakan selama proses *testing* berjalan, bisa berupa *hardware* yaitu spesifikasi komputer atau hal lain selain *hardware*.

12. Responsibilities

Rincian pihak-pihak yang akan bertanggung jawab terhadap suatu kegiatan di dalam serangkaian kegiatan *testing* yang akan dilaksanakan.

13. Staffing and Training Needs

Secara garis besar menjelaskan bagaimana melakukan pendekatan untuk menentukan peran para anggota di dalam proyek dan melakukan pelatihan apabila diperlukan.

14. Schedule

Ada beberapa tujuan dalam membuat *schedule* di dalam *test plan*, antara lain :

1) Merincikan tolak ukur waktu pengerjaan *testing*.
2) Merincikan *event transmittal item*.
3) Estimasi waktu yang dibutuhkan untuk setiap *task*.
4) Menjadwalkan *testing task* dan *test milestone*.
5) Merincikan periode pemakaian *testing resources*.

15. Risk and Contingencies

Digunakan untuk memastikan agar hasil *testing* tetap berkualitas dengan memeriksa beberapa bagian yang tidak termasuk di dalam kontrol pengerjaan *software*, namun bagian tersebut dapat berdampak langsung terhadap proses.

16. Approvals

Lembar persetujuan sebagai tanda bahwa seluruh tim/pimpinan telah menyetujui *test plan* yang telah dibuat.

B. What is Regression Testing?

Pengujian Regresi didefinisikan sebagai jenis pengujian perangkat lunak untuk mengonfirmasi bahwa program atau perubahan kode baru-baru ini tidak berdampak buruk

pada fitur yang ada. Pengujian Regresi tidak lain adalah seleksi penuh atau sebagian dari kasus uji yang sudah dieksekusi, ~~yang~~ dieksekusi ulang untuk memastikan fungsionalitas yang ada berfungsi dengan baik. Pengujian ini dilakukan untuk memastikan bahwa perubahan kode baru tidak memiliki efek samping pada fungsionalitas yang ada. Ini memastikan bahwa kode lama masih berfungsi setelah perubahan kode ~~ter~~baru selesai.

Kebutuhan Pengujian Regresi terutama timbul setiap kali ada kebutuhan untuk mengubah kode dan kita perlu menguji apakah kode dimodifikasi mempengaruhi bagian lain dari aplikasi perangkat lunak atau tidak. Selain itu, pengujian regresi diperlukan, ketika fitur baru ditambahkan ke aplikasi perangkat lunak dan untuk perbaikan cacat serta perbaikan masalah kinerja. Untuk melakukan proses pengujian regresi, dimulai dari proses *debugging* kode untuk mengidentifikasi *bug*. Setelah *bug* diidentifikasi, ~~perubahan yang dibuat~~ perlukan untuk memperbaiki~~nya~~, kemudian pengujian regresi dilakukan dengan memilih kasus uji yang relevan dari rangkaian pengujian yang mencakup bagian kode yang dimodifikasi dan terpengaruh.

Pemeliharaan perangkat lunak adalah kegiatan yang meliputi peningkatan, koreksi kesalahan, pengoptimalan, dan penghapusan fitur yang ada. Modifikasi ini dapat menyebabkan sistem bekerja dengan tidak benar. Oleh karena itu, pengujian regresi menjadi penting untuk dilakukan. Seleksi uji regresi adalah teknik di mana beberapa kasus uji yang dipilih dari rangkaian uji dijalankan untuk menguji apakah kode yang dimodifikasi mempengaruhi aplikasi perangkat lunak atau tidak.

Kasus uji dikategorikan menjadi dua bagian, kasus uji yang dapat digunakan kembali pada siklus regresi selanjutnya dan kasus uji usang yang tidak dapat digunakan pada siklus berikutnya. Prioritas kasus uji tergantung pada dampak bisnis, fungsi penting & sering digunakan. Pemilihan kasus uji berdasarkan prioritas akan sangat mengurangi rangkaian uji regresi. Berdasarkan data industri, sejumlah besar cacat yang dilaporkan oleh pelanggan disebabkan oleh perbaikan *bug* pada "menit-menit terakhir" yang menciptakan efek samping. Oleh karena itu, memilih kasus uji untuk pengujian regresi bukanlah hal yang mudah. Uji regresi efektif dapat dilakukan dengan memilih kasus uji berikut:

1. Kasus uji yang sering mengalami cacat
2. Fungsionalitas yang lebih terlihat oleh pengguna
3. Uji kasus yang memverifikasi fitur inti produk
4. Uji kasus fungsionalitas yang telah mengalami lebih banyak perubahan

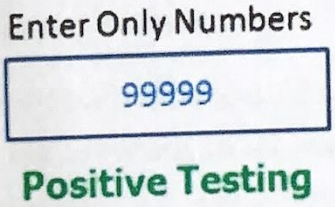
5. Semua kasus uji integrasi
6. Semua kasus uji kompleks
7. Kasus uji nilai batas
8. Contoh kasus uji yang berhasil
9. Contoh kasus uji Kegagalan

C. Positive Testing and Negative Testing

Pengujian secara positif adalah untuk menentukan bahwa aplikasi berfungsi seperti yang diharapkan. Jika terjadi kesalahan selama pengujian positif, pengujian dianggap gagal. Adapun pengujian secara negatif adalah memastikan bahwa aplikasi dapat menangani input yang tidak valid atau perilaku pengguna yang tidak terduga.

a. Positive testing

Pengujian ini berupa menguji inputan nilai yang *valid*. Kemudian melihat apakah aplikasi berjalan sesuai dengan harapan. Contohnya pada inputan berikut ini:

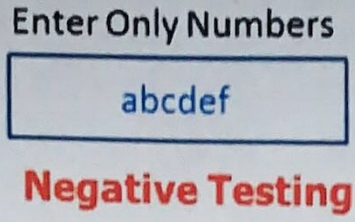


Gambar 6.1 Pengujian dengan inputan yang benar

Memasukkan nilai dengan 99999 pada kolom tersebut dapat diterima/*valid* oleh sistem dikarenakan kolom tersebut bertipe Number.

b. Negative testing

Pada *negative testing*, pengujian yang dilakukan berupa menguji inputan yang salah, dan melihat apakah program akan memberikan notifikasi berupa *invalid* atau tidak. Contohnya jika kolom inputan seperti berikut :



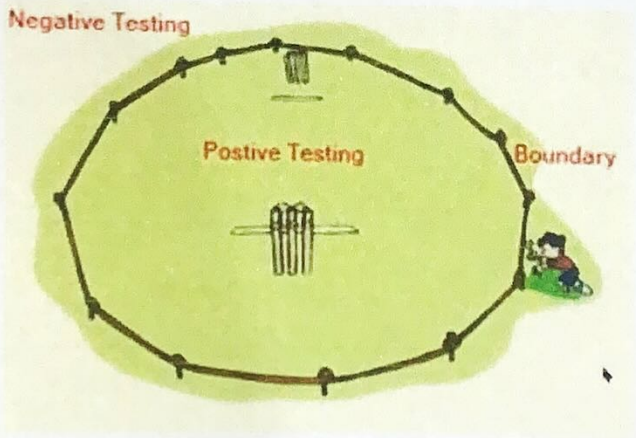
Gambar 6.2 Pengujian dengan inputan yang salah

~~Menginput~~ nilai selain dari angka seperti nilai A-Z/a-z pada kolom *input number* maka aplikasi akan memberikan berupa *alert*/pemberitahuan bahwa inputan tersebut *invalid*/salah.

D. Boundary Value Analysis

Teknik pengujian di mana kasus uji dirancang untuk memasukkan nilai-nilai pada batas. Pengujian positif pengujian berupa data input berada dalam batas nilai batas. Pengujian negatif adalah pengujian berupa data *input* berada di luar batas nilai batas. Analisis dari Nilai Batas didasarkan pada pengujian nilai batas partisi yang *valid* dan *invalid*. Perilaku di tepi partisi ekivalensi lebih cenderung salah daripada perilaku di dalam partisi *valid*, jadi batas adalah area di mana pengujian cenderung menghasilkan cacat.

Pengujian ini akan memeriksa nilai *input* di dekat batas yang memiliki peluang kesalahan lebih tinggi. Setiap partisi memiliki nilai maksimum dan minimum, ~~dan~~ nilai maksimum dan nilai minimum ini adalah nilai batas dari sebuah partisi. Adapun ilustrasi dari pengujian nilai batas dapat dilihat pada gambar 6.3.



Gambar 6.3 Ilustrasi pengujian nilai batas

Dari ilustrasi tersebut dapat diketahui bahwa pengujian nilai batas adalah proses uji di batas antara *positive testing* dengan *negative testing*. Nilai batas untuk partisi yang *valid* adalah

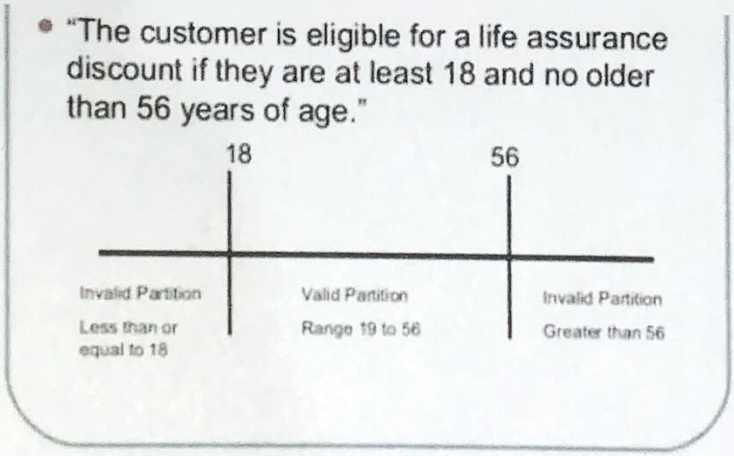
nilai batas yang *valid*. Oleh karena itu, pengujian *boundary value analysis* melakukan pengujian untuk setiap variabel mulai dari : nilai minimal, tepat di atas minimal, nilai nominal, tepat di bawah nilai maksimal, dan nilai maksimal.

Contoh dari pengujian nilai batas yaitu: Jika sebuah sistem hanya menerima nilai inputan untuk umur ~~hanya~~ mulai dari 18 sampai 56. Dapat diketahui bahwa nilai minimal dari umur tersebut adalah 18, dan nilai maksimal dari umur adalah 56. Oleh karena itu pengujian batas sebagaimana tabel 6.1.

Tabel 6.1 Contoh pengujian nilai batas

Invalid (min-1)	***Valid*** (min, min+1, nominal, max-1, dan max)	***Invalid*** (max+1)
17	18, 19, 37, 55, 56	57

Valid Test Cases: Pengujian yang *valid* dari nilai umur 18 sampai 56 terletak di atas nilai 17 dan di bawah nilai 57. Yaitu *valid* jika menginput nilai 18, 19, 37, 55, 56. ***Invalid Test Cases***: Ketika nilai di bawah nilai 18 dan di atas nilai 56 maka nilai akan *invalid*. Adapun ilustrasi dari permasalahan di atas sebagai berikut.



Gambar 6.4 Contoh pengujian nilai batas

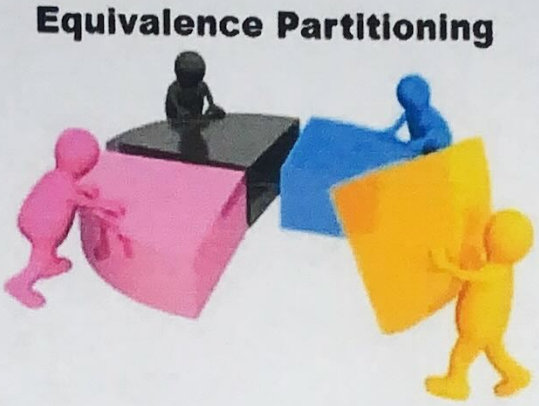
E. Equivalence Partitioning

ecp

Equivalence partitioning juga dikenal sebagai *Equivalence Class Partitioning* (ECP). ~~Ini~~ adalah teknik pengujian perangkat lunak atau pengujian *black-box* yang membagi domain *input* ke dalam kelas data, ~~dan~~ dengan bantuan kelas data ini, kasus uji dapat diturunkan. Kasus

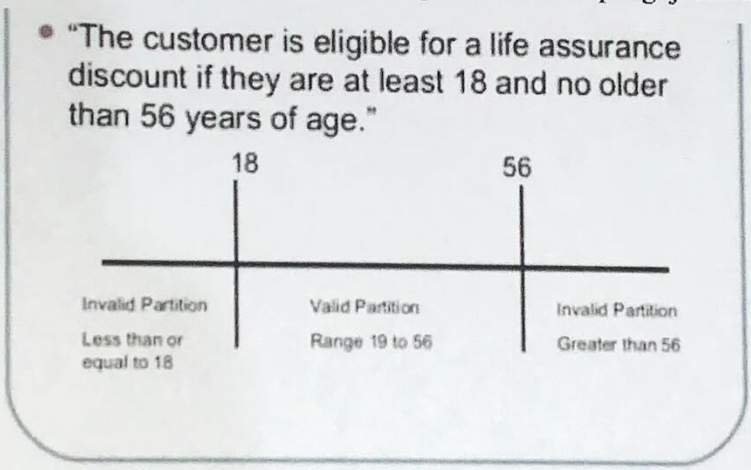
uji yang ideal mengidentifikasi kelas kesalahan yang mungkin memerlukan banyak kasus uji arbitrer(tidak tetap) untuk dieksekusi sebelum kesalahan umum diamati.

Dalam partisi kesetaraan, kelas kesetaraan dievaluasi untuk kondisi *input* yang diberikan. Setiap kali *input* diberikan, maka jenis kondisi *input* diperiksa, kemudian untuk kondisi *input* ini, kelas ekuivalensi mewakili atau menggambarkan kumpulan status *valid* atau *invalid*. Adapun ilustrasi dari *equivalence partitioning* digambarkan pada gambar 6.5.



Gambar 6.5 Ilustrasi *equivalence partitioning*

Teknik pengujian yang membagi data *input* menjadi banyak partisi. Nilai dari setiap partisi harus diuji setidaknya sekali pengujian. Partisi dengan nilai yang *valid* digunakan untuk pengujian positif, dan partisi dengan nilai *invalid* digunakan untuk pengujian negatif. Contoh:



Gambar 6.6 Ilustrasi kasus uji

Gambar 6.6 mengilustrasikan kasus pada *boundary value analysis*. Pengujian secara *equivalence partitioning* adalah pengujian yang akan membagi tiap-tiap nilai ke dalam kelompok masing. Contoh dari nilai kasus di atas yaitu minimal = 18 dan maksimal = 56. Maka terdapat 3 partisi yang dapat dibagi yaitu : 0-18, 19-56-, 56+. Contoh nilai inputan : 5, 40, dan 76 dapat diambil dari setiap bagian partisi untuk menguji setiap skenario yang ada.

BAB VII
WEB ARCHITECTURE

A. Modern Web Architecture Explained

Arsitektur pengujian adalah ilmu disiplin melihat aliran pengiriman dan mencari tahu apa, bagaimana, dan kapan pengujian untuk mencapai hasil terbaik. Pengujian ini menghasilkan kerangka kerja aktivitas pengujian yang kemudian dapat dicakup, disiapkan, dan dieksekusi.

Pengujian web arsitektur terbagi atas dua uji, yaitu secara *Front-end* dan *Back-end*. Pengujian secara *Front-end* meliputi uji aplikasi dari bagian *user* yang melakukan interaksi secara langsung. Contoh:

Gambar 7.1 Ilustrasi interaksi *user*

Pada gambar 1.7 dapat diketahui bahwa *user* adalah pada bagian *client*, dan *Server Side* adalah bagian yang tidak diketahui oleh *user* atau biasa disebut dengan *Back-end* aplikasi. Pada bagian *Server Side, user* tidak akan mengetahui proses logika dari aplikasi, *database*, dan hal sensitif lainnya.

Web Architecture bekerja secara *front-end* dan *back-end*. Adapun proses yang terjadi selama *user* melakukan *request* untuk membuka sebuah web terbagi atas 2 proses, yaitu proses pada bagian *user* yang disebut *front-end* dan proses pada bagian yang tidak dapat dilihat oleh *user* disebut *back-end*.

Frond-end :

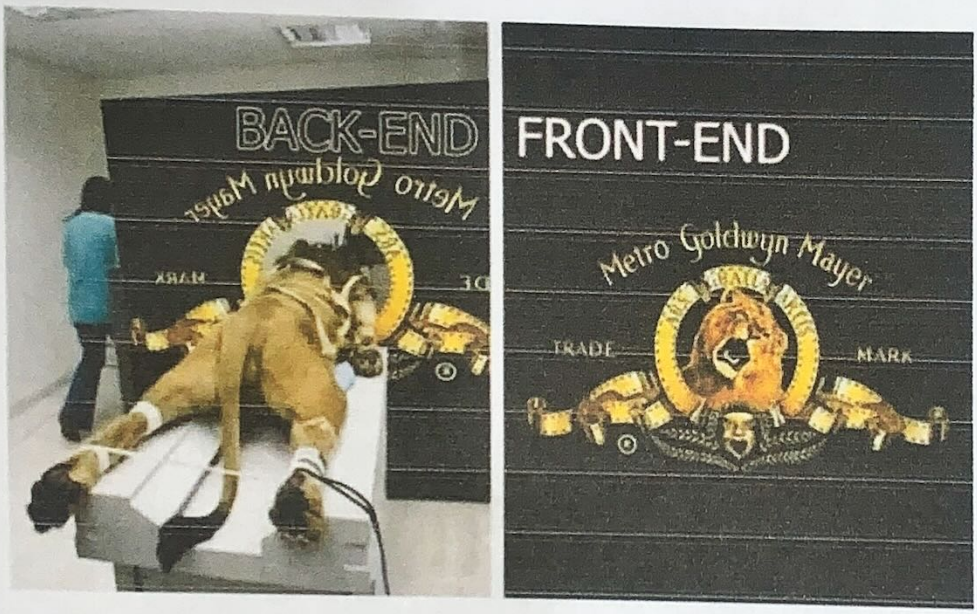
- *Browser* : *User* mengakses halaman web (HTTP/HTTPS Protocol)

Back-end :

- *Back-End* : Mengirim kembali sebagai HTML (Bahasa Standar untuk web *browser*)
- *Browser* : Menerjemahkan halaman web
- *Browser: Download* seluruh *asset* yang dibutuhkan seperti *script, font*, gambar Menghubungi pihak ketiga dari *back-end* seperti (metode pembayaran, *analytics, maps*)
- *Browser* : Menampilkan halaman web ke *user*

B. Back-End Testing

Pengujian *back-end* adalah pengujian yang dilakukan dari segi logika atau segi yang tidak dapat dilihat oleh *user*. Pengujian dari segi *back-end* biasanya melibatkan pengujian seperti *Web Service Testing, Database Testing,* dan *Server Side Testing*. Adapun analogi dari *back-end* dan *front-end* sebagaimana gambar 7.2.



Gambar 7.2 Ilustrasi *back-end* dan *front-end*

Dari gambar 7.2 dapat diketahui bahwa *Back-end* adalah sebuah hal yang dilakukan untuk membuat *Front-end* sesuai dengan yang diinginkan. Seperti pada gambar 7.2, *developer* ingin menampilkan singa mengaung, tentunya pada segi *back-end* atau belakang panggung menunjukkan bahwa singa tersebut diikat dan hanya menampilkan kepalanya saja.