

## PERANCANGAN PENGUJIAN PERANGKAT LUNAK BERORIENTASI OBYEK: BERBASIS STATUS (*STATE-BASED TESTING*)

*Retno Hendrowati*

**D**alam pengembangan perangkat lunak, tahapan pengujian (*testing*) merupakan proses yang penting dalam menentukan tingkat kebenaran perangkat lunak. Pengujian perangkat lunak merupakan aktifitas yang sangat mahal dan dapat menghabiskan waktu. Jika proses pengujian dapat dilakukan secara otomatis, maka efisiensi pengujian akan meningkat dan biaya pengembangan perangkat lunak dapat dikurangi. Oleh karena itu pengujian otomatis harus dirancang dengan baik agar dapat menemukan klasifikasi kesalahan secara sistematis dan dapat diperbaiki dalam waktu dan usaha yang minimal.

Teknologi perangkat lunak berorientasi objek telah meningkat dengan cepat dalam hal perancangan dan pemrograman. Saat ini banyak penelitian dalam teknologi informasi dalam rangka pengembangan perangkat lunak terfokus pada tahapan pengujian.

Pengujian terbagi atas sejumlah aktifitas yaitu pengujian unit, pengujian integrasi dan pengujian sistem. Pengujian unit difokuskan pada pengujian bagian terkecil (blok) program. Pengujian integrasi menguji kebenaran interaksi antara bagian-bagian sistem perangkat lunak yang diuji. Proses pengujian ini mengkombinasikan dan menguji bagian-bagian sistem secara bersama-sama. Pengujian sistem dilaksanakan ketika sistem telah di-*install* sesuai dengan *platform* dan digunakan untuk menunjukkan apakah sistem sesuai dengan kebutuhan dan sesuai dengan objektivitasnya atau bukan. Teknik pengujian dikelompokkan atas pengujian kotak putih (*white box testing*) dan pengujian kotak hitam (*black box testing*). Pengujian kotak

putih bertujuan untuk menguji struktural program, sedangkan pengujian kotak hitam bertujuan untuk menguji fungsional perangkat lunak.

Dalam pengujian berorientasi objek, unit terkecil adalah objek dan *class*, sistem merupakan sekumpulan komunikasi-komunikasi antar objek. *Class* sering dirancang untuk menerima urutan pesan-pesan tertentu yang mengakibatkan respon *class* terhadap pesan-pesan tersebut menjadi berbeda-beda. Hal ini selanjutnya disebut dengan perilaku (*behavior*) *class*. Perilaku tersebut dapat dikendalikan dengan nilai yang ter-enkapsulasi, urutan pesan atau keduanya. Masalah yang dapat muncul adalah bagaimana menguji perilaku *class* terhadap semua kombinasi nilai-nilainya dan apakah kombinasi tersebut benar. Pada tulisan ini akan dibahas bagaimana pengujian berbasis status (*State-Based Testing /SBT*), dengan alasan SBT menggunakan model transisi-status dapat mencakup semua objek yang ada, namun model menjadi kompleks karena mencakup semua karakteristik *class*.

### **Pengujian Perangkat Lunak Berorientasi Objek**

Pengujian adalah suatu proses pengekseskuan program yang bertujuan untuk menemukan kesalahan (Berard, 1994). Pengujian sebaiknya menemukan kesalahan yang tidak disengaja dan pengujian dinyatakan sukses jika berhasil memperbaiki kesalahan tersebut. Selain itu, pengujian juga bertujuan untuk menunjukkan kesesuaian fungsi-fungsi perangkat lunak dengan spesifikasinya.

Pengujian dapat dikategorikan atas :

1. Pengujian terhadap proses pengembangan sistem dan dokumen-dokumen pendukung. Proses berarti sejumlah aktivitas yang didukung oleh dokumen yang mendeskripsikan aktivitas-aktivitas.
2. Pengujian terhadap analisis dan model perancangan. Dalam sistem berorientasi objek, pengujian model analisis dan perancangan adalah hal yang sangat penting.

3. Pengujian secara statik dan dinamik untuk implementasi. Tujuannya adalah mencari kesalahan sedini mungkin dalam proses, tetapi kesalahan dalam kode untuk sistem yang besar dan kompleks tidak dapat dihindarkan. Pengujian statik merupakan inspeksi kode untuk menemukan kesalahan *logic*. Pengujian dinamik merupakan eksekusi dengan data uji untuk menemukan kesalahan dalam kode.

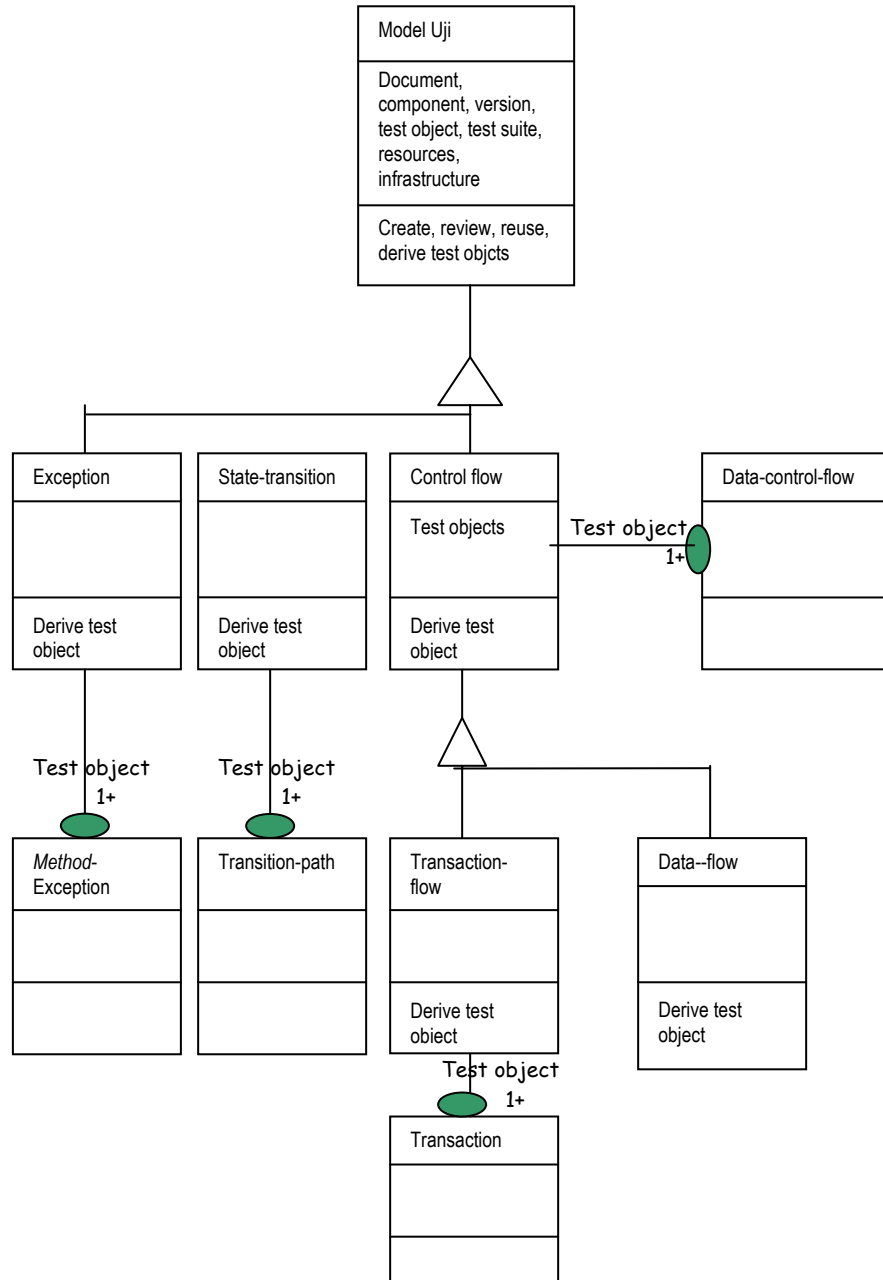
Perangkat lunak berorientasi objek berbeda dari perangkat lunak *procedural* (konvensional) dalam hal analisis, perancangan, struktur dan teknik-teknik pengembangannya. Bahasa pemrograman berorientasi objek mempunyai ciri-ciri adanya pembungkusan (*encapsulation*), keanekaragaman (*polymorphism*), dan pewarisan (*inheritance*) yang membutuhkan dukungan pengujian tertentu.

Fokus pengujian perangkat lunak berorientasi objek dimulai pada hasil analisisnya, dilanjutkan pada hasil perancangannya, dan diakhir pada hasil pemrogramannya (Rochimah, 1997). Model yang dihasilkan pada analisis dan perancangan harus diperiksa terutama dalam hal :

1. *Semantic correctness*, yaitu kesesuaian model dengan domain permasalahan di dunia nyata. Jika model merefleksikan dunia nyata secara akurat, berarti model tersebut benar secara semantik, dan
2. *Consistency*, yaitu kesesuaian kelas dengan objek turunannya maupun kesesuaian asosiasi kelas dengan kelas lainnya.

### **Model Uji (*Test Model*)**

Berbagai model pengujian perangkat lunak berorientasi objek diusulkan oleh para peneliti. Setiap model mempunyai konstruksi atau aturan yang menjadi dasar dalam langkah-langkah pengujian. Dalam pengujian *class*, Siegel (1996) mendefinisikan tiga model fungsional yang digambarkan dalam struktur objek model pengujian dalam gambar 1 berikut ini :



Gambar 1. Model objek untuk "Model Pengujian"

### **State-Transition Model**

Transisi dalam pengujian *method* (operasi) suatu *class* menunjukkan konsep perilaku *class*. Setiap *method* dalam *class* mengekspresikan beberapa elemen dari keseluruhan perilaku *class*. Dalam beberapa metoda perancangan berorientasi objek menggunakan model *State-Transition* untuk merepresentasikan perilaku *class*. *State* (Status) suatu objek adalah kombinasi dari semua nilai atribut. Pada saat tertentu, status bersifat statik. Untuk model dinamik objek, perlu ditambahkan transisi dari satu status ke status lainnya dan terjadilah suatu aksi. Model 'transisi-status' digambarkan dengan grafik, dimana simpul menyatakan status dan busur menyatakan transisi.

Boris Beizer memberikan beberapa aturan untuk pengecekan model "Transisi-status", yaitu :

- a. Verifikasi bahwa status telah merepresentasikan himpunan satu dengan benar;
- b. Cel model untuk semua kemungkinan *event* suatu *class*, yaitu transisi dari setiap status untuk setiap *method* dalam *class* dan cek spesifikasi perancangannya;
- c. Cek terhadap ketetapan satu tansisi untuk setiap kombinasi "event-state". Lakukan pengecekan, misal dengan representasi matriks sebagai kombinasi kemungkinan status dan event;
- d. Cek status yang tidak terjangkau dan status mati, dimana pada status tersebut tidak ada lintasan atau transisi;
- e. Cek aksi yang tidak benar (invalid), termasuk *method* (operasi) yang tidak ada atau *method* yang tidak sesuai dengan kebutuhan selama transisi.

Model "transisi-status" dapat mencakup keseluruhan perilaku *class*, sehingga lebih produktif dibanding dengan model lainnya, namun mempunyai beberapa kelemahan yaitu :

1. Karena model dibentuk dari spesifikasi kebutuhan, bukan dari kode, mudah menyebabkan kesalahan sehingga perlu menguji model seperti halnya menguji kode;
2. Karena model mencakup seluruh perilaku *class* dan *superclass*-nya, maka model menjadi kompleks. Namun pemakaian “transisi-status” secara hirarki dapat mengurangi kompleksitas tersebut;
3. Model perilaku dapat mengakibatkan hilangnya kendali dan data yang salah. Oleh karena itu perlu adanya asumsi yang sama tentang perilaku *class* yang didasarkan pada kebutuhan dengan asumsi yang dibuat oleh pemrogram.

### **Rencana Uji (*Test Plan*)**

Perancangan pengujian dilakukan dengan menyiapkan standar-standar yang harus dipenuhi dalam proses pengujian. Jika tidak sesuai dengan standar, maka proses pengujian telah menemukan kesalahan. Standar-standar tersebut harus terdokumentasi dengan baik. Kasus uji terdiri atas sekumpulan masukan untuk pengujian, sekumpulan kondisi yang harus dieksekusi dan hasil yang diharapkan berdasarkan tujuan yang telah ditetapkan.

Metoda perancangan kasus uji untuk perangkat lunak berorientasi objek yang diusulkan Berrard (dalam Pressman, 1997) adalah :

1. Setiap kasus uji harus diidentifikasi secara unik dan secara eksplisit diasosiasikan dengan *class* yang diuji
2. Tujuan pengujian harus ditetapkan dengan jelas
3. Langkah-langkah pengujian harus dikembangkan untuk setiap kasus uji dan berisi :
  - a. Daftar status tertentu untuk objek yang diuji;
  - b. Daftar pesan (*message*) dan operasi yang akan dieksekusi sebagai akibat pengujian;
  - c. Daftar *exception* yang mungkin terjadi karena objek yang diuji;

- d. Daftar kondisi eksternal (misalnya mengubah lingkungan eksternal perangkat lunak);
- e. Informasi tambahan yang membantu pemahaman dan implementasi pengujian.

### **Teknik Pengujian Berbasis Status (*State-based testing* / SBT)**

Pengujian berbasis status adalah bentuk implementasi pengujian *class*. Hal utama dalam SBT adalah pengujian nilai yang disimpan dalam suatu objek pada suatu saat. Nilai tersebut merepresentasikan status dari suatu objek. SBT juga melakukan validasi interaksi yang terjadi antara transisi dan status suatu objek. *StateChart* dapat digunakan untuk membantu dalam SBT.

Suatu status dapat didefinisikan sebagai nilai vektor [McG-93] :  $V = \langle a_1, a_2, \dots, A_n \rangle$  ; dimana setiap  $a_i$  merupakan nilai *current* dari atribut suatu objek. Himpunan status  $S$  suatu objek adalah produk kartesian dari himpunan status setiap atribut. Jika suatu objek mempunyai dua atribut,  $A$  dan  $B$ , dimana  $A$  adalah Boolean dan  $B = \{-1, 0, 1\}$ , maka himpunan status objek adalah  $\{(T, -1), (T, 0), (T, 1), (F, -1), (F, 0), (F, 1)\}$ .

Perancangan status harus didasarkan pada observasi perilaku objek. Model objek merepresentasikan kemungkinan perilaku objek, atribut dan hubungan antar objek dalam suatu sistem. Status suatu objek adalah kombinasi dari seluruh nilai-nilai atribut yang terkandung dalam objek. Sekelompok nilai-nilai tersebut mempengaruhi perilaku (*behavior*) suatu objek. Perubahan status dapat terjadi sebagai akibat dari antar objek saling mempengaruhi. Hal ini disebut kejadian (*event*). Transisi merupakan perubahan dari satu status ke status lainnya yang disebabkan oleh suatu event. Suatu aksi (*action*) dapat terjadi selama terjadi transisi dari satu status ke status lainnya. Transisi sebagai konsep perilaku *class*. Setiap *method* dari suatu *class* mengekspresikan beberapa elemen secara keseluruhan perilaku *class*.

Pengujian berbasis status terhadap suatu *class* dapat dilihat *method* yang mempengaruhi status dalam empat kemungkinan berikut ini :

1. mengubah status objek ke status baru yang sesuai;
2. berpindah dari status tersebut;
3. mengubah status objek ke status yang tak terdefinisi, berarti terjadi kesalahan;
4. mengubah status objek ke status yang tidak sesuai, berarti terjadi kesalahan.

*State Table* (Tabel Status) merupakan suatu model yang digunakan untuk menggambarkan perilaku sistem. Tabel status disebut juga tabel transisi status, direpresentasikan dengan baris dan kolom sebagai berikut :

- setiap baris menyatakan status;
- setiap kolom menyatakan kondisi masukan/transisi;
- kotak yang merupakan interseksi antara baris dan kolom menentukan status selanjutnya dan keluaran, jika ada.

### **Implementasi Pengujian Berbasis Status**

Implementasi sistem pengujian ini dimulai dengan membangun model objek yang memperlihatkan struktur data statik dari sistem dunia nyatanya (gambar 2).

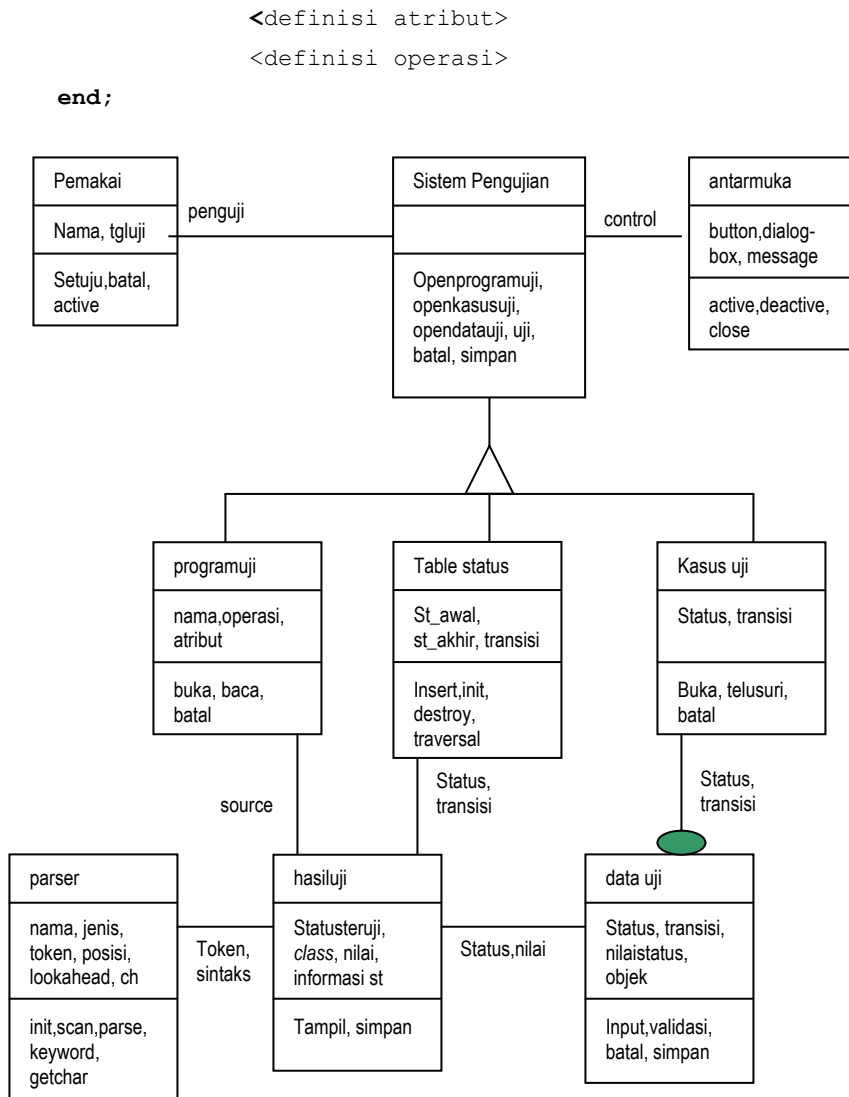
### **Program Uji**

Program uji sebagai file program sumber ditulis dalam bahasa Pascal Objek. Program yang akan diuji tersebut telah benar secara leksik dan sintaks, yang berisi satu definisi *class* lengkap dengan atribut dan operasinya (*method*). Aturan sintaks program uji adalah :

```
<id_class>    = class
               <definisi atribut>
               <definisi operasi>

end;

{Definisi class turunan}
<id_class>    = class<id_superclass>
```



Gambar 2. Model Objek Sistem Pengujian Berbasis Status

### Data Uji dan Kasus Uji

Suatu kasus uji dapat dinotasikan dalam tiga *tuple*, yaitu (Tsai, 1997):  $\langle S_1, t_1 \dots t_n, S_2 \rangle$ ; dimana  $S_1$  adalah *current state* (status awal) dan  $S_2$  adalah *next state* (status akhir) yang akan dicapai setelah mengeksekusi

transisi  $t_1$  (atau urutan *message*,  $t_1..t_n$ ). Dari kasus uji tersebut dapat diidentifikasi data uji.

Pada saat eksekusi pengujian dengan kasus uji, jika status hasil sama dengan status yang seharusnya dicapai, berarti benar, sebaliknya terjadi kesalahan. Jika himpunan kasus uji dituliskan <status awal, transisi, - >, maka ini merupakan kasus khusus dan berarti data uji yang diberikan tidak valid.

### **Studi Kasus**

Pada studi kasus pengujian berbasis status pada program berorientasi objek diberikan deskripsi singkat tentang struktur penyimpanan data dengan STACK berikut ini

Suatu *class* STACK (*of array*) untuk penyimpanan data dengan sifat operasi stack adalah *first-in last-out*. Kondisi-kondisi yang terjadi pada stack adalah kosong (*empty*), penuh (*full*) atau di antaranya (*not full*). Untuk mengakses data dalam stack, operasi-operasi yang ada adalah tambah data (*PUSH/add*) dan hapus data (*POP/delete*). Selain itu, untuk menyatakan bahwa stack penuh maka dapat dilihat dari ukuran stack yang diwakili oleh fungsi *size* yang bernilai Boolean.

Dari deskripsi tersebut dapat ditentukan status-status untuk *class* STACK, yaitu *empty*, *notfull*, dan *full*. Dengan keterangan ukuran STACK adalah :

- *full* ; jika ukuran stack = n (missal ditentukan n adalah ukuran stack maksimum)
- *notfull* ; jiks  $0 < \text{ukuran} < n$
- *empty* ; jika ukuran = 0

Digunakan atribut tambahan *count* untuk menyatakan ukuran relatif data dalam stack.

Dari hasil rancangan *class*, dapat didefinisikan beberapa *method*, yaitu :

- *delete(integer)* : untuk menghapus data dari *stack*

- *add(character, integer)* : untuk menyatakan character apa yang ditambahkan pada *stack* dan berapa ukuran *stack* akibat penambahan data tersebut
- *initStack()* : untuk menyatakan konstruksi objek *stack*
- *isEmpty()* : untuk menyatakan apakah *stack* kosong atau tidak
- *size()* : untuk mengecek ukuran *stack*

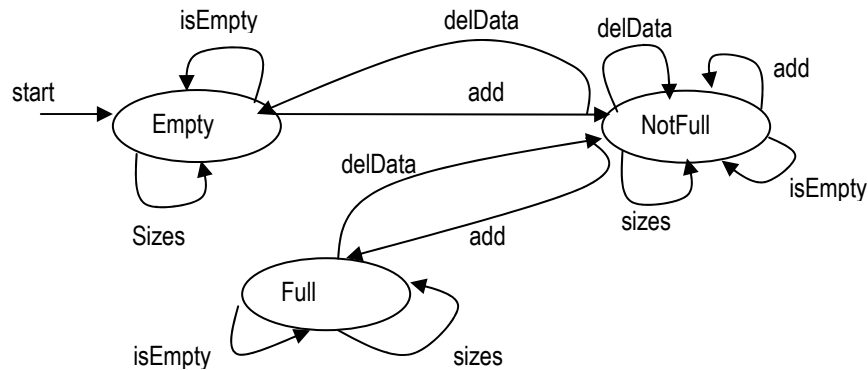
Dari deskripsi tersebut dapat dinyatakan dalam table contoh perubahan status berikut ini

**Tabel 1. Contoh Perubahan Status Objek dari Class STACK**

<i>Current state</i>	Fungsi yang dieksekusi	Status berikut (validasi terhadap nilai status)
--	<i>initStack()</i>	<i>Empty (count = 0)</i>
<i>empty</i>	<i>Add('a', 1)</i>	<i>Notfull (count = 1)</i>
<i>notfull</i>	<i>Add('b', 2)</i>	<i>Notfull (count = 2)</i>
<i>notfull</i>	<i>Delete()</i>	<i>Notfull (count = 1)</i>
<i>notfull</i>	<i>Delete(1)</i>	<i>Empty (count = 1)</i>
<i>empty</i>	<i>Add('c', 1)</i>	<i>Notfull (count=1)</i>
<i>.....dst</i>	<i>.....dst</i>	<i>.....dst</i>

**Tabel 2. Tabel status dari deskripsi sistem**

Status	Transisi			
	<i>size</i>	<i>isEmpty</i>	<i>Del_data</i>	<i>Add</i>
<i>Empty</i>	<i>Empty</i>	<i>Empty</i>	-	<i>NotFull</i>
<i>NotFull</i>	<i>NotFull</i>	<i>NotFull</i>	{ <i>NotFull</i> , <i>Empty</i> }	{ <i>NotFull</i> , <i>Full</i> }
<i>Full</i>	<i>Full</i>	<i>Full</i>	<i>NotFull</i>	-



Gambar 3. Diagram Transisi Status (*state chart*)

### Rencana Uji

1. Tujuan pengujian : menguji perilaku *class* STACK
2. Identifikasi Kasus Uji :
  - a. Daftar Status : *Empty, NotFull, Full*
  - b. Daftar Method : *Add, delData, Sizes, isEmpty*
  - c. Daftar Exception : *<empty,delData,...>; <full,add,...>*

### Himpunan Kasus Uji

```

{<empty, sizes, empty>, <notfull, sizes, notfull>, <full, sizes, full>,
<empty, isEmpty, empty>, <notfull, isEmpty, notfull>,
<full, isEmpty, full>, <empty, delData, ->, <notfull, delData, notfull>,
<notfull, delData, Empty>, <full, delData, notfull>,
<empty, add, notfull>, <notfull, add, notfull>, <full, add, ->}
  
```

### Program Uji Stack

```

Unit programuji;
Const
  Max = 10;
Type
  TStack = class
    Size : integer;
    Tab_char : array[1..Max] of char;
    Top : integer;
    Procedure Constructor;
    Procedure Add(x :string);
    Procedure DelData(var x : string);
  end class;
  
```

```
        Function sizes : integer;
        Function isEmpty : Boolean;
    End;
Var
    S : TStack;
Implementation
    Procedure TStack.Constructor;
    Begin
        Size := max;
        Top := 0;
    End;
    Procedure TStack.Add(x:string);
    Begin
        If top<size then
        Begin
            Top := top+1;
            Tab_char[top]:=x;
        End;
    End;
    Procedure TStack.DelData(var x:string);
    Begin
        If top>0 then
        Begin
            X := tab_char[top];
            Top := top-1;
        End;
    End;
    Function TStack.Sizes:integer;
    Begin
        Sizes := top;
    End;
    Function TStack.isEmpty : Boolean;
    Begin
        If top = 0 then
            isEmpty := true
        else
            isEmpty := false;
        end;
    end.
end.
```

Dari program uji tersebut terlihat bahwa :

- nilai batas ukuran STACK adalah 10 (nilai batas inilah yang dijadikan atribut peubah status *class*)
- *instance* yang diuji adalah S

### Hasil Pengujian

Program uji : programuji.pas

Data uji	Current state	Fungsi eksekusi	Next state	Nilai status	Hasil
S.add('a')	Empty	Add	NotFull	1	OK
S.add('b')	NotFull	Add	NotFull	2	OK
S.add('c')	NotFull	Add	NotFull	3	OK
S.Sizes	NotFull	Sizes	NotFull	3	OK
S.add('d')	NotFull	Add	NotFull	4	OK
S.add('e')	NotFull	Add	NotFull	5	OK
S.add('f')	NotFull	Add	NotFull	6	OK
S.add('g')	NotFull	Add	NotFull	7	OK
S.add('h')	NotFull	Add	NotFull	8	OK
S.add('i')	NotFull	Add	NotFull	9	OK
S.add('j')	NotFull	Add	NotFull	10	OK
S.add('k')	NotFull	Add	NotFull	10	FAIL

Dari table hasil pengujian tersebut bahwa :

- jika pengujian benar, yaitu benar sintaks dan nilai atribut untuk uji (nilai status), maka hasil uji adalah OK;
- jika pengujian benar sintaks, namun salah nilai atribut, maka menghasilkan FAIL;
- jika terjadi kesalahan dalam pengisian data uji, maka proses pengujian gagal dan tidak menghasilkan tabel pengujian. Kesalahan data uji tersebut meliputi
  - o salah sintaks;
  - o salah nama objek yang diuji;
  - o salah nama *method*.

### Penutup

Dari uraian di atas dapat dinyatakan bahwa pengujian berbasis status pada perangkat lunak berorientasi objek dilakukan dengan aturan-aturan dalam representasi data uji yang didasarkan pada kasus uji. Perilaku status yang direpresentasikan dalam transisi status sangat membantu dalam mendeskripsikan perilaku *class* berdasarkan kondisi-kondisi dari nilai *instance*-nya.

Pengujian berbasis status ini termasuk dalam pengujian unit dan semi statik-dinamik. Dinamik karena penguji dapat memberikan deskripsi transisi status sesuai dengan sistem yang akan diuji. Statik, karena implementasinya masih terbatas pada satu *class* program uji (yaitu STACK). Hal ini dikarenakan rumit dan kompleksnya interpretasi setiap *method* dengan nilai parameter yang akan diuji.

## **Daftar Pustaka**

- Beizer, Boris. 1990, *Software Testing Techniques*, 2<sup>nd</sup> Edition, New York: Van Nostrand Reinhold
- Berard, Edward V. 1994, *Issues in the Testing of Object-Oriented Software*, Berard Software Engineering Inc., article at [www.toa.com](http://www.toa.com).
- Pacheco, Xavier. 1995, *Borland Delphi Developer's Guide*, First Edition, New York: Sam Publishing
- Pressman, Roger S. 1997, *Software Engineering a Practitioner's Approach*, 4<sup>th</sup> edition, Singapore: Mc.Graw Hill Inc.
- Robert V. Binder, 1995, *Testing Object : State-based testing, The Approach for sistem testing*, Chicago: Object Magazine, RSBC corp.
- Rochimah, Siti. 1997, *Penerapan Method Berarah Objek pada Kasus Penjadwalan*, Bandung: Institut Teknologi Bandung
- Rumbaugh, James, et al. 1991, *Object Oriented Modelling and Design*, New York: Prentice-Hall International
- Siegel, Shel. 1996, *Object Oriented Software Testing an Hierarchical Approach*, Canada: John Wiley & Sons Inc.
- Tsai, et al. 1997, *A Method for Automatic Class Testing (MACT) Object Oriented Program Using A State-based Testing Method*, Fifth European Edinburg: Conf. Software Testing Analysis and Review