PENGUJIAN PERANGKAT LUNAK

-STRATEGI BLACK BOX TESTING-

Eka Widhi Yunarso (EWD)
2017-2

Digunakan dilingkungan internal prodi D3 Manajemen Informatika, Fakultas Ilmu Terapan, Universitas Telkom

TECHNIQUES FOR TESTING - DYNAMIC

Basis Path Testing

- a white-box testing technique, proposed by Tom McCabe, 1976
- to derive a logical complexity measure of a procedural design, and use this measure as a guide for defining a basis set of execution paths
- test cases derived to exercise every *statement and branch* in the program at least once during testing (statement/branch coverage)
- if every condition in a compound condition is considered, *condition* coverage can be achieved
- Steps:
  - Draw a (control) *flow graph*, using the flowchart or the code
  - Calculate the *cyclomatic complexity*, using the flow graph
  - Determine the *basis set* of linearly independent paths
  - Design *test cases* to exercise each path in the basis set

Telkom University

STRATEGI BLACK BOX TESTING

Pada kajian ini akan dibahas mengenai strategi pengujian *Black Box*, antara lain

1. *customer requirements,*
2. *equivalence class,*
3. *boundary value analysist (BVA),*
4. *use case testing.*

Telkom University

TEST OF CUSTOMER REQUIREMENTS

1. We begin by **looking at each customer requirement**.
2. To start, we want to **make sure** that **every single customer requirement has been tested at least once**.
3. As a result, we can **trace every requirement** to its test case(s) and every test case **back to its stated customer requirement**.

Telkom University

TEST OF CUSTOMER REQUIREMENTS

Requirement: ***When a user lands on the "Go to Jail" cell, the player goes directly to jail, does not pass go, does not collect $200. On the next turn, the player must pay $50 to get out of jail and does not roll the dice or advance. If the player does not have enough money, he or she is out of the game.***

There are many things to test in this short requirement above, including:

1. Does the player get sent to jail after landing on "Go to Jail"?
2. Does the player receive $200 if "Go" is between the current space and jail?
3. Is $50 correctly decremented if the player has more than $50?
4. Is the player out of the game if he or she has less than $50?

Telkom University

Telkom University

TEST OF CUSTOMER REQUIREMENTS

Test ID	Description	Expected Results	Actual Results
5	Precondition: Game is in test mode. Number of players: 1 Money for player 1: $1200 Player 1 dice roll: 3 Player 1 clicks "End Turn" button.		
		Player 1 is sent to jail Only "Get Out of Jail" button is enabled for Player 1.	
	Player 1 clicks "Get Out of Jail" button.		
		Money for Player 1: $1150	
6	Precondition: Game is in test mode. Number of players: 2 Money for player 1: $1200 Money for player 2: $1200 Player 1 dice roll: 3 Player 1 clicks "End Turn" button.		
		Player 1 is sent to jail	
	Player 2 dice roll: 2 Player 2 clicks "End Turn" button.		
		Only "Get Out of Jail" button is enabled for Player 1.	
	Player 1 clicks "Get out of Jail" button.		
		Money for Player 1: $1150	

Telkom University

EQUIVALENCE CLASS TESTING

Merupakan teknik yang digunakan untuk **mengurangi jumlah *test case*** yang ada pada saat pengujian. Kebanyakan *tester* menggunakan teknik yang simpel ini meskipun secara formal *tester* tersebut tidak mengetahui mengenai metode desain formal dalam pengujian perangkat lunak.

Kasus uji yang didesain untuk *Equivalence class testing* berdasarkan pada evaluasi dari ekuivalensi jenis/*class* untuk kondisi *input*. ***Class-class* yang ekuivalen merepresentasikan sekumpulan keadaan valid dan invalid untuk kondisi *input***. Biasanya kondisi *input* dapat berupa spesifikasi nilai numerik, kisaran nilai, kumpulan nilai yang berhubungan atau kondisi boolean.

Telkom University

EQUIVALENCE CLASS TESTING

Sebagai **contoh** misalkan kita diberikan kasus mengenai sebuah modul untuk sistem *Human Resource Development* (HRD) untuk penerimaan pegawai baru berdasarkan usia. Dengan *rule*/ aturan sebagai berikut:

0–16	Don't hire
16–18	Can hire on a part-time basis only
18–55	Can hire as a full-time employee
55–99	Don't hire[*]

Telkom University

EQUIVALENCE CLASS TESTING

0–16	Don't hire
16–18	Can hire on a part-time basis only
18–55	Can hire as a full-time employee
55–99	Don't hire[*]

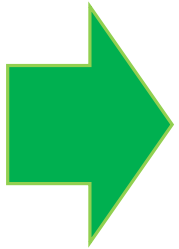
Berdasarkan kasus yang diberikan ini seharusnya kita menguji modul tersebut dengan data uji usia dengan nilai 0, 1, 2, 3, 4, 5, 6, 7, 8, ..., 90, 91, 92, 93, 94, 95, 96, 97, 98, 99. Hal ini mungkin dilakukan jika kita memiliki banyak waktu luang dan memiliki sisa energi yang cukup. Contoh berikut ini adalah contoh yang simple untuk mengimplementasikan *rule* pada contoh sebelumnya.

```
If (applicantAge == 0) hireStatus="NO";
If (applicantAge == 1) hireStatus="NO";

If (applicantAge == 14) hireStatus="NO";
If (applicantAge == 15) hireStatus="NO";
If (applicantAge == 16) hireStatus="PART";
If (applicantAge == 17) hireStatus="PART";
If (applicantAge == 18) hireStatus="FULL";
If (applicantAge == 19) hireStatus="FULL";

If (applicantAge == 53) hireStatus="FULL";
If (applicantAge == 54) hireStatus="FULL";
If (applicantAge == 55) hireStatus="NO";
If (applicantAge == 56) hireStatus="NO";

If (applicantAge == 98) hireStatus="NO";
If (applicantAge == 99) hireStatus="NO";
```

Telkom University

EQUIVALENCE CLASS TESTING

```
If (applicantAge == 0) hireStatus="NO";
If (applicantAge == 1) hireStatus="NO";

If (appl        == 14) hir        "NO";
If (ar          15)       O";
If (ap                    ART";
If (applic                "PART";
If (applicant             tus="FULL";
If (applicant             tus="FULL";

If (ar                    ULL";
If (ap          54)       ULL";
If (appl        == 55) hire       "NO";
If (applica     ge == 56) hireSt  s="NO";

If (applicantAge == 98) hireStatus="NO";
If (applicantAge == 99) hireStatus="NO";
```



```
If (applicantAge >= 0 && applicantAge <=16)
        hireStatus="NO";
If (applicantAge >= 16 && applicantAge <=18)
        hireStatus="PART";
If (applicantAge >= 18 && applicantAge <=55)
        hireStatus="FULL";
If (applicantAge >= 55 && applicantAge <=99)
        hireStatus="NO";
```

Telkom University

EQUIVALENCE CLASS TESTING

Dengan menggunakan pendekatan *equivalence class*, kita dapat **mengurangi jumlah *test case*** pada studi kasus diatas dari 100 *test case* (menguji tiap-tiap usia yang mungkin) menjadi empat *test case* saja (*testing*) hal ini merupakan penghematan yang sangat signifikan.

Range nilai yang **diwakili** oleh **suatu nilai tertentu** yang dijelaskan disini disebut sebagai ***equivalence classes***. *Equivalence class* terdiri dari kumpulan data yang diperlakukan sama oleh sistem atau menghasilkan *output* yang sama. Setiap nilai data dalam sebuah *class* adalah equivalent, secara spesifik kita berharap bahwa:

1. Jika satu *test case* dalam *equivalence class* mendeteksi cacat, semua *test case* yang lain dalam *equivalence class* yang sama kemungkinan akan mendeteksi cacat yang sama.
2. Jika satu *test case* dalam *equivalence class* tidak mendeteksi adanya cacat, maka *test case* yang lain dalam *equivalence class* yang sama masih ada kemungkinan untuk mendeteksi cacat.

Telkom University

TEKNIK EQUIVALENCE CLASS TESTING

1. Identifikasi kelas-kelas yang ekuivalen (*equivalence class*).
2. Buat *test case* untuk tiap-tiap *equivalence class*.
3. Jika memungkinkan buat *test case* tambahan yang acak yang memungkinkan ditemukannya cacat pada perangkat lunak.

Telkom University

BOUNDARY VALUE ANALYSIST (BVA)

Boundary Value Analysist merupakan teknik **desain *testing*** yang **paling dasar**. Itu membantu *tester* untuk **memilih subset yang kecil** untuk membuat *test case* yang mungkin. *Boundary Value Analysist* juga memiliki keuntungan yaitu memandu kita untuk membuat *boundary value testing* yang akan kita pelajari pada bab ini. Pada bahasan yang lalu *rule*/ aturan usia yang diberikan adalah sebagai berikut:

0–16	**Don't hire**
16–18	Can hire on a part-time basis only
18–55	Can hire as a full-time employee
55–99	Don't hire[*]

Sebagai catatan masalah dalam suatu *boundary*/ batas adalah **"edge"** untuk tiap-tiap kelas. Usia 16 tahun masuk kedalam dua equivalent *class* yang berbeda. *Rule* yang pertama mengatakan bahwa jangan menerima calon pegawai yang masih berumur 16 taun. *Rule* yang kedua mengatakan bahwa calon pegawai yang berumur 16 tahun dapat diterima sebagai tenaga kerja paruh waktu.

Telkom University

BOUNDARY VALUE ANALYSIST (BVA)

Boundary value testing focus terhadap batasan-batasan yang simple karena disitulah kebanyakan **cacat perangkat lunak tersembunyi**. *Tester* yang berpengalaman telah mempertimbangkan situasi ini sejak lama. Sedangkan *tester* yang kurang berpengalaman mungkin juga memiliki intuisi untuk mengetahui masalah cacat yang tersembunyi pada *boundary*. Cacat ini bisa terdapat dalam *requirement* yang ditunjukkan di atas maupun pada code yang ditunjukkan di samping:

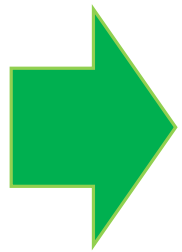
```
If (applicantAge >= 0 && applicantAge <=16)
        hireStatus="NO";
If (applicantAge >= 16 && applicantAge <=18)
        hireStatus="PART";
If (applicantAge >= 18 && applicantAge <=55)
        hireStatus="FULL";
If (applicantAge >= 55 && applicantAge <=99)

        hireStatus="NO";
```

Dapat terlihat bahwa **kesalahan** dilakukan oleh programmer ketika membuat program tersebut menimbulkan masalah dengan menuliskan > (lebih besar) serta ≥ (lebih besar sama dengan) pada contoh diatas.

Telkom University

BOUNDARY VALUE ANALYSIST (BVA)

0–16	Don't hire
16–18	Can hire on a part-time basis only
18–55	Can hire as a full-time employee
55–99	Don't hire[*]



0–15	Don't hire
16–17	Can hire on a part-time basis only
18–54	Can hire as full-time employees
55–99	Don't hire

Telkom University

TEKNIK BOUNDARY VALUE ANALYSIST (BVA)

1. Identifikasi kelas-kelas yang ekuivalen (*equivalence class*).
2. Identifikasi batasan untuk tiap *equivalence class*.
3. Buat *test case* untuk tiap batasan suatu nilai dengan memilih titik pada batasan, satu titik pada nilai bawah batasan dan satu titik pada nilai atas batasan.

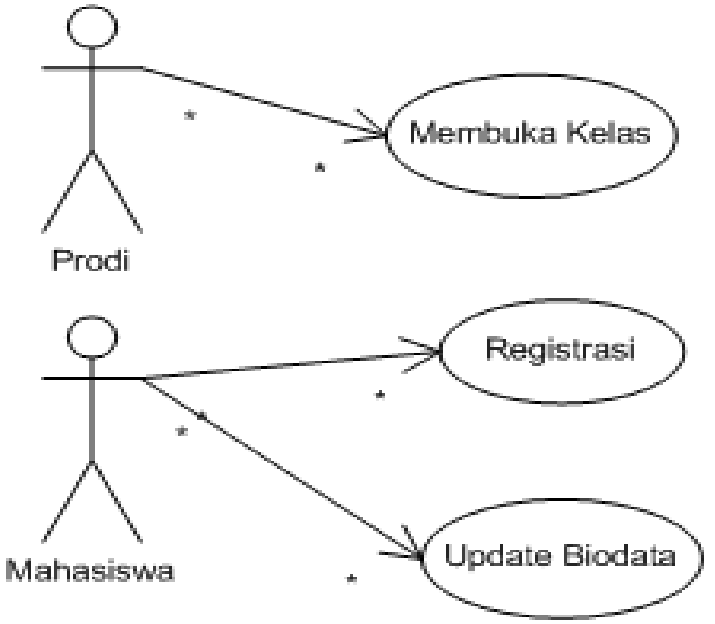
Telkom University

USE CASE TESTING

Mendefinisikan transaksi pada proses-proses yang ada pada suatu sistem adalah hal yang sangat penting untuk dilakukan pada proses pendefinisian kebutuhan sistem (*requirements definition*).

Terdapat banyak pendekatan untuk mendokumentasikan transaksi-transaksi tersebut seperti flowchart, HIPO diagram, dan teks. Sekarang pendekatan yang paling populer dilakukan adalah dengan menggunakan use case diagram.

Use case biasanya dibuat oleh *developer* dan untuk *developer*, namun demikian dalam pengujian perangkat lunak, informasi yang ada pada use case sangat berguna bagi *tester*. Contoh use case:



Telkom University

USE CASE TESTING

Pada use biasanya terdapat symbol yang menggambarkan actor, bentuk elips mendefinisikan use case, dan tanda panah menunjukkan actor menginisiasi suatu use case. Dalam pengembangan berbasis object oriented use case merupakan elemen yang sangat penting dalam mendefinisikan kebutuhan fungsional, lebih dari itu use case juga dapat digunakan dalam paradigma yang lain secara baik.

Fungsi dari use case

1. Menggambarkan *functional requirement* dari sebuah sistem dari perspektif pengguna bukan dari perspektif teknik dan mengabaikan paradigma pengembangan yang digunakan.
2. Dapat digunakan untuk proses identifikasi kebutuhan pengguna.
3. Menyediakan dasar untuk komponen internal sistem, struktur, *database*, dan keterhubungan.
4. Menyediakan dasar dalam membangun *test case* dalam sistem dan *acceptance level*.

Telkom University

CONTOH DESKRIPSI USE CASE

Example use case.		
Use Case Component	Description	
Use Case Number or Identifier	SIM153	
Use Case Name	Registrasi	
Goal in Context		
Scope	System	
Level	Primary task	
Primary Actor	Mahasiswa	
Preconditions	None	
Success End Conditions	Mahasiswa terdaftar dalam kelas untuk matakuliah(kelas diajar oleh dosen)	
Failed End Conditions	Daftar matakuliah mahasiswa yang diambil oleh mahasiswa tidak bisa berubah	
Trigger	Mahasiswa memilih matakuliah dan mengambil matakuliah tersebut.	
Main Success Scenario **A: Actor** **S: System**	Step	Action
	1	S: menampilkan daftar kelas yang dibuka
	2	A: Memilih kelas dan menambahkan kelas
	3	S: melakukan filter terhadap SKS dan Kuota
	4	……
Extensions	2a	Mahasiswa telah selesai melakukan semua proses registrasi S: Mencegah mahasiswa mengulang registrasi dan menampilkan pesan
	4a	Matakuliah yang diambil memiliki pre-requisite S: Mencegah mahasiswa mengambil matakuliah tersebut jika belum mengambil matakuliah pre-requisite dan menampilkan pesan
	4b	Kelas yang diambil penuh S: Mencegah mahasiswa mengambil kelas tersebut.

Telkom University

USE CASE TESTING

Dengan syarat tiap-tiap use case telah benar-benar dipastikan sebelum implementasi. Untuk menguji suatu use case aturan dasar yang harus diperhatikan adalah **membuat minimal satu *test case*** untuk main success scenario dan setidaknya **satu *test case extension*** pada use case.

Karena use case tidak menspesifikasikan data *input*, maka *tester* harus menentukan data *input* tersebut. Secara umum teknik-teknik yang telah dijelaskan sebelumnya seperti *the equivalence class and boundary value techniques*.

Telkom University

TEKNIK USE CASE TESTING

1. Sangat penting untuk mempertimbangkan resiko dari transaksi dan jenis-jenisnya pada saat pengujian. Transaksi-transaksi yang memiliki resiko rendah diuji dengan intensitas rendah sebaliknya transaksi-transaksi yang resikonya tinggi harus diuji dengan intensitas yang tinggi.
2. Untuk membuat *test case*, mulailah dengan data yang normal dan sering digunakan dalam transaksi. Kemudian pilih transaksi-transaksi yang jarang dilakukan tapi merupakan transaksi yang vital bagi sistem, misalkan transaksi mematikan sistem,dll.
3. Pastikan setiap extension pada use case telah diuji. Ujilah dengan kondisi-kondisi dan sesuatu yang ekstrim serta langgarlah ketentuan-ketentuan yang diberikanoleh sistem. Jika suatu transaksi memiliki suatu perulangan/loop, teliti dan cobalah perulangan tersebut berulang-ulang dan jika terdapat alur yang rumit dalam suatu transaksi ujilah transaksi tersebut dengan cara yang ekstrim misalkan uji dengan alur yang tidak seharusnya dilakukan.

Telkom University

CONTOH USE CASE TESTING

Test cases for operation mousePressed
First, the pre and post conditions…
This operation changes the object state by modifying the attributes, x and y. Sometimes operations change the object state of other objects that it uses. In this case, the operation just changes its own object state.

We should ask, what are the valid values for the x and y coordinates? Is there a range, or is any integer value acceptable? We could think of the mouse coordinates generated by the physical mouse relative to our virtual screen as having a valid range defined by the dimensions of the frame. For example, x: [0..100] and y: [0..200]. That is, x can take any integer value from 0 to 100 inclusive. The values for x and y form the input set.

What can we select as representative test cases? What data values are relevant? We don't want to test the operation with every possible input value or variation in object state. That could take days or years! One strategy is to think in terms of *equivalent* sets or classes of data. For each equivalent set, we normally take a test case for a value in the middle. Then test cases for values on and around the boundary of the set.

So…for the mousePressed operation, what are the valid values for x and y? Can we decompose this set into equivalent sets, or can we simply have a single test case? What are the boundary values? The boundary is given by the boundary of the screen. Do we have to test for negative values? This depends on preconditions and possible input values. For this class, it is not possible to have negative (in this case, illegal) values of the x and y coordinates
of the mouse relative to the sketchpad window.

Telkom University

CONTOH USE CASE TESTING

Self-Study Module: Test Case Design

Test case	Event	Input	Expected Result	Description
1	left mouse press (do not release)	x: 10, y: 15	last_x = 10, last_y = 15	Normal case mousePressed
2	left mouse drag (do not release)	x: 80, y: 50	last_x = 80, last_y = 50 line drawn from (10,15) to (80,50)	Normal case mouseDragged
3	left mouse drag	x: 100, y: 0	last_x = 100, last_y = 0 line drawn from (80,50) to (100,0)	Boundary case mouseDragged
4	left mouse drag (then release)	x: 99, y: 199	last_x = 99, last_y = 199 line drawn from (100,0) to (99,199)	Boundary case mouseDragged (close to boundary)

Telkom University

LATIHAN

No	Studi Kasus
1.	Suppose you are writing a simple calculator program. This program can handle positive integer calculation, including addition, subtraction, multiplication, and division. The input is a string composed of digits (0, 1, 2, …, 9) and operators (+, -, *, /). No space is allowed. The input string can be at most 100 characters long, and each number can compose of at most 10 digits. Division of two integers produces one integer by truncation. If the answer contains more than 10 digits, this program simply outputs an overflow error message. Using the **equivalence partitioning and boundary value analysis methods**, derive a set of test cases for the program.

Telkom University

REFERENSI

Boriz Beizer, Control Flow Testing.

Cognizant Technology Solutions, Software Testing, 2010.

Desikan, Srinivasan, dan Gopalaswamy Ramesh. Software Testing: Principles and Practices. Dorling Kindersley, 2006.

Hoyle, David. ISO9000 Handbook 6th Ed. Elsevier Science and Technology, 2009.

IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.

Koirala, S, dan S Sheikh. Software Testing: Interview Questions. Infinity Science Press, 2008.

Limaye, MG. Software Testing: Principles, Techniques and Practice. Tata McGraw-Hill, 2009.

Malik, Kamma. Software Quality, A Practitioner's Approach . New Delhi: Tata McGraw-Hill, 2008.

Patton, Ron. Software Testing (2nd Edition). Sams, 2005.

Pressman, Roger R. Software Engineering, A Practitioner's Approach. Singapore: McGraw-Hill, 2005.

Sharon Robson, White Box Testing, STANZ, 2009.

Telles, Matt, dan Yuan Hsieh. The Science Of Software Debugging. Dreamtech Press, 2004.

Yunarso, Eka Widhi. Student Workbook Jaminan Mutu Sistem Informasi. DeePublish, 2013.

Telkom University