PENGUJIAN DENGAN TEST CASE DAN UNIT TEST PADA PROYEK PENGEMBANGAN MODUL MANAJEMEN PENGGUNA

by John Doe

Submission date: 24-Nov-2020 07:22PM (UTC+0700)
Submission ID: 1454393156
File name: UNIT_TEST_PADA_PROYEK_PENGEMBANGAN_MODUL_MANAJEMEN_PENGGUNA.pdf (494.5K)
Word count: 2571
Character count: 16233

PENGUJIAN DENGAN *TEST CASE DAN UNIT TEST* PADA PROYEK PENGEMBANGAN MODUL MANAJEMEN PENGGUNA

Modul Manajemen Pengguna adalah suatu modul yang berfungsi untuk mengelola pengguna suatu sistem perangkat lunak. Dalam pengembangan modul ini, penulis dan tim diberi spesifikasi modul yang diinginkan oleh perusahaan. Untuk memastikan bahwa modul yang dihasilkan sudah sesuai dengan spesifikasi perusahaan tersebut, penulis dan tim perlu melakukan pengujian. Pengujian perangkat lunak berfungsi untuk menemukan kesalahan dan cacat saat menjalankan program perangkat lunak tersebut. Pengujian dapat menjadi alat ukur kualitas suatu perangkat lunak sebelum diluncurkan. Makalah ini menyampaikan aktivitas pengujian dalam pengembangan modul Manajemen Pengguna dengan test case dan unit testing. Test case digunakan agar proyek tersebut tersebut sesuai dengan spesifikasi dari PT. Dua Empat Tujuh. Sedangkan, unit test digunakan untuk menemukan error yang tidak terlihat saat program dijalankan dan menemukan kode program yang kurang efektif pada modul Manajemen Pengguna tersebut.

Keywords—Pengujian, Test Case, Unit Testing

I. PENDAHULUAN

Modul Manajemen Pengguna adalah suatu modul yang berfungsi untuk mengelola pengguna suatu sistem perangkat lunak. Modul ini bersifat generik sehingga dapat dimasukkan ke dalam proyek lain. Saat ini, PT. Dua Empat Tujuh sedang mengembangkan modul Manajemen Pengguna yang akan dimasukkan ke dalam proyek ERP yang berjalan. Proyek pengembangan modul Manajemen Pengguna merupakan proyek baru yang dikerjakan oleh penulis dan tim mulai dari nol. Dalam pengembangan proyek ini, penulis dan tim diberi spesifikasi modul yang diinginkan oleh perusahaan. Untuk memastikan bahwa modul yang dihasilkan sudah sesuai dengan spesifikasi perusahaan tersebut, penulis dan tim perlu melakukan pengujian (*testing*) modul Manajemen Pengguna yang sedang dikembangkan.

Pengujian merupakan salah satu aktivitas penting dalam pengembangan perangkat lunak. Pengujian bertujuan untuk meminimalkan peluang kegagalan dalam eksekusi perangkat lunak. Pengujian perangkat lunak juga bertujuan untuk menemukan kesalahan (*error*) dan cacat (*bug*) saat eksekusi perangkat lunak tersebut. Kesalahan pada perangkat lunak dapat menyebabkan perangkat lunak tersebut tidak dapat dijalan. Sedangkan cacat pada perangkat lunak menyebabkan perangkat lunak tersebut tidak berjalan sesuai dengan yang diharapkan.

Pengujian dapat menjadi alat ukur kualitas perangkat lunak yang sangat diperlukan sebelum sebuah perangkat lunak tersebut dirilis atau diluncurkan. Hal tersebut perlu dilakukan untuk meyakinkan pelanggan atau pengguna bahwa perangkat lunak tersebut sudah layak untuk digunakan dan memenuhi kinerja sesuai kebutuhan dari pelanggan atau pengguna [1]. Teknik pengujian yang digunakan dalam pengembangan modul Manajemen Pengguna adalah *test case* dan *unit testing*.

Test case merupakan teknik pengujian perangkat lunak yang menggunakan serangkaian skenario eksekusi untuk mengetahui apakah modul yang sedang dikembangkan sudah memenuhi spesifikasi dari PT. Dua Empat Tujuh. Pada hakikatnya pengujian tidak dapat membuktikan kebenaran semua kemungkinan yang terjadi dalam eksekusi modul tersebut. Namun hal ini dapat didekati dengan melakukan perencanaan dan desain *test case* yang baik, sehingga *test case* dapat memberikan jaminan efektifitas perangkat lunak sampai pada tingkat tertentu sesuai dengan yang diharapkan [2].

Unit testing adalah teknik untuk menguji apakah kode program perangkat lunak sudah efektif dan bebas dari kesalahan. Pada beberapa kasus pengembangan perangkat lunak, sering terjadi bahwa kode program yang dikembangkan ternyata kurang efektif atau bahkan tidak pernah digunakan. Pengujian jenis ini dapat membantu dalam menemukan kode program yang kurang efektif tersebut serta dapat mengukur seberapa efektif kode program pada pengembangan perangkat lunak tersebut. Selain itu, pengujian ini sering gunakan oleh para pengembang untuk menemukan kesalahan dan cacat yang tidak terlihat saat program dieksekusi.

Tujuan makalah ini adalah untuk memberi gambaran implementasi pengujian menggunakan *test case* dan *unit testing* pada proyek pengembangan modul Manajemen Pengguna.

II. DASAR TEORI

A. *Pengujian*

Pengujian perangkat lunak adalah proses eksekusi sistem perangkat lunak dengan tujuan untuk menemukan kesalahan atau cacat pada perangkat lunak tersebut [3]. Pada hakikatnya, perangkat lunak yang berkualitas adalah perangkat lunak yang bebas dari kesalahan dan cacat secara objektif. Untuk mendapatkan hasil yang objektif, perangkat lunak harus melalui suatu proses pengujian yang terstruktur, terencana, dan terdokumentasi dengan baik. Pengujian yang baik memiliki probabilitas tinggi dalam menemukan kesalahan [4]. Selain itu, perangkat lunak juga harus tepat waktu dan dana, sesuai dengan kebutuhan dan keinginan pelanggan atau pengguna, serta mudah dalam pemeliharaannya.

Berdasarkan hasil pengujian, pengembang dapat mencari dan menelusuri kesalahan sampai ke kebutuhan pelanggan dan pengguna, untuk kemudian memperbaiki sebanyak mungkin kesalahan dalam kode program suatu perangkat lunak sebelum menyerahkan perangkat lunak tersebut kepada pelanggan atau pengguna. Hal tersebut dilakukan untuk menghindari cacat yang paling fatal, yaitu cacat yang dapat dilihat oleh pelanggan atau pengguna perangkat lunak tersebut. Cacat pada perangkat lunak dapat mengakibatkan perangkat lunak tersebut gagal dalam memenuhi kebutuhan atau keinginan dari pelanggan atau pengguna perangkat lunak tersebut.

Pada umumnya pengujian perangkat lunak dilakukan sesudah tahap pemrograman pada pengembangan perangkat lunak, akan tetapi perencanaan pengujian sebaiknya dilakukan pada tahap analisis pengembangan perangkat lunak.

B. *Test Case*

Test case adalah pengujian yang dilakukan berdasarkan beberapa masukan seperti kondisi dan hasil yang telah ditentukan sebelumnya [2]. Hasil dari *test case* yang telah dilakukan akan dibandingkan dengan hasil yang telah ditentukan sebelumnya. Jika terdapat perbedaan di antara keduanya maka akan dilakukan perbaikan pada kode program. *Test case* menjadi titik awal untuk melakukan pengujian setelah memasukan nilai-nilai inputan ke sistem. Setelah itu, akan didapat hasil yang definitif dan meninggalkan sistem di beberapa titik akhir atau juga dikenal sebagai *postcondition* eksekusi [5].

Test case yang baik adalah *test case* yang mempunyai kemungkinan tinggi dalam menemukan kesalahan dan cacat yang sebelumnya belum ditemukan, bukan yang dapat memperlihatkan bahwa program telah bekerja dengan benar [2].

C. *Unit Testing*

Unit testing merupakan tahap dasar dalam pengujian. *Unit testing* berfokus pada penguijian *building block* yang lebih kecil daripada program atau sistem yang diuji. Pengujian ini mengeksekusi setiap fitur atau fungsionalitas untuk memastikan masing-masing fitur atau fungsionalitas tersebut berfungsi sesuai dengan yang diharapkan [6]. Pada definisi lainnya, *unit testing* merupakan pengujian yang mencakup pengujian sepotong atau sebagian kode [7]. Dengan demikian, pengujian ini dapat digunakan untuk menguji kelas atau modul tertentu.

Pada implementasinya, *unit testing* berupa sebuah kode program yang dibuat oleh tim pengembang untuk menguji sebuah fitur atau fungsionalitas pada sebuah sistem. Hal tersebut dilakukan untuk membuktikan apakah kode program yang telah dikembangkan sudah sesuai yang diharapkan atau tidak. *Unit testing* ini sangat efektif bagi tim pengembang, karena akan mempermudah tim pengembang dalam memperbaiki desain kode program dan mengurangi waktu saat melakukan *debugging*.

Unit testing dilakukan setelah tim pengembang selesai menulis kode program pada suatu modul dalam program perangkat lunak, atau bisa juga setelah selesai menambahkan suatu fitur atau fungsionalitas dalam program perangkat lunak tersebut. *Unit testing* termasuk pengujian otomatis yang membutuhkan suatu *tool*, yang mana di setiap bahasa pemrograman yang dipakai akan menggunakan *tools* yang berbeda.

III. METODE

A. *Test Case*

Pengujian menggunakan *test case* termasuk pengujian manual, sehingga dalam implementasinya tim pengembang harus melakukan pengujian ini satu per satu pada setiap fitur atau fungsionalitas yang akan diuji. Dalam pembuatan *test case*, ada beberapa atribut yang dibutuhkan, seperti pada **Tabel 1.**

TABLE I. ATRIBUT *TEST CASE*

Atribut	Keterangan Atribut
Fitur	Nama fitur atau fungsionalitas yang akan diuji
Test case id	Identitas dari fitur atau fungsionalitas yang akan diuji
Test case description	Deskripsi fitur atau fungsionalitas yang akan diuji
Pre condition	Kondisi sistem sebelum melakukan pengujian
Test steps	Langkah-langkah saat melakukan pengujian
Expected result	Hasil yang diinginkan dari fitur atau fungsionalitas yang diuji
Actual result	Hasil yang sebenarnya terjadi pada fitur atau fungsionalitas yang telah diuji
Status	Status udah melakukan test (*pass or fail*)

Setelah menyusun atribut-atribut yang dibutuhkan, langkah selanjutnya adalah membuat model *test case* sesuai dengan atribut yang telah ditentukan sebelumnya yang dapat dilihat pada **Tabel 2**.

TABLE II. MODEL *TEST CASE*

Fitur	*Test Case ID*	*Test Case Description*	*Pre Condition*	*Test Steps*	*Expected Result*	*Actual Result*	Status

B. *Unit Testing*

Berbeda dengan *test case*, *unit testing* termasuk pengujian otomatis sehingga tim pengembang perlu menyediakan kode program untuk menguji fitur atau fungsionalitas yang diinginkan, kemudian menjalankan program perangkat lunak tersebut. Pada pengujian *unit testing* ini, penulis menggunakan *testing framework* Jest sebagai *tool* pengujiannya. Adapun alasan menggunakan *tools* tersebut karena dalam pengembangan modul Manajemen Pengguna, tim pengembang menggunakan dua *framework* yang salah satunya adalah *framework* React Js.

Selain itu, *testing framework* Jest ini memiliki *running test suites* yang cepat dan, secara *default, test suites* akan dijalankan secara paralel, sehingga sangat cocok untuk menguji komponen-komponen *framework* React [8]. Tidak hanya itu, penulis juga menggunakan *testing library* Enzyme untuk melakukan pengujian pada komponen-komponen React. *Testing library* Enzyme memiliki *method* yang disediakan sangat baik untuk memanipulasi *output* dari virtual DOM [9].

Langkah pertama dalam *unit testing* adalah melakukan konfigurasi pada program yang akan diuji. **Gambar 1** merupakan perintah konfigurasi *framework* Jest pada program.

```
yarn add --dev jest
#or
npm install --save-dev jest
```

Gambar 1. Konfigurasi *framework* Jest

Setelah mengkonfigurasi *framework* Jest pada program, langkah berikutnya adalah mengkonfigurasi *library* Enzyme. **Gambar 2** merupakan perintah konfigurasinya.

```
yarn add --dev enzyme enzyme-adapter-react-16 react-test-renderer
#or
npm install --save-dev enzyme enzyme-adapter-react-16 react-test-
renderer
```

Gambar 2. Konfigurasi *libsary* Enzyme

Setelah melakukan semua konfigurasi yang diperlukan, langkah selanjutnya adalah membuat kode program untuk fitur atau fungsionalitas yang akan diuji. Sebagai contoh implementasinya, penulis akan menguji fitur *form create user* pada modul Manajemen Pengguna. **Gambar 3** merupakan kode program untuk melakukan *unit testing* pada *form create user*.

```
import React from 'react';
import CreateUser, {Role} from '../models/create.user';
import Enzyme, {shallow, mount} from "enzyme";
import Adapter from "enzyme-adapter-react-16";
Enzyme.configure({adapter:new Adapter()});

describe("create.user component", () => {
  test("renders", () => {
    const wrapper = shallow(<CreateUser />);
    //expect(wrapper.exists()).toBe(true);
    expect(wrapper).toMatchSnapshot();
  });
  it('Should capture name correctly onChange', () =>{
    const component = mount(<CreateUser />);
    const input = component.find('input').at(0);
    input.instance().value = 'Nauli';
    input.simulate('change');
    expect(component.state().fields["campName"]).toEqual('Nauli');
    expect(component).toMatchSnapshot();
  })
  it('Should capture email correctly onChange', () =>{
    const component = mount(<CreateUser />);
    const input = component.find('input').at(1);
    input.instance().value = 'nauli@gmail.com';
    input.simulate('change');
    expect(component.find('input').at(1).props().value).toEqual('nauli@gmail.com');
    expect(component).toMatchSnapshot();
   })
   it('Should capture password correctly onChange', () =>{
    const component = mount(<CreateUser />);
    const input = component.find('input').at(2);
    input.instance().value = 'Nauli17_';
    input.simulate('change');
    expect(component.find('input').at(2).props().value).toEqual('Nauli17_');
    expect(component).toMatchSnapshot();
  })
  it('Should capture phone correctly onChange', () =>{
    const component = mount(<CreateUser />);
    const input = component.find('input').at(3);
    input.instance().value = '08126312810';
    input.simulate('change');
    expect(component.find('input').at(3).props().value).toEqual('08126312810');
    expect(component).toMatchSnapshot();
  })
  it('Should capture select role correctly onChange', () =>{
    const component = mount(<CreateUser />);
    const input = component.find('select').at(0);
    const optionRole = component.find('option').at(1);
    optionRole.instance().selected  = true;
    input.simulate('change');
    expect(component.state().selectRole).toEqual('1');
    expect(component).toMatchSnapshot();
  });
  it('calls handleSubmit prop function when form is submitted', () => {
    const wrapper = shallow(<CreateUser />);
    const fn = jest.fn();
    wrapper.instance().handleSubmit = fn;
    wrapper.instance().handleSubmit();
        wrapper.find('form').simulate('submit', {
      preventDefault: () => {}
    });
    expect(fn).toHaveBeenCalled();
    expect(wrapper).toMatchSnapshot();
  });
  it('calls handleSubmit prop function when form is submitted', () => {
    const wrapper = shallow(<CreateUser />);
    const fn = jest.fn();
    wrapper.instance().handleSubmit = fn;
    wrapper.instance().handleSubmit();
        wrapper.find('form').simulate('submit', {
      preventDefault: () => {}

    });
    //wrapper.setState(state)
    expect(fn).toHaveBeenCalled();
    expect(wrapper).toMatchSnapshot();
  });

})
```

Gambar 3. Kode program *unit testing form create user*

Setelah itu, untuk mempermudah *running test* dalam *unit testing* ini, penulis melakukan konfigurasi perintah pada package.json program modul Manajemen Pengguna, seperti pada **Gambar 4**. Ketika akan menjalankan *running test* kode program yang telah dibuat sebelumnya, cukup dilakukan dengan perintah "*npm run test*".

```
"scripts": {
  "test": "react-scripts test",
```

Gambar 4. Konfigurasi perintah *test* pada package.json

IV. HASIL DAN PEMBAHASAN

A. *Test Case*

Setelah menyusun atribut-atribut yang dibutuhkan dalam melakukan pengujian menggunakan *test case*, **Tabel 3** adalah hasil implementasinya untuk fitur atau fungsionalitas *create user*.

TABLE III. MODEL *TEST CASE*

Fitur	*Test Case ID*	*Test Case Description*	*Pre Condition*	*Test Steps*	*Expected Result*	*Actual Result*	Status
Create User	TC01	Menambahkan pengguna baru dengan data yang *valid*	Sudah berada di halaman *dashboard*	1. Klik tombol "*Create User*" 2. Masukkan nama : Magdalena Joana 3. Masukkan nomor telepon : +6289631977045 4. Masukkan *email* : magda@gmail.com 5. Masukkan *password* : Agh.Bil* 6. Pilih *role* : *Role* 1 7. Klik tombol "Simpan"	Kembali ke halaman daftar pengguna dan data pengguna baru akan langsung muncul pada daftar pengguna.	Sesuai	*Pass*
	TC02	Salah satu kolom ada yang kosong		1. Klik tombol "*Create User*" 2. Masukkan nama : 3. Masukkan nomor telepon: +6289631977045	Tetap berada didalam halaman *form* "*Create User*" dan pada kolom	Sesuai	*Pass*

				4. Masukkan *email* : magda@gmail.com 5. Masukkan *password* : Agh.Bil* 6. Pilih *role* : *Role* 1 7. Klik tombol "Simpan"	nama terdapat *alert*, karna kolom nama belum diisi.		
	TC03	Kolom nomor telepon harus *valid* nomor telepon Indonesia		1. Klik tombol "*Create User*" 2. Masukkan nama : Magdalena Joana 3. Masukkan nomor telepon: +6889631977045 4. Masukkan *email* : magda@gmail.com 5. Masukkan *password* : Agh.Bil* 6. Pilih *role* : *Role* 1 7. Klik tombol "Simpan"	Tetap berada didalam halaman *form* "*Create User*" dan pada kolom nomor telepon terdapat *alert*, karna nomor telepon tidak nomor telepon Indonesia	Sesuai	*Pass*
	TC04	Kolom nama hanya boleh berisi abjad dan spasi		1. Klik tombol "*Create User*" 2. Masukkan nama : Magdalena_Joana 3. Masukkan nomor telepon: +6289631977045 4. Masukkan *email* : magda@gmail.com 5. Masukkan *password* : Agh.Bil* 6. Pilih *role* : *Role* 1 7. Klik tombol "Simpan"	Tetap berada didalam halaman *form* " *Create User*" dan pada kolom nama terdapat *alert*, karna kolom nama berisi selain abjad dan spasi	Sesuai	*Pass*
	TC05	Kolom *password* kurang dari 8 karakter dan harus berisi minimal karakter, huruf besar, huruf kecil, angka, simbol		1. Klik tombol "*Create User*" 2. Masukkan nama : Magdalena Joana 3. Masukkan nomor telepon: +6289631977045 4. Masukkan *email* : magda@gmail.com 5. Masukkan *password* : nauuuu 6. Pilih *role* : *Role* 1 7. Klik tombol "Simpan"	Tetap berada didalam halaman *form* " *Create User*" dan pada kolom *password* terdapat *alert*, karna *password* kurang dari 8 karakter	Sesuai	*Pass*

Pada **Tabel 3**, dapat dilihat bahwa semua fitur atau fungsionalitas *create user* sudah sesuai dengan spesifikasi yang diberikan oleh perusahaan kepada tim pengembang. Selain itu, pada gambar tersebut terdapat perbandingan hasil yang diharapkan dengan hasil yang sebenarnya. Hal tersebut yang nantinya akan dapat digunakan oleh tim pengembang untuk memperbaiki fitur atau fungsionalitas yang sedang diuji.

Tabel 4 merupakan statistik data hasil pengujian dengan *test case* untuk semua fitur atau fungsionalitas dari modul Manajemen Pengguna.

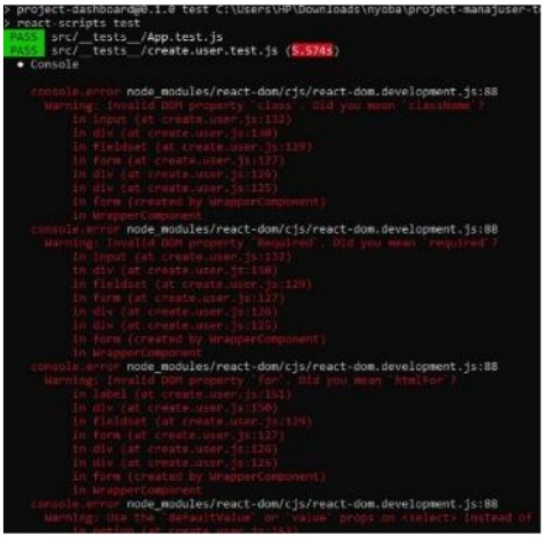
TABLE IV. STATISTIK DATA PENGUJIAN

Fitur	Jumlah *Test Case*	Persentase Lulus Uji
Create User	5	100 %
Edit User	5	100 %
Delete User	2	100 %
Search	4	75%
Paging	1	100 %
Entry limit	1	100 %

B. Unit Testing

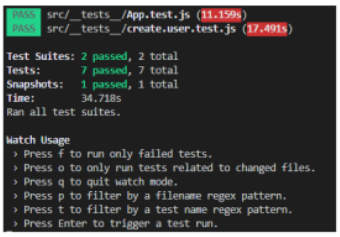
Setelah selesai memasukkan kode program untuk *unit testing* program yang akan diuji, langkah selanjutnya melakukan *running test*. **Gambar 5** merupakan hasil dari *running unit testing* yang sudah dilakukan. Pada gambar tersebut, dapat dilihat terdapat kesalahan di beberapa baris pada *class form create user*, seperti kesalahan dalam penulisan komponen program *'class'* yang seharus *'className'*, *'Required'* yang seharusnya *'required'*, *'for'* yang seharus *'htmlfor'*, *'selected'* yang seharusnya *'defaultValue'* atau *'value'*, dan kesalahan pada identifikasi variabel yang dibutuhkan dalam program.

Padahal, jika program tersebut dijalankan sudah tidak terdapat kesalahan pada tampilannya dan semua fungsionalitasnya berfungsi sesuai dengan yang diharapkan. Hal ini sering sekali terjadi pada pengembangan perangkat lunak, yang membuat tim pengembang mengira jika program yang dikembangkan sudah berhasil dan tidak ada kesalahan. Namun, ternyata realitanya tidak demikian. Oleh karena itulah pengujian *unit testing* pada pengembangan perangkat lunak penting untuk dilakukan.



Gambar 5. Hasil *unit tesing form create user*

Setelah mengetahui dibagian mana saja yang salah pada *class form create user* tersebut, tim pengembang memperbaiki bagian-bagian yang salah tersebut sampai semua kesalahan terselesaikan. **Gambar 6** merupakan hasil dari *running testing unit* yang dilakukan setelah memperbaiki kode program pada *class form create user*.



Gambar 6. Hasil *unit tesing form create user* terbaru

V. KESIMPULAN

Dari pembahasan sebelumnya dapat disimpulkan bahwa pengujian dalam pengembangan perangkat lunak itu sangat penting dilakukan. Salah satu manfaat yang didapatkan dari pengujian perangkat lunak ini adalah pengembangan menghasilkan perangkat lunak yang berkualitas. Salah satu komponen perangkat lunak dikatakan berkualitas adalah perangkat lunak yang bebas dari kesalahan dan cacat.

Pengujian dengan *test case* dapat mempermudah tim pengembang yang baru mengembangkan proyek perangkat lunak, agar proyek tersebut tersebut sudah sesuai dengan keinginan dan kebutuhan klien. Pada implementasinya *test case* dilakukan dengan membandingkan antara hasil yang sebenarnya terjadi dan hasil yang diharapkan. Hal tersebut akan mempermudah tim pengembang menemukan dan memperbaiki kesalahan yang terdapat pada perangkat lunak. Jika ditarik kesimpulannya, pengujian dengan menggunakan *test case* ini dapat menemukan kesalahan dan cacat yang dapat dilihat.

Pengujian dengan menggunakan *unit testing* dapat menemukan kesalahan dan cacat yang tidak dapat dilihat, seperti implementasi *unit testing* sebelumnya yang terdapat pada Bab Hasil dan Pembahasan. Hal tersebut dapat terjadi karena kode program yang telah dikembangkan kurang efektif. Selain itu, *unit testing* ini dapat membantu tim pengembang untuk menemukan dan memperbaiki kesalahan dan cacat lebih cepat, sehingga dapat mempersingkat waktu pengembangan perangkat lunak tersebut.

VI. REFERENCES

[1] J. F. Andry and Reinaldo, "Testing dan Implementasi Aplikasi Parkir di PT ABC Menggunakan Metode Black Box," *Prosiding Seminar Nasional Multidisiplin Ilmu*, 2017.

[2] S. Romeo, Testing dan Implementasi Sistem, Surabaya: STIKOM, 2003.

[3] G. Myers, The Art Of Sortware Testing, New York: Wiley, 1979.

[4] R. Pressman, Software Engineering : A Practitioner's Approach, New York: McGraw-Hill, 2010.

[5] R. Yunisa, "Perbandingan 2 Teknik White Box Testing: Statement Coverage Testing Dan Branch Coverage Testing (Studi Kasus : Sistem Informasi Reporting Community TB-HIV 'Aisyiyah Tanggamus)," Universitas Islam Indonesia, Yogyakarta, 2018.

[6] W. Lewis, Software Testing and Continuous Quality Improvement, Third Edition., Boston: Auerbach Publication, 2009.

[7] R. Black, Managing the Testing Process:Practical Tolls and Techniques for Managing Hardware and Software Testing, 2nd Edition., Hoboken: Wiley Publishing Inc, 2002.

[8] M. R. Rijal, "React dan TDD dalam 10 Menit — Part 1," Wonderlabs, 6 April 2017. [Online]. Available: https://medium.com/wonderlabs/react-dan-tdd-dalam-10-menit-part-1-98ee19807ff9. [Accessed 20 July 2020].

[9] Nostra Technology, "React Unit Testing Menggunakan Enzyme," Nostra Technology, 21 December 2018. [Online]. Available: http://blog.nostratech.com/2018/07/react-unit-testing-menggunakan-enzyme.html. [Accessed 20 July 2020].

PENGUJIAN DENGAN TEST CASE DAN UNIT TEST PADA PROYEK PENGEMBANGAN MODUL MANAJEMEN PENGGUNA

ORIGINALITY REPORT

10%	10%	2%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	id.scribd.com Internet Source	1%
2	www.slideshare.net Internet Source	1%
3	dspace.uii.ac.id Internet Source	1%
4	fransiscusgea.wordpress.com Internet Source	1%
5	123dok.com Internet Source	1%
6	Mirza Mahmood Baig, Ansar Ahmad Khan. "Plummeting the Software Testing Time Complexity", 2009 WRI World Congress on Software Engineering, 2009 Publication	1%
7	docplayer.info Internet Source	1%

8	media.neliti.com Internet Source	1%
9	www.altexsoft.com Internet Source	<1%
10	Ladan Tahvildari, Ajit Singh. "Software Bugs", Wiley, 1999 Publication	<1%
11	www.progweap.com Internet Source	<1%
12	Gunawan Wang, D Y Bernanda, J F Andry, Ahmad Nurul Fajar, Sfenrianto. "Application Development and Testing Based on ISO 9126 Framework", Journal of Physics: Conference Series, 2019 Publication	<1%
13	es.scribd.com Internet Source	<1%
14	lintangsekarsanti.wordpress.com Internet Source	<1%
15	12oktober.wordpress.com Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		