IMPLEMENTASI *AUTOMATED UNIT* DAN *INTEGRATION TESTING* PADA PENGUJIAN PERANGKAT LUNAK (STUDI KASUS APLIKASI PENJUALAN BUKU *ONLINE*)

Agung Setiawan[*)], Kodrat Iman Satoto, and R. Rizal Isnanto

Jurusan Teknik Elektro, Fakultas Teknik, Universitas Diponegoro,
Jl. Prof. Sudharto, SH, Kampus UNDIP Tembalang, Semarang 50275, Indonesia

[*)] *email : blinkawan@gmail.com*

Abstrak

Pertumbuhan bisnis yang semakin pesat pada saat ini selalu diimbangi dengan peningkatan kebutuhan terhadap suatu aplikasi bisnis yang digunakan untuk mempermudah pekerjaan. Untuk mendukung kegiatan bisnis tersebut suatu aplikasi harus dalam keadaan siap yaitu setelah melalui beberapa tahap pengujian. Sebagian besar pengembang yang masih awam, menggunakan cara pengujian tradisional yaitu mencoba satu persatu menu yang ada pada aplikasi ketika aplikasi yang dikembangkan tersebut sudah jadi. Cara seperti ini tidak bisa dilakukan secara berulang-ulang dengan mudah dan juga tidak bisa menguji logika pada method dari suatu kelas. Dalam penelitian ini diberikan solusi dalam melakukan pengujian perangkat lunak pada sebuah sistem e-commerce. Sistem e-commerce yang dimaksud dibangun dengan bahasa pemrograman Java EE menggunakan framework Spring. Framework JUnit akan menangani pengujian kode-kode program dengan menggunakan metode pengujian unit serta pengujian integrasi. Berdasarkan hasil penelitian dapat simpulkan, pertama, dengan menggunakan pengujian automatis sebuah kasus pengujian bisa dieksekusi secara cepat dan berulang-ulang. Kedua, supaya bisa dilakukan pengujian unit, suatu kelas harus menerapkan dependency injection, teknik ini digunakan untuk memasukkan sebuah objek palsu untuk simulasi. Ketiga, berbeda dengan pengujian unit yang bersifat independen, pengujian integrasi menguji integrasi layer data access dengan basisdata.

Kata Kunci : pengujian automatis, pengujian unit, pengujian integrasi, JUnit

Abstract

The rapid growth of the business at the moment is always offset by an increase in the need for a business application that is used to facilitate the work. To support the business activities, an application must be in the ready state after going through several stages of testing. Most of the developers who still lay, using traditional testing method that is try the application menu one by one when the application is completely constructed. This method can not be done repeatedly with ease and also can not test the logic of the method of a class. In this research is given solution in testing software with example of e-commerce system. E-commerce system is constructed using Java EE programming language and Spring framework. JUnit framewokr will handle source code testing using unit testing and integration testing. Based on the result of this study can be concluded, firstly, by using an automated testing a test case can be executed quickly and repeatedly. Secondly, in order to do unit testing, a class must implement dependency injection, this technique is used to insert a fake object for simulation. Thirdly, in contrast to unit testing that run independently, integration testing is used to test the integration between data access layer and database.

Keyword: automated testing, unit testing, integration testing, JUnit.

1. Pendahuluan

Proses pengembangan suatu perangkat lunak yang melibatkan banyak pihak diantaranya analis sistem dan pemrogram, harus menghasilkan suatu perangkat lunak yang berkualitas. Perangkat lunak yang berkualitas yaitu perangkat lunak yang bekerja sesuai dengan apa yang diinginkan (memenuhi *user requirements* dan *business process*), mudah untuk dimodifikasi guna pengembangan lebih lanjut serta minim adanya *bug*.

Sayangnya dalam pengembangan sebuah perangkat lunak, terutama para mahasiswa dan pengembang-pengembang non profesional, sering kali tidak menerapkan pengujian terutama *whitebox testing* pada perangkat lunak yang sedang dikembangkannya untuk menguji masing-masing logika dari method yang ada pada sebuah kelas.

Sehubungan dengan itu, kiranya perlu bagi para pengembang, untuk memahami dan menerapkan hal-hal mendasar (*minimum requirement*) dalam *whitebox testing* untuk melakukan pengujian logika dari *method* pada kelas yang menyusun suatu aplikasi. Di samping itu pengembang juga dapat menerapkan pengujian yang dapat mendeteksi adanya regresi ketika terjadi perubahan pada sebuah kode program.

Tujuan dari penelitian ini adalah mengetahui kebutuhan, pembuatan, dan cara kerja metode pengujian unit dan pengujian integrasi dalam melakukan pengujian aplikasi serta menerapkan metode-metode tersebut untuk melakukan pengujian terhadap sebuah sistem *e-commerce* yang dibangun menggunakan bahasa Java.

Penelitian ini dibatasi pada beberapa masalah berikut:

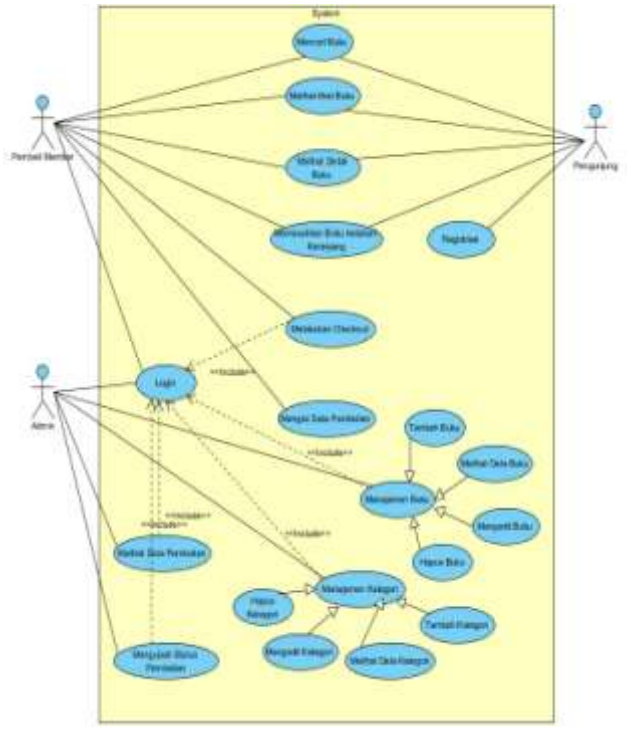
1. Menganalisis penerapan pengujian perangkat lunak menggunakan metode *Unit Testing* dan *Integration Testing*.
2. Pengujian sistem menggunakan metode *unit* dan *integration testing* menggunakan *framework* JUnit.
3. Perancangan sistem *e-commerce* menggunakan teknologi Java EE, *framework* Spring serta *framework* Hibernate.
4. Perancangan sistem *e-commerce* yang dibuat hanya membahas pada entitas pelanggan, entitas buku, entitas kategori, entitas order dan entitas order detail.
5. Perancangan aplikasi menggunakan arsitektur *three tier* dan pola MVC. *Three tier* yang dimaksud adalah aplikasi dibagi menjadi 3 lapisan yaitu lapisan *web*, lapisan *service* dan lapisan *data access.* Sedangkan pola MVC yang dimaksud adalah lapisan *web* dibagi lagi menjadi 3 bagian yaitu *Model*, *View* dan *Controller.*
6. Aplikasi yang dirancang digunakan sebagai contoh penerapan pengujian perangkat lunak menggunakan metode *Unit Testing* dan *Integration Testing.*
7. Pengujian unit hanya diterapkan untuk menguji kelas pada lapisan *service* serta *web*. Sedangkan pengujian integrasi hanya diterapkan untuk menguji kelas pada lapisan data *access.*
8. Perancangan aplikasi dan perancangan pengujian menggunakan konsep *Object Oriented Programming* (OOP).

2. Metode

Dalam penelitian ini dibuat sebuah program *e-commerce* yang didalamnya menerapkan arsitektur 3 *tier*, pola *Model View Controller* (MVC) dan juga *dependency injection*. Program yang dikembangkan adalah sebuah program yang bertujuan memberikan pemodelan mengenai penggunaan metode pengujian unit dan integrasi menggunakan *automated testing framework* JUnit. Fokus penelitian ini adalah mengenai pengujian perangkat lunak menggunakan pengujian unit dan integrasi sehingga program yang dibuat mencakup entitas yang dibutuhkan saja.

2.1 Perancangan Diagram *Use-Case*

Use-Case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use-case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Gambar 1 menunjukkan diagram *Use-Case* pada sistem yang akan dibuat.



Gambar 1. Diagram Use-Case sistem yang akan dibangun

2.2 Perancangan Testabilitas Kode

Tidak semua kode perangkat lunak yang dibuat bisa dengan mudah diuji menggunakan pengujian unit. Untuk itu diperlukan perancangan yang baik dalam penulisan kode agar kode perangkat lunak tersebut bisa dengan mudah diuji menggunakan pengujian unit. Berikut ini adalah perancangan penulisan kode yang penulis lakukan untuk mencapai testabilitas kode yang baik.

1. Menerapkan *interface*, dengan menggunakan *interface* maka proses *mocking* (memalsukan objek yang menjadi dependensi kelas yang sedang diuji) akan mudah dilakukan.
2. Menerapkan *dependency injection*, teknik ini memungkinkan untuk memasukkan *mock object* kedalam kelas yang sedang diuji.

3. Menghindari *hidden dependency*, hidden dependency akan membuat proses *mocking* tidak bisa dilakukan.

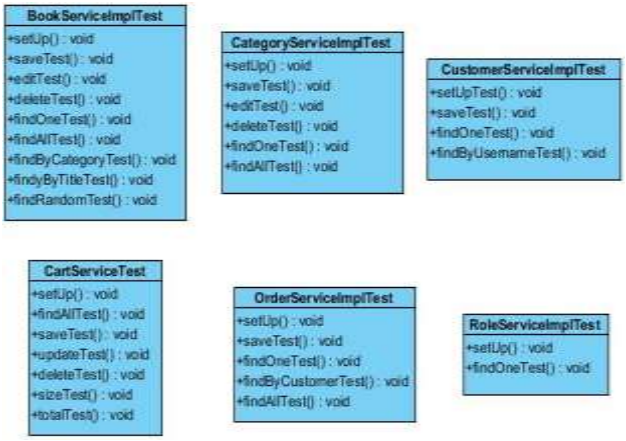
2.3 Perancangan Pengujian Unit

Gambar 2 menunjukkan diagram alir proses pengujian unit

Gambar 2. Diagram alir pengujian unit

2.4 Perancangan Pengujian Unit pada Layer *Service*

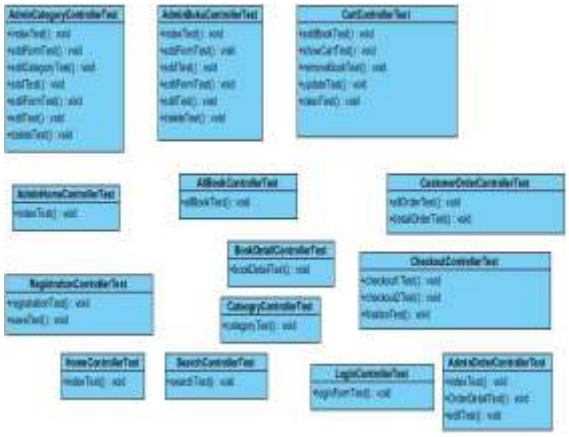
Layer *service* memiliki ketergantungan dengan layer *data access* oleh karena itu maka perlu dibuat sebuah *mock object* dari layer *data access*. Dengan *mock object* ini kita tinggal mengatur nilai kembaliannya sesuai dengan yang kita inginkan. Tanpa proses *mocking* maka pengujian akan tergantung pada objek dari kelas *data access* yang tergantung dengan basisdata sehingga hasil keluaran tidak bisa kita kontrol yang menyebabkan pengujian menjadi tidak independen dan tidak terisolasi. Gambar 3 merupakan gambar diagram kelas dari pengujian unit pada layer *service*.



Gambar 3. Diagram kelas pengujian unit pada layer service

2.5 Perancangan Pengujian Unit pada Layer *Web*

Kelas pada layer ini tergantung pada objek dari kelas pada layer s*ervice* sehingga perlu dibuat mock object dari layer *service*. Pada layer ini komponen c*ontroller*-lah yang akan diuji. Gambar 4 berikut ini menunjukkan diagram kelas pengujian unit dari layer *web*.



Gambar 4 Diagram kelas pengujian unit pada layer web

2.6 Perancangan Pengujian Integrasi

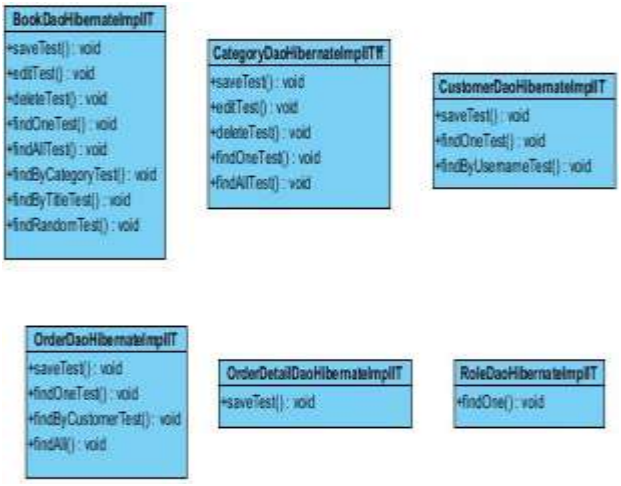
Secara ringkas pengujian integrasi dilakukan dengan cara menulis sebuah kode yang memiliki tahap-tahap sebagai berikut

1. Membuat struktur tabel secara automatis
2. Mengisi tabel dengan data contoh yang telah disiapkan secara automatis
3. Mengeksekusi *method* yang sedang diuji untuk dibandingkan nilai keluarannya
4. Membandingakan nilai keluaran pada langkah 3 (*actual value*) dengan nilai yang diharapkan (*expected value*)
5. Mengulangi langkah 1-4 sebanyak *method* yang akan diuji pada kelas yang sedang diuji.

Gambar 5 merupakan diagram alir dari pengujian integrasi yang dilakukan untuk menguji layer *data access*.

Gambar 5. Diagram alir pengujian unit

Gambar 6 menunjukkan diagram kelas pengujian integrasi pada layer *data access.*



Gambar 6. Diagram kelas pengujian integrasi pada layer data access

3. Hasil dan Analisa

3.1 Implementsi Pengujian Unit

3.1.1 Pengujian Unit Layer *Service* (Kelas `BookServiceImpl` *Method* findOne)

Pengujian pada *method* **findOne** ini memastikan bahwa nilai kembalian dari `bookDao` diterima oleh `bookServiceImpl` dan kemudian dijadikan sebagai nilai kembalian juga. Kode pengujian ditunjukkan dibawah ini.

```
   @Test
public void findOneTest(){
    Book bookToFind=new ……
    ……

   Mockito.when(bookDao.findOne(1L)).th
   enReturn(bookToFind);

   Book
   actual=bookServiceImpl.findOne(1L);

   Mockito.verify(bookDao,Mockito.times
   (1)).findOne(1L);

   Mockito.verifyNoMoreInteractions(boo
   kDao);

   Assert.assertNotNull(actual);
   Assert.assertEquals(bookToFind,
   actual);

   Assert.assertEquals(bookToFind.getId
   (), actual.getId());
}
```

Objek `bookToFind` adalah objek yang digunakan sebagai data contoh pada pengujian. Objek ini digunakan sebagai nilai kembalian dari method **findOne** yang dimiliki `bookDao`. Objek buku ini memiliki id dengan nilai 1. Id inilah yang akan dijadikan sebagai kunci untuk pencarian data. Selanjutnya mengkonfigurasi *mock object* untuk mengembalikan `bookToFind` ketika method **findOne** pada `BookDao` dieksekusi dengan 1 sebagai parameternya.

Langkah pengujian berikutnya adalah mengeksekusi method yang diuji yaitu **findOne** pada `bookServiceImpl`. Nilai kembalian hasil eksekusi disimpan pada variabel dengan nama `actual`. Selanjutnya adalah melakukan verifikasi bahwa `bookDao` mengeksekusi method **delete** sebanyak satu kali dan juga memverifikasi bahwa `bookDao` tidak melakukan operasi lagi. Terakhir adalah memeriksa bahwa variabel `actual` memiliki nilai dan juga variabel ini adalah sama dengan `bookToFind` yang berarti data kembalian `bookDao` sampai pada `bookServiceImpl` dan menjadi nilai kembalian lagi.

3.1.2 Pengujian Unit Layer *Service* (Kelas CartService *Method* save)

Method **save** digunakan untuk menyimpan data buku pada memori. Dibawah ini merupakan implementasi kode pengujian *method* **save**.

```
@Test
 public void saveTest(){
    Book bookToSave=new ……
    ……
    Book
    actual=cartService.save(bookToSave)
    ;

    Assert.assertNotNull(actual);
  Assert.assertEquals(bookToSave,
    actual);
    Assert.assertEquals("Java Testing",
    actual.getTitle());
    Assert.assertEquals(1,
    cartService.size(),0);
}
```

Seperti biasa hal pertama yang dilakukan pada pengujian adalah menyiapkan data, dalam hal ini data buku yang akan disimpan. Data buku ini memiliki judul "Java Testing" dan beberapa properti lainnya. Kemudian *method* **save** yang diuji dieksekusi dengan data buku yang telah disiapkan tadi menjadi parameternya dan nilai kembaliannya disimpan pada variabel yang bernama `actual`. 4 baris terakhir adalah memastikan bahwa variabel `actual` tidak bernilai kosong, memastikan variabel `actual` sama dengan data buku yang disimpan yaitu `bookToSave`, lalu memeriksa bahwa buku yang telah disimpan memang berjudul "Java Testing" dan terakhir adalah memeriksa bahwa jumlah buku yang berada pada keranjang belanja adalah 1 setelah adanya proses penyimpanan.

3.1.3 Pengujian Unit Layer *Web* (Kelas Book Detail Controller *Method* book Detail)

Method **bookDetail** digunakan untuk melihat data buku secara detail berdasarkan id buku yang terdapat pada url sebagai *path variable.* Ketika data buku dengan id tertentu tidak ditemukan maka *behavior* yang terjadi adalah *method* ini akan menampilkan halaman 404.

```
@Test
public void bookDetailNotFoundTest()
throws Exception{

Mockito.when(bookService.findOne(1L)).
thenReturn(null);

mockMvc.perform(get("/public/book/deta
il/{bookId}",1L))
.andExpect(status().isNotFound())
.andExpect(view().name("404"))
.andExpect(forwardedUrl("/WEB-
INF/jsp/404.jsp"));

Mockito.verify(bookService,Mockito.tim
es(1)).findOne(1L);
Mockito.verifyNoMoreInteractions(bookS
ervice);
}
```

3.2 Implementasi Pengujian Integrasi

3.2.1 Pengujian Integrasi dengan Basisdata layer *data access* (Kelas BookDao *Method* findAll)

Method **findAll** digunakan untuk mengambil semua data buku yang tersimpan pada basisdata. Pada pengujian ini digunakan 3 buah data buku dan berikut adalah kode pengujiannya

```
@Test
@DatabaseSetup("classpath:sampleData.x
ml")
public void findAllTest(){
  List<Book>
  listBook=bookDao.findAll();
  assertNotNull(listBook);
  assertEquals(3,     listBook.size());
  assertEquals("Spring            MVC",
  listBook.get(1).getTitle());
  assertEquals("Java",
  listBook.get(1).getCategory().getName
  ());
}
```

Setelah data contoh dimasukkan berikutnya yang dilakukan adalah mengeksekusi *method* **findAll** ini yang sedang diuji dan nilai kembaliannya disimpan pada sebuah *list* yang bernama `listBook`. Berikutnya adalah memastikan bahwa variabel ini tidak bernilai kosong, memiliki jumlah 3 data buku, memiliki salah satu data buku yang berjudul "Spring MVC" yang terdapat pada kategori "Java".

3.3 Hasil Pengujian

Tabel 1 menunjukkan hasil pengujian unit yang dilakukan terhadap kelas CartController pada layer *web*. Dari tabel tersebut menunjukkan bahwa semua *method* yang ada telah sesuai dengan yang diinginkan.

Tabel 1 Hasil Pengujian Unit Kelas CartController

No.	Method yang diuji	Hasil
1.	cartController.showCart()	sesuai
2.	cartController.addBook()	sesuai
3.	cartController.removeBook() [kondisi 1]	sesuai
4.	cartController.removeBook() [kondisi 2]	sesuai
5.	cartController.update()	sesuai
6.	carController.clear()	sesuai

Tabel 2 menunjukkan hasil pengujian unit yang dilakukan terhadap kelas CartService pada layer *service*. Berdasarkan tabel tersebut maka semua *method* yang ada telah sesuai dengan harapan.

Tabel 2 Hasil Pengujian Unit Kelas CartService

No.	Deskripsi	Hasil
1.	cartService.findAll()	sesuai
2.	cartService.save() [kondisi 1]	sesuai
3.	cartService.save() [kondisi 2]	sesuai
4.	cartService.update()[kondisi1]	sesuai
5.	cartService.update()[kondisi2]	sesuai
6.	cartService.delete()	sesuai
7.	cartService.clear()	sesuai
8.	cartService.size()	sesuai
9.	cartService.total()	sesuai

Tabel 3 menunjukkan hasil pengujian integrasi yang dilakukan terhadap kelas BookDaoHiberntaeImpl pada layer *data access*. Berdasarkan tabel tersebut maka semua *method* yang ada memiliki *behavior* yang sesuai dengan yang diharapan.

Tabel 3 Hasil Pengujian Integrasi Kelas Book Dao Hibernate Impl

No.	Deskripsi	Hasil
1.	bookDaoHibernateImpl.save()	sesuai
2.	bookDaoHibernateImpl.delete()	sesuai
3.	bookDaoHibernateImpl.findAll()	sesuai
4.	bookDaoHibernateImpl.edit()	sesuai
5.	bookDaoHibernateImpl.findOne()	sesuai
6.	bookDaoHibernateImpl.findByCategory()	sesuai
7.	bookDaoHibernateImpl.findByTitle()	sesuai

4. Kesimpulan

Berdasarkan hasil perancangan, implementasi dan analisis terhadap pengujian unit dan integrasi dapat disimpulkan bahwa pada akhir perancangan dihasilkan sebuah model program untuk menerapkan pengujian automatis unit dan integrasi yang dapat dieksekusi secara cepat dan dapat dilakukan secara berulang-ulang dengan mudah.

Pengujian unit bisa dilakukan ketika sebuah sistem belum terbangun secara utuh, hal ini dimungkinkan karena pengujian unit hanya digunakan untuk menguji *behavior* pada method suatu kelas.

Pengujian integrasi basisdata dilakukan dengan diawali proses menciptakan sebuah tabel baru, diikuti dengan memasukkan data contoh, kemudian dilakukan proses pengujian. Terakhir adalah menghapus seluruh data yang ada dan menghapus tabel sehingga saat pengujian berikutnya data selalu dalam keadaan bersih.

Baik pengujian unit maupun pengujian integrasi dilakukan dengan cara membandingkan nilai kembalian yang dihasilkan oleh suatu *method* yang sedang diuji dengan suatu nilai yang diperkirakan sendiri oleh pengembang.

Adapun saran yang dapat diberikan untuk menjadi masukan pada penelitian lebih lanjut yaitu pertama bahwa aplikasi yang digunakan sebagai contoh pengujian masih tergolong sederhana sehingga perlu dilakukan penelitian lebih lanjut terhadap pengujian pada aplikasi yang kompleks. Kedua, diharapkan pengujian automatis menggunakan metode pengujian unit dan integrasi untuk diterapkan pada PENELITIAN mahasiswa disamping pengujian manual menggunakan pengujian fungsional.

Referensi

[1]. Bauer, C. and G. King, *Java Persistence With Hibernate*, Manning, Greenwich, 2006

[2]. Chonoles , M. J. and James A. S., *UML 2 for Dummies*, Wiley Publishing, Inc, New York, 2003.

[3]. Deinum, Marten dkk, *Pro Spring MVC:with Web Flow*, Apress, New York City, 2012

[4]. Gupta, Arun, *Java EE 6 Pocket Guide, O'Reilly*, California, 2012

[5]. Hamilton, K. and R. Miles, *Learning UML 2.0*, O'Reilly, California, 2006.

[6]. Khannedy, Eko Kurniawan, *Modul Pelatihan Java Dasar*, Bandung, 2006.

[7]. Pillay, Anban, *Object Oriented Programming using Java*, University of KwaZulu-Natal, Durban, 2007.

[8]. Rosa dan M. Shalahuddin, *Modul Pembelajaran Rekayasa Perangkat Lunak (Terstruktur & Berorientasi Objek)*, Modula, Bandung, 2011

[9]. Tahciev, Petar dkk, *JUnit in Action*, Manning, Greenwich, 2010

[10]. Wahana, *Pengembangan Aplikasi Database Berbasis JavaDB*, Penerbit Andi, Yogyakarta, 2010.

[11]. Yulianto , *Ardhian Agung, Analisis dan Desain Sistem Informasi*, Politeknik Telkom, Bandung, 2009.