

Perancangan Kasus Pengujian

PERANCANGAN KASUS PENGUJIAN

- **Menurut Berard [BER93] :**
 - Setiap kasus uji diidentifikasi secara unik dan diasosiasikan secara eksplisit dengan kelas yang akan diuji.
 - Tujuan pengujian harus dinyatakan
 - Daftar tahapan pengujian harus dibuat untuk setiap pengujian :
 - daftar keadaan dari objek yang diuji
 - daftar pesan dan operasi yang muncul
 - daftar exception yang mungkin muncul saat objek diuji
 - daftar kondisi eksternal
 - informasi tambahan yang membantu dalam memahami pengujian atau mengimplementasikan pengujian.

PERANCANGAN KASUS PENGUJIAN (2)

- **Implikasi Konsep Berorientasi Objek :**
 - Enkapsulasi menyebabkan informasi dari status objek yang sedang diuji sulit diperoleh
 - Inheritance menyebabkan perlu dilakukannya pengujian setiap reuse (jika subclass digunakan dalam konteks yang berbeda dengan superclass-nya)
 - Multi inheritance menyebabkan penambahan konteks yang harus diuji
- ***Applicability* Metoda Perancangan Kasus Uji Konvensional**
 - White Box → pengujian operasi
 - *basis path, loop testing* , atau *data flow* → untuk memastikan bahwa setiap pernyataan dalam operasi telah diuji
 - Black Box → untuk menguji sistem
 - Use case → untuk membuat input dalam perancangan *black box* dan pengujian *state-based*

PERANCANGAN KASUS PENGUJIAN (3)

- **Pengujian *Fault-Based* :**
 - Untuk merancang pengujian yang mempunyai kemungkinan besar untuk menemukan kesalahan yang masuk akal
 - Perancangan awal perlu pengujian ini dalam tahap Analisis
 - Efektivitas pengujian tergantung dari cara pandang penguji dalam melihat kesalahan
 - Pengujian integritas → mencari kesalahan yang masuk akal dalam *message connection*. Ada 3 jenis kesalahan yang dapat muncul dalam konteks ini :
 - hasil yang tidak diharapkan
 - kesalahan penggunaan message/operasi
 - pemanggilan yang salah

PERANCANGAN KASUS PENGUJIAN (4)

- **Pengaruh OOP pada Pengujian :**

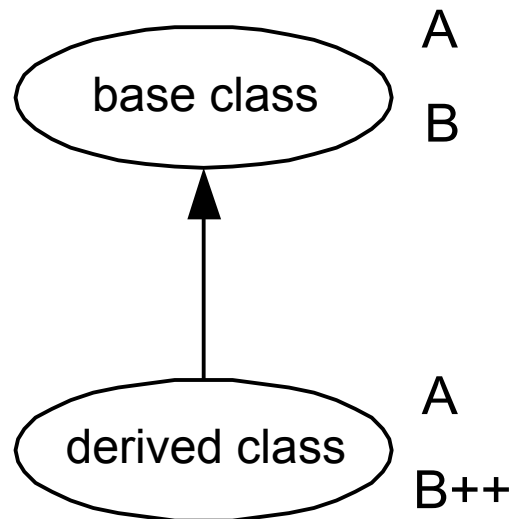
- beberapa jenis kesalahan menjadi kurang masuk akal
- beberapa jenis kesalahan menjadi lebih masuk akal
- muncul beberapa tipe kesalahan yang baru

– Contoh :

karena operasi OO biasanya lebih kecil, ada kecenderungan untuk lebih diperlukan adanya integrasi. Dalam hal ini kesalahan integrasi lebih masuk akal.

PERANCANGAN KASUS PENGUJIAN (5)

- **Kasus Pengujian dan Hirarki dari Kelas :**
 - *Inheritance* memerlukan adanya kebutuhan untuk menguji semua kelas turunan → memperumit proses pengujian



- Fungsi B++ harus diuji ulang
- Fungsi A ?

PERANCANGAN KASUS PENGUJIAN (6)

- Jika fungsi A memanggil fungsi B, dan kelakuan dari B telah berubah, maka A harus diuji ulang walaupun perancangan dan kodenya tidak berubah.
 - Jika fungsi A tidak bergantung pada fungsi B maka fungsi A tidak perlu diuji ulang
- B dan B++ → dua operasi yang berbeda dengan spesifikasi dan implementasi yang berbeda. Pengujian baru perlu diturunkan hanya untuk kebutuhan fungsi B++ yang tidak dipenuhi oleh pengujian fungsi B.
- Pengujian untuk B dapat diterapkan pada objek dari kelas B++. Input pengujian mungkin cocok untuk kedua kelas, tapi hasil yang diharapkan dapat berbeda.

PERANCANGAN KASUS PENGUJIAN (7)

- **Perancangan Pengujian Scenario-Based**

- Pengujian *Fault-based* mengabaikan dua kesalahan utama :
 - Kesalahan spesifikasi
 - Kesalahan interaksi antar subsistem
- Pengujian *scenario-based* memusatkan pada apa yang dilakukan oleh user, bukan pada apa yang dilakukan produk
- *Use cases* digunakan untuk menentukan apa yang dilakukan oleh user kemudian menerapkannya sebagai pengujian
- Pengujian *scenario-based* cenderung untuk memeriksa banyak subsistem dalam suatu pengujian tunggal.

PERANCANGAN KASUS PENGUJIAN (7)

- Contoh perancangan pengujian *scenario-based* untuk text editor:

Use Case : *Fix the Final Draft*

- **Background** : Use case ini menggambarkan urutan *event* yang muncul
 1. Cetak seluruh dokumen
 2. Periksa dokumen, ubah halaman tertentu
 3. Untuk setiap halaman yang diubah, cetak halaman tersebut
 4. Kadang-kadang sederetan halaman dicetak
- **Kebutuhan user** :
 - metoda untuk mencetak satu halaman
 - metoda untuk mencetak beberapa halaman berurutan
- **Yang diuji** : pengeditan setelah pencetakan
- **Penguji berharap** untuk menemukan bahwa fungsi pencetakan menyebabkan kesalahan dalam fungsi pengeditan

PERANCANGAN KASUS PENGUJIAN (8)

- **Testing Surface Structure dan Deep Structure**

- **Surface Structure :**

- struktur yang terlihat secara eksternal dari program berorientasi objek → struktur yang langsung berhubungan dengan *end user*
 - menggunakan daftar semua objek dari semua objek sebagai *checklist* pengujian
 - Perancangan kasus uji harus menggunakan objek dan operasinya sebagai petunjuk yang menuntun pada task yang terabaikan

- **Deep Structure :**

- detail teknis internal dari program berorientasi objek → struktur yang dimengerti dengan memeriksa perancangan dan/atau kode
 - dirancang untuk memeriksa ketergantungan, kelakukan, dan mekanisme komunikasi yang telah dibuat

METODA PENGUJIAN PADA LEVEL KELAS

- **Random Testing**

Contoh : Aplikasi perbankan mempunyai kelas **account** yang dengan operasi : *open, setup, deposit, withdraw balance, summarize, creditLimit*, dan *close*.

– Sejarah hidup minimum dari **account** adalah operasi :

open • setup • deposit • withdraw • close

– Variasi yang mungkin muncul :

open • setup • deposit • [deposit | withdraw | balance | summarize | creditLimit]ⁿ • withdraw • close

– Kasus uji lain yang mungkin :

*open•setup•deposit•deposit•balance•summarize
•withdraw•close*

METODA PENGUJIAN PADA LEVEL KELAS (2)

- **Partition Testing**

- mengurangi jumlah kasus uji dengan mengelompokkan input dan output, kasus uji dirancang untuk memeriksa setiap kelompok

- Ada beberapa cara :

- 1. Pembagian berdasarkan status (*state-based partitioning*)**

- mengelompokkan operasi berdasarkan kemampuan untuk mengubah status dari kelas

- Contoh pada kelas **account** :

- operasi yang mengubah status adalah *deposit* dan *withdraw*

- operasi yang tidak mengubah status adalah *balance*, *summarize*, dan *creditLimit*

METODA PENGUJIAN PADA LEVEL KELAS (2)

2. Pembagian berdasarkan atribut (*attribute-based partitioning*)

- mengelompokkan operasi berdasarkan atribut yang digunakan

Contoh :

- operasi yang menggunakan *creditLimit*,
- operasi yang mengubah *creditLimit*
- operasi yang tidak menggunakan atau mengubah *creditLimit*.

3. Pembagian berdasarkan kategori (*category-based partitioning*)

- mengelompokkan operasi berdasarkan fungsi generik dari setiap operasi

Contoh :

- operasi inisialisasi (*open, setup*)
- operasi perhitungan (*deposit, withdraw*)
- operasi query (*balance, summarize, creditLimit*)
- operasi terminasi (*close*)

Perancangan Kasus Pengujian Antar Kelas

- dimulai pada saat pengintegrasian sistem OO
- dapat dilakukan dengan menerapkan metoda acak dan partisi, pengujian *scenario-based* dan *behavioral*

- **Pengujian Multiple Class**

Kirani dan Tsai [KIR94] menyarankan tahapan sebagai berikut :

1. Untuk setiap kelas *client*, gunakan daftar operator untuk membuat sederetan pengujian acak.
2. Untuk setiap pesan yang dibangkitkan, tentukan kelas kolaborator dan operator yang berhubungan pada objek *server*.
3. Untuk setiap operator pada objek *server*, tentukan pesan yang dikirimkan
4. Untuk setiap pesan, tentukan operator level berikutnya yang dibangkitkan dan masukkan ke dalam urutan pengujian.

Perancangan Kasus Pengujian Antar Kelas (2)

- **Pengujian yang Diperoleh dari Behavior Model**
 - *State-transition diagram* (STD) menyatakan kelakuan dinamik dari kelas .
 - STD dapat digunakan untuk membantu dalam mendapatkan urutan pengujian yang akan memeriksa kelakuan dinamik dari kelas dan kelas-kelas yang berkolaborasi dengannya
 - Untuk situasi di mana kelakuan kelas menghasilkan suatu kolaborasi dengan satu kelas atau lebih, gunakan *multiple* STD untuk menelusuri aliran kelakuan dari sistem.
 - Penelusuran status dapat dilakukan dengan menggunakan metoda *breadth first* → kasus uji memeriksa sebuah transisi tunggal dan ketika transisi baru diuji, hanya transisi yang sudah diuji yang boleh digunakan.