GENERATOR KODE UNIT *TESTING* UNTUK *JAVASCRIPT* BERBASIS *FRAMEWORK QUNIT*

Taufik Nur Adi
Program Studi Sistem Informasi, Fakultas Rekayasa Industri, Telkom University
taufik.nur.adi@gmail.com

Abstrak—Pada pengembangan perangkat lunak, kegiatan pengujian merupakan kegiatan yang sangat penting dalam menjaga kualitas perangkat lunak. Melakukan pengujian aplikasi yang memiliki kompleksitas yang tinggi memerlukan waktu dan biaya yang tidak sedikit. Maka dari itu diperlukan sebuah strategi dan teknik pengujian yang tepat agar pengujian perangkat lunak dapat dilakukan secara efektif dari segi waktu dan biaya. *Unit Testing* adalah salah satu teknik yang dapat digunakan untuk melakukan pengujian perangkat lunak. *Unit Testing* memiliki focus pengujian pada bagian terkecil dari sebuah aplikasi. Menulis kode *Unit Testing* dan *Test Case* secara manual ketika akan melakukan pengujian, memerlukan waktu dan sangat rentan terhadap kesalahan. Javascript memiliki fungsi-fungsi umum yang sering digunakan ketika membangun sebuah aplikasi. Fungsi-fungsi umum ini sangat dimungkinkan untuk diuji dengan sebuah kode *Unit Testing* yang dihasilkan secara otomatis oleh aplikasi *Generator*. Pada penelitian ini dihasilkan sebuah aplikasi *Generator* yang menghasilkan kode *Unit Testing* Javascript yang berjalan pada lingkungan pengujian berbasis *Framework* QUnit.

Kata kunci: *Javascript Unit Testing*, *Automated Unit Testing*, *Software Testing*, *Unit Testing Generator*, Generator Kode, Pengujian Unit.

I. PENDAHULUAN

Unit Testing adalah proses pengujian fungsi pada suatu aplikasi untuk memastikan bahwa fungsi yang digunakan terbebas dari *bug*. Untuk fungsi yang dikembangkan dengan menggunakan bahasa pemrograman Javascript, sudah tersedia *framework* untuk membantu dalam melakukan proses pengujian yaitu *framework* QUnit. Penggunaan *framework* untuk pengujian suatu fungsi akan sangat memudahkan proses pengujian karena pada *framework* sudah terdapat beberapa fungsi generik yang dapat langsung digunakan. *Framework* QUnit dikembangkan oleh pengembang JQuery yaitu John Resig dengan tujuan untuk membantu proses pengujian fungsi yang dibangun dengan bahasa pemrograman Javascript atau JQuery [1].

Untuk melakukan proses pengujian dengan menggunakan *framework* QUnit, penguji harus membuat kode *unit testing* yang terdiri dari fungsi generik QUnit dan kode *test case*. Fungsi-fungsi generik QUnit yang sering digunakan dalam pengujian, seharusnya dapat dihasilkan secara otomatis oleh aplikasi generator yang akan dikembangkan pada penelitian ini sehingga dapat menghemat waktu dalam pembuatan kode *unit testing*.

Permasalahan lain yang sering dihadapi oleh penguji adalah membuat kembali kode *unit testing* untuk fungsi-fungsi yang sering digunakan seperti validasi email, kartu kredit, no telp, url, dll. Selain itu, kadang pengembang ingin menguji sebuah fungsi javascript yang ada tanpa harus membuat kembali data kasus uji (*test case data*) jika fungsi tersebut adalah fungsi umum. Untuk fungsi yang tidak umum, diharapkan ada mekanisme yang memudahkan untuk menambahkan kasus uji yang sesuai dengan keinginan pengembang untuk diujikan pada fungsi tersebut.

Dari permasalahan diatas, aplikasi generator yang dikembangkan pada penelitian ini harus mampu menyediakan mekanisme pengotomatisasian dalam menghasilkan kode *unit testing* berserta kasus uji cobanya sehingga dapat mempermudah proses pengujian fungsi.

Penelitian sejenis sebenarnya sudah pernah dilakukan oleh Xie [12] dan Karine [13]. Pada penelitian yang dilakukan oleh Xie, sebuah kakas pengujian unit (*unit testing tools*) dibangun untuk membantu pengembang dalam melakukan pengujian terhadap sebuah bagian program yang dibangun menggunakan bahasa pemrograman Java. Kakas pengujian yang dibangun pada penelitian ini menekankan kepada sekumpulan data kasus yang diperlukan pada pengujian sehingga pengembang dapat bekerja secara efektif ketika melakukan pengujian unit. Sedangkan pada penelitian Karine [13], kakas pengujian unit yang dibangun ditujukan untuk menguji sebuah bagian program yang dibangung menggunakan bahasa pemrograman Eiffel.

II. TINJAUAN PUSTAKA

A. Pengujian Perangkat Lunak

Pengujian perangkat lunak (*software testing*) adalah salah satu aktifitas dari jaminan kualitas perangkat lunak. Pada awalnya, pengujian perangkat lunak hanya dilakukan pada tahap akhir pengembangan perangkat lunak. Tetapi dengan berkembangnya konsep jaminan kualitas perangkat lunak, pendeteksian dini terhadap *error* dan *defect* pada perangkat lunak menjadi semakin penting dalam rangka untuk mengurangi biaya dan usaha yang harus dilakukan pada tahap pemeliharaan. Saat ini, pengujian perangkat lunak dilakukan sejak awal dan dimulai dari tingkat yang paling rendah yaitu modul atau *unit testing* dan *integration testing*.

Berdasarkan Standart IEEE 610.12, *software testing* memiliki definisi sebagai berikut [2]:

1. Proses dalam mengoperasikan sebuah sistem atau komponen pada suatu kondisi, mengamati atau mencatat

hasilnya, dan membuat sebuah tindakan evaluasi terhadap beberapa aspek sistem atau komponen.

2. Proses analisa sebuah *item software* untuk mendeteksi perbedaan antara kondisi yang ada dengan kondisi yang diharapkan (yaitu, bug) dan untuk mengevaluasi fitur dari *item software*.

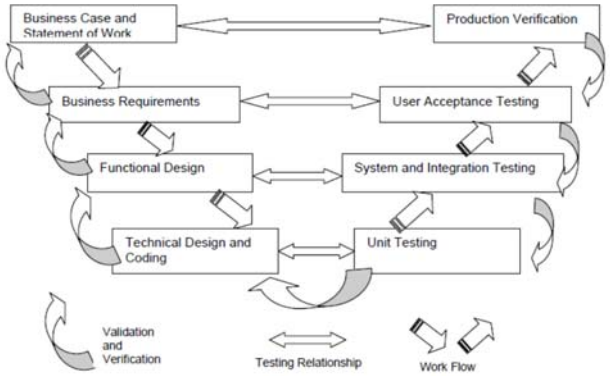
Sedangkan definisi *software testing* menurut Galin adalah sebagai berikut [3]:

"*Software testing* adalah proses formal yang dilakukan oleh sebuah tim penguji khusus, dimana suatu *software unit* yang terintegrasi atau sebuah paket *software* yang lengkap, diperiksa dengan menjalankan program tersebut pada komputer. Semua yang terkait dengan pengujian, dilakukan sesuai dengan prosedur dan kasus uji coba yang telah disetujui".

Dari kedua definisi diatas, terlihat sangat jelas bahwa tujuan utama dari pengujian perangkat lunak adalah

1. Untuk mengidentifikasi dan mengungkapkan *error* sebanyak mungkin pada perangkat lunak yang sedang diuji.
2. Untuk membawa perangkat lunak yang sedang diuji pada tingkatan kualitas yang dapat diterima setelah melakukan perbaikan terhadap *error* yang teridentifikasi dan melakukan pengujian kembali.
3. Untuk melakukan pengujian yang diperlukan secara efektif dan efisien terhadap keterbatasan waktu dan anggaran.

Pendeteksian *error* dan *defect* sejak awal menjadi hal yang sangat penting untuk mengurangi dampak, biaya dan usaha yang diperlukan ketika proses pengujian hanya dilakukan pada tahap akhir pengembangan. Saat ini aktifitas pengujian dilakukan disepanjang siklus hidup pengembangan perangkat lunak. Hal ini dikarenakan bentuk pengujian disetiap tahap pada sikul hidup pengembangan perangkat lunak adalah berbeda. Gambar 1 dibawah ini menggambarkan perbedaan bentuk pengujian pada masing-masing tahap pengembangan.



Gambar 1 V-Model pengujian perangkat lunak [4]

Gambar 1 diatas menggambarkan bahwa setiap tahapan pengembangan perangkat lunak (sebelah kiri) memiliki keterkaitan dengan aktifitas pengujian yang sesuai (sebelah kanan). Pada penerapannya, konsep tadi bisa disesuaikan dengan model atau metode pengembangan yang digunakan oleh pihak pengembang. Hal yang paling penting disini adalah bagaimana setiap tahap pengembangan memiliki cara untuk bisa memastikan bahwa apa yang dilaksanakan itu sesuai dengan apa yang diharapkan. Dalam hal ini adalah aktifitas pengujian pada masing-masing tahap pengembangan untuk menemukan kecacatan. Sehingga kecacatan yang ada dapat ditemukan sejak dini dan mengurangi resiko yang jauh lebih besar, baik dari segi biaya maupun usaha yang diperlukan.

B. Teknik Pengujian

Secara konsep, terdapat dua teknik untuk melakukan pengujian sebuah perangkat lunak, yaitu:

1. ***Black Box Testing*** (*Functionality Testing*). Berdasarkan IEEE (1990), definisi Black Box Testing adalah (1) proses pengujian dimana mekanisme internal dari sebuah komponen atau sistem diabaikan dan berfokus kepada kondisi eksekusi serta nilai keluaran yang dihasilkan sebagai respon terhadap nilai input yang dipilih. (2) Proses pengujian yang dilakukan untuk mengevaluasi pemenuhan sistem atau komponen dengan kebutuhan fungsional tertentu [2].
2. ***White Box Testing*** (*Structural Testing*). Proses pengujian yang berfokus kepada struktur dan mekanisme internal dari suatu komponen atau sistem.

C. Strategi Pengujian

Strategi pengujian perangkat lunak akan memberikan *road-map* kepada pengembang mengenai langkah-langkah yang harus dilakukan sebagai bagian dari proses pengujian secara keseluruhan. Dengan strategi pengujian, pengembang akan menilai apakah proses pengujian hanya akan dilakukan pada bagian terkecil dari sistem (*unit*) atau akan dilakukan ketika perangkat lunak yang dikembangkan sudah menjadi satu kesatuan utuh.

Strategi pengujian juga menentukan apakah proses pengujian ulang akan dilakukan jika terjadi penambahan modul atau komponen baru. Secara umum, ada dua bentuk strategi pengujian perangkat lunak, yaitu :

1. ***Big Bang Testing***. Pengujian yang dilakukan setelah perangkat lunak yang dikembangkan telah rampung secara keseluruhan.
2. ***Incremental Testing***. Pengujian yang dilakukan secara bertahap. Dimulai dari pengujian pada tingkat yang paling rendah yaitu *unit testing*, lalu pengujian terhadap sekumpulan unit atau modul yaitu *integration testing* hingga pengujian terhadap seluruh modul telah diintegrasikan yaitu *system testing*.

D. Unit Testing

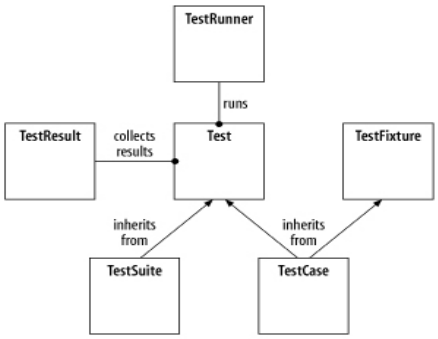
Menurut J. Myers, *unit testing* adalah proses pengujian secara individual terhadap *subprogram, subroutine* atau *prosedur* pada sebuah program [5]. *Unit testing* berfokus kepada bagian terkecil sebuah program. Tujuan dari *unit testing* ini adalah memastikan bahwa fungsi yang telah dibuat dan akan diintegrasikan dengan fungsi lainnya dapat berjalan baik dalam arti fungsi tersebut menghasilkan keluaran (*output*) yang benar.

Selain itu dengan melakukan *unit testing,* pengembang dalam mengidentifikasikan kesalahan sejak dini sehingga kesalahan yang lebih besar dapat dihindari. Pada umumnya, penguji akan melakukan *unit testing* dengan cara membuat sekumpulan *method* atau *prosedur* sederhana untuk menguji fungsi yang sedang dalam pengujian. Proses seperti ini biasanya dilakukan secara manual oleh penguji. Saat ini, proses pengujian dapat dijalankan secara otomatis dengan menggunakan *unit testing framework.*

1. *Unit Testing Framework*

Menurut Hamill, *unit testing framework* adalah perangkat lunak yang membantu dalam menuliskan kode *unit test,* menjalankan *unit test* dan menghasilkan laporan hasil pengujian [6]. Berbeda dengan pendapat Wells, *unit testing framework* bukan saja berupa perangkat lunak yang dapat membantu dalam proses pengujian. Melainkan memiliki ruang lingkup yang lebih luas yang meliputi kerangka kerja dan prosedur dalam menyusun kebutuhan (*requirement*), klarifikasi arsitektur perangkat lunak, penulisan kode program, *debug* kode program, integrasi kode program dan *release* kode program [7].

Sebagian besar *unit testing framework* yang ada saat ini mengadopsi ide Kent Beck sebagai pencipta *unit testing framework* untuk bahasa SmallTalk (SUnit) pada tahun 1999 dan semua *unit testing framework* yang mendopsi konsep ini dikenal dengan xUnit. Semua xUnit memiliki konsep dan arsitektur dasar yang sama. Secara umum arsitektur xUnit tampak pada Gambar 2 berikut ini:



Gambar 2 Arsitektur generic xUnit

Penerapan arsitektur *unit testing framework* ini bisa bervariasi untuk pengujian bahasa pemrograman tertentu. Tidak lama sejak Kent Beck mempublikasikan SUnit, Erich Gamma membuat JUnit yang pada akhirnya konsep JUnit ini yang banyak menurunkan *unit testing framework* untuk bahasa pemrograman lainnya seperti CppUnit untuk C++, PyUnit untuk Pyton, NUnit untuk dotnet, dsb.

E. Automated Testing

Automated Testing adalah sebuah proses pengujian perangkat lunak yang dilakukan secara otomatis dengan bantuan *tools* tertentu. Motivasi utama dari penggunaan *tools* dalam *automated testing* ini adalah mengurangi biaya pengujian, mengurangi durasi pelaksanaan pengujian, mengurangi *rework* pengembang dalam melakukan pengujian dan memungkinkan pengujian yang tidak dapat dilakukan dengan pengujian manual. Secara umum, *automated testing* terdiri dari empat fase proses, yaitu [3]:

1. ***Test Planning.*** Pada fase ini, menentukan apa saja yang akan diuji dan menentukan prioritas pengujian. Selain itu, pada fase ini juga pengembang perangkat lunak menentukan teknik pengujian yang akan dilakukan pada bagian-bagian tertentu pada aplikasi dan menentukan kategori pengujian seperti *functionality test, performance test, security test,* dsb
2. ***Test Design***. Pada fase ini, pengembang perangkat lunak menentukan bagaimana sesuatu yang telah ditentukan pada fase perencanaan itu akan diuji. Hasil dari fase ini adalah berupa *test case* yang berhubungan dengan *test objective*. *Test case* yang dihasilkan pada fase ini berupa sejumlah pengujian yang terdiri dari nilai input tertentu, keluaran yang diharapkan (*expected outcome*) dan informasi yang dibutuhkan untuk melaksanakan pengujian seperti kebutuhan lingkungan pengujian (*environment prerequisites*).
3. ***Test Performance.*** Melaksanakan pengujian dengan menjalankan *tools* pengujian. Pada fase ini, hasil pengujian dibandingkan dengan keluaran yang diharapkan (*expected outcome*).
4. ***Re-testing After Correction.*** Melaksanakan pengujian ulang setelah melakukan perbaikan kesalahan yang telah terdeteksi pada pengujian sebelumnya.

F. Framework QUnit

QUnit adalah *unit testing framework* yang pada awalnya digunakan pada proyek JQuery untuk melakukan *unit testing.* Qunit juga dapat digunakan untuk melakukan pengujian pada kode generik javascriPT Berdasarkan situs resminya, QUnit juga memiliki kemampuan untuk menguji kode javascript yang berada pada server.

QUnit sangat mirip dengan *unit testing framework* lainnya seperti JUnit tetapi khusus untuk pengujian kode javascript pada browser dan memiliki fitur untuk pengujian *asynchronous*. Untuk melakukan pengujian menggunakan QUnit, diperlukan sebuah halaman HTML yang mengikutsertakan file `qunit.js` dan `qunit.css`.

Pada *Framework Qunit* terdapat dua jenis pengujian, yaitu pengujian *Syncronous* dan pengujian *Asyncronous.* Pengujian *Syncronous* digunakan untuk menguji sebuah fungsi Javascript dengan menggunakan *assertion.* Sedangkan pengujian *Asyncronous* digunakan untuk melakukan pengujian terhadap fungsi Javascript yang mengimplementasikan AJAX.

Pengujian *Syncronous* adalah pengujian yang dilakukan terhadap sebuah fungsi Javascript yang ditulis pada sebuah halaman HTML dan tidak mengimplementasikan AJAX yang memerlukan sumber data dari halaman HTML lain. Berbeda dengan pengujian *Asyncronous* yang dilakukan pada fungsi Javascript yang memerlukan data dari halaman HTML lain. Pada pengujian *Asyncronous* terdapat waktu *timeout* yang perlu diset agar tidak terjadi *infinite loading* ketika fungsi Javascript yang sedang diuji gagal memanggil data dari halaman HTML lain.

III. ANALISIS DAN PERANCANGAN

A. Analisis Prosedur *Unit Testing* menggunakan *Framework* QUnit

Analisis prosedur *unit testing* dengan *framework* QUnit diperlukan untuk memetakan beberapa prosedur yang akan diotomatisasikan oleh aplikasi generator. Prosedur yang akan diotomatisasikan adalah prosedur yang sering dilakukan secara berulang kali ketika akan melakukan *unit testing*. Berikut ini merupakan beberapa prosedur umum yang sering dilakukan berulang-ulang jika akan melakukan *unit testing* fungsi javascript dan JQuery menggunakan *framework* QUnit:

1. Membuat dan mengatur lingkungan ujicoba (*Test Suite*). Lingkungan uji coba ini berfungsi untuk menjalankan *Framework* QUnit dalam melakukan *unit testing* dengan menyertakan beberapa *file* penting terlebih dahulu seperti HTML, CSS, Javascript dan JQuery.
2. Menentukan jenis pengujian. (*synchronous* atau *asyncronous*)
3. Menentukan *assertions* yang akan digunakan.
4. Membuat data kasus ujicoba (*Test Case Data*).

B. Analisis Aplikasi Generator

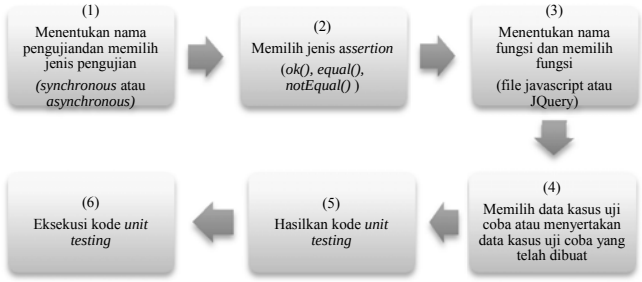
Dari hasil analisis prosedur *unit testing* terlihat bahwa untuk menghasilkan sebuah kode *unit testing* memerlukan:

1. Halaman *template* yang berupa *file* HTML sebagai lingkungan uji coba.
2. Jenis uji coba yang akan dilakukan seperti uji coba *synchronous* dan *asynchronous.*
3. Jenis *assertion* yang akan digunakan pada uji coba.
4. Memilih fungsi javascript atau JQuery yang akan diuji (dalam bentuk *file*).
5. Data kasus uji coba.

Disini aplikasi generator akan menghasilkan sebuah kode *unit testing* jika pengguna telah menentukan jenis uji coba, jenis *assertion,* memilih fungsi javascript atau JQuery serta memilih data kasus uji yang akan digunakan untuk pengujian. Aplikasi generator yang dikembangkan pada penelitian ini akan memandu pengguna dalam bentuk *wizard* untuk menentukan hal-hal yang diperlukan dalam menghasilkan kode *unit testing*.

C. Analisis dan Perancangan Struktur File XML Konfigurasi

Berdasarkan Gambar 3 bahwa aplikasi generator memerlukan sebuah *file* untuk menyimpan konfigurasi yang berasal dari masukan pengguna.



Gambar 3 Alur *wizard* aplikasi generator

Beberapa nilai penting yang perlu disimpan pada *file* konfigurasi adalah nama pengujian, jenis pengujian, jenis *assertion*, nama fungsi, nama *file* javascript atau JQuery, pilihan data kasus uji dan nama *file* data kasus uji yang dihasilkan oleh aplikasi generator. Oleh karena itu, struktur *file* XML konfigurasi aplikasi harus memiliki tag XML sebagai berikut :

1. **`<config>`** adalah *tag* yang menyatakan bahwa elemen yang diapit oleh *tag* ini adalah struktur elemen untuk data konfigurasi.
2. **`<testtype>`** adalah *tag* yang berisi informasi jenis pengujian (*asynchronous* atau *synchronous*).
3. **`<atype>`** adalah *tag* yang berisi informasi jenis *assertion*.
4. **`<functionname>`** adalah *tag* yang berisi informasi nama fungsi yang akan diuji.
5. **`<numparam>`** adalah *tag* yang berisi informasi jumlah parameter dari fungsi yang akan diuji.
6. **`<functionfile>`** adalah *tag* yang berisi informasi nama file javacript atau JQuery yang berisi kode fungsi yang akan diuji.
7. **`<tctype>`** adalah *tag* yang berisi informasi pilihan *file* data kasus uji (menggunakan *file default* atau *custom*).
8. **`<tcfile>`** adalah *tag* yang berisi informasi nama *file* data kasus uji yang digunakan untuk melakukan pengujian.

Berikut ini adalah contoh implementasi XML Konfigurasi untuk melakukan pengujian Fungsi Tunggal Javascript:

```
<config>
        <testname>Pengujian Fungsi
Tambah</testname>
        <testtype>synchronous</testtype>
        <atype>ok</atype>
        <functionname>tambah</functionname>
        <numparam>2</numparam>
        <functionfile>tambah.js</functionfile>
        <tctype>default</tctype>
        <tcfile>tambah.xml</tcfile>
</config>
```

Kode 1 XML Konfigurasi Pengujian Fungsi Tunggal Javascript

```
<config>
  <testname>bilGenap-tambahlagi</testname>
  <testtype>multifunction</testtype>
  <functionFile>f_multifungsi.js</functionFile>
<function>
    <atype>ok</atype>
    <functionname>bilGenap</functionname>
    <numparam>1</numparam>
    <tctype>default</tctype>
    <tcfile>tc_ok_1_bilgenap.xml</tcfile>
  </function>
<function>
    <atype>equal</atype>
    <functionname>tambahlagi</functionname>
    <numparam>2</numparam>
    <tctype>default</tctype>
    <tcfile>tc_equal_2_tambah.xml</tcfile>
  </function>
</config>
```

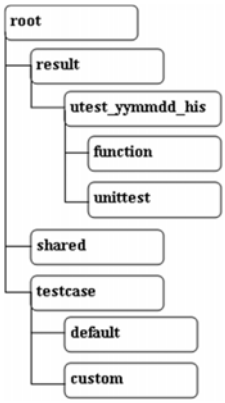
Kode 2 ***XML Konfigurasi Pengujian Multi Fungsi Javascript***

D. Analisis dan Perancangan Struktur Direktori Aplikasi

Aplikasi generator ini memerlukan beberapa direktori yang berfungsi untuk menyimpan *fili-file* berikut ini :

1. *file* data kasus uji *default*.
2. *file* qunit.css, qunit.js, jquery-1.4.2.min yang selalu disertakan di dalam *file testsuite* untuk semua pengujian.
3. *file testsuite* untuk masing-masing pengujian.
4. *file* javascript atau JQuery yang berisi kode fungsi yang akan diuji.
5. *file* data kasus uji yang dibuat oleh pengguna (*custom*).
6. *file unit testing* untuk masing-masing pengujian.

Berdasarkan *file-file* diatas maka struktur direktori aplikasi generator pada penilitian ini adalah sebagai berikut:



Gambar 4 Struktur direktori aplikasi generator

Berikut ini adalah keterangan Gambar 4:

1. **root/result.** *file-file* yang diperlukan untuk setiap pengujian akan disimpan disini. Masing-masing pengujian memiliki nama direktori yang unik yang merupakan kombinasi dari *prefix* "utest_" diikuti dengan tahun, bulan, tanggal, jam, menit dan detik. Pola penamaan ini berdasarkan waktu kapan sebuah kode *unit testing* dihasilkan dengan menggunakan aplikasi generator sehingga untuk setiap pengujian memiliki nama direktori yang unik.
2. **root/result/utest_yymmdd_his/.** Direktori utama untuk masing-masing pengujian dimana *file testsuite* disimpan.
3. **root/result/utest_yymmdd_his/function.** Direktori ini menyimpan *file* javascript atau JQuery yang berisi kode fungsi yang akan diuji.
4. **root/result/utest_yymmdd_his/unittest.** Direktori dimana *file* yang berisi kode *unit testing* disimpan.
5. **root/shared.** Direktori yang menyimpan *file* qunit.css, qunit.js dan jquery-1.4.2.min.js.
6. **root/testcase.** Direktori yang menyimpan seluruh *file-file* data kasus uji coba.
7. **root/testcase/default.** Direktori yang menyimpan *file* data kasus uji coba *default.*
8. **root/testcase/custom.** Direktori yang menyimpan *file* data kasus uji coba yang ditambahkan oleh pengguna.

E. Analisis dan Perancangan Struktur XML Data Kasus Ujicoba

Struktur XML data kasus uji coba ini dirancang berdasarkan jenis *assertion* yang telah dijelaskan pada bab sebelumnya dengan melihat banyaknya parameter yang ada. Jika dibedakan berdasarkan banyaknya parameter yang ada, jenis *assertion* terdapat dua jenis, yaitu *assertion* dengan dua parameter seperti *assertion ok()*, dan *assertion* dengan tiga parameter seperti *assertion equal(), notEqual(), deepEqual(), notDeepEqual(), strictEqual(), notStrictEqual(), raises()*. Secara umum, struktur XML data kasus uji harus memiliki tag XML sebagai berikut:

1. **`<testcase>`** adalah *tag* yang menyatakan bahwa elemen yang diapit oleh *tag* ini adalah struktur elemen untuk data kasus uji.
2. **`<atype>`** adalah *tag* yang berisi informasi jenis *assertion*.
3. **`<tcname>`** adalah *tag* yang berisi informasi nama data kasus uji.
4. **`<tcdata>`** adalah *tag* pengapit untuk elemen data kasus uji.

F. Analisis Fungsi-fungsi Umum dan Data Kasus Uji

Pada bagian ini terdapat tujuh fungsi javascript dan satu fungsi JQuery yang akan digunakan sebagai studi kasus dalam pengujian aplikasi generator untuk menghasilkan kode *unit testing.* Fungsi-fungsi tersebut juga dianalisa untuk menentukan bentuk data kasus uji yang sesuai. Data kasus uji yang dianalisa pada bagian ini akan menjadi pustaka data kasus uji bagi aplikasi generator yang dikembangkan. Berikut adalah fungsi javascript dan JQuery yang akan dianalisa:

1) Fungsi Javascript

a. Fungsi validasi email.
b. Fungsi validasi username.
c. Fungsi pengecekan hanya angka.
d. Fungsi pengecekan hanya huruf.
e. Fungsi pengecekan hanya angka dan huruf.
f. Fungsi pengecekan format tanggal.
g. Fungsi AJAX (String Response)

2) Fungsi JQuery

a. Fungsi AJAX (String Response)

G. Perancangan Aplikasi Generator

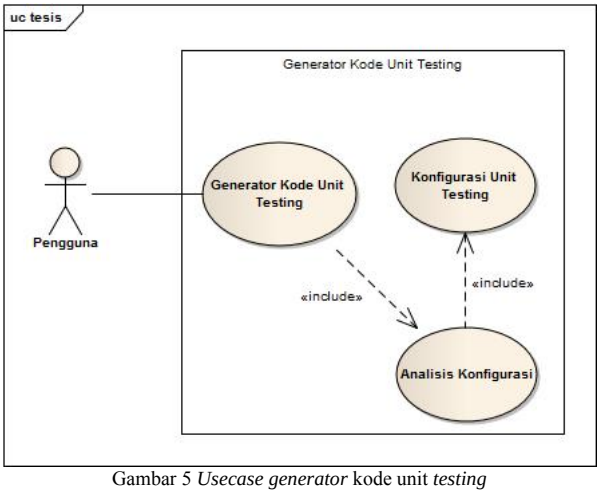
Aplikasi generator kode *unit testing* yang dikembangkan pada penelitian ini akan menghasilkan sebuah kode *unit testing* yang sesuai jika pengguna telah memberikan masukan yang diperlukan untuk menghasilkan kode *unit testing.*

Pada bagian sebelumnya telah dijelaskan bahwa sebuah *wizard* akan memandu pengguna aplikasi generator untuk memberikan beberapa masukan penting yang nantinya masukan ini akan disimpan pada *file* konfigurasi. *File* konfigurasi inilah yang menentukan bentuk kode *unit testing* yang akan dihasilkan oleh aplikasi generator. Gambar 5 adalah *usecase* dari aplikasi generator *unit testing*.

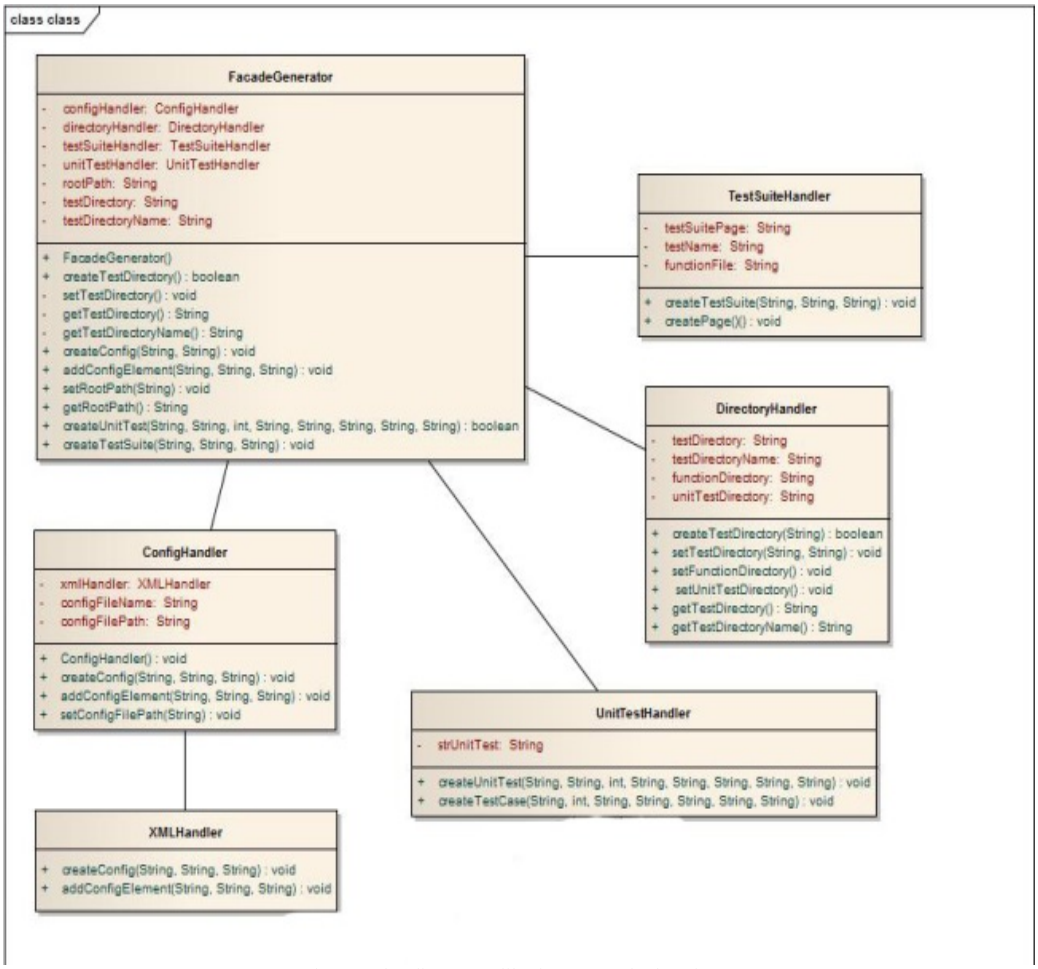
Pendekatan yang digunakan untuk melakukan perancangan aplikasi generator kode adalah pendekatan berorientasi objek. Perancangan ini dituangkan ke dalam empat model diagram UML yaitu diagram kelas (*class diagram*), diagram sekuen (*sequence diagram*), diagram

aktifitas (*activity diagram*), diagram komunikasi (*communication diagram*) dan diagram komunikasi (*communication diagram*).

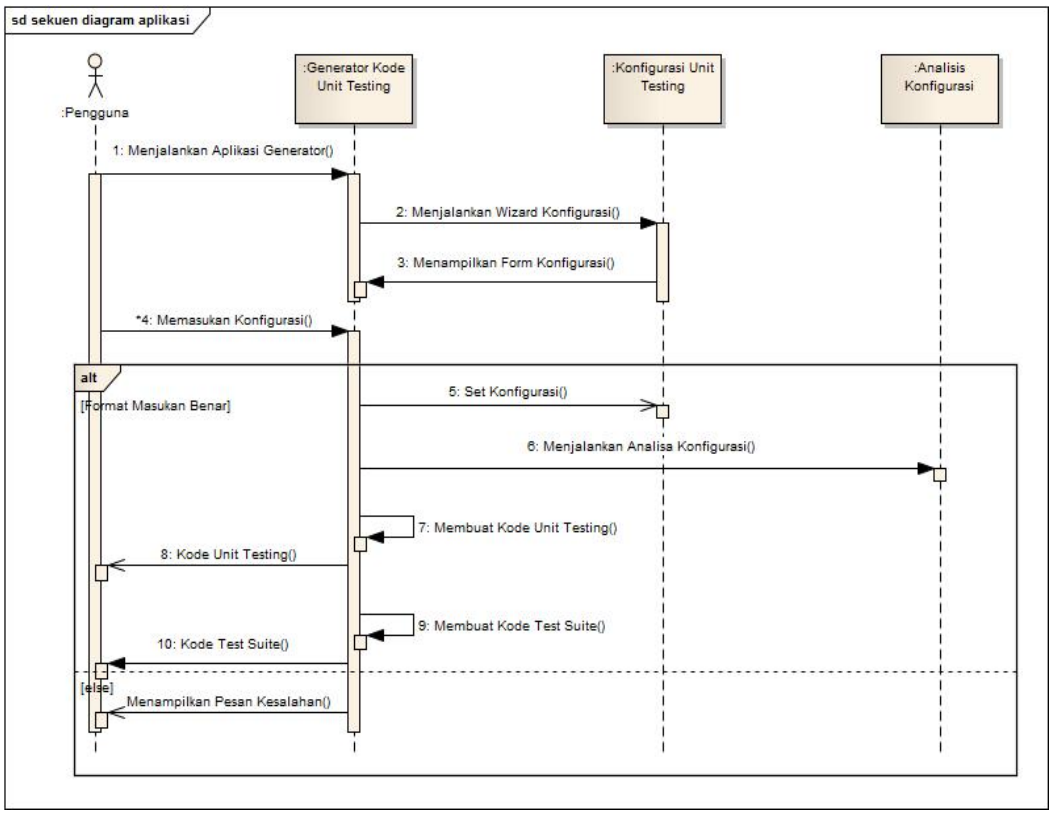
Gambar 6 adalah kelas diagram yang menggambarkan hubungan antar kelas pada aplikasi generator. Gambar 7 adalah diagram sekuen aplikasi generator kode *unit testing* yang menggambarkan hubungan antar komponen dalam aplikasi. Gambar 8 adalah diagram komunikasi aplikasi generator kode *unit testing* yang menggambarkan hubungan komunikasi antar subproses dalam aplikasi. Gambar 9 adalah diagram aktifitas aplikasi generator kode *unit testing* yang menggambarkan urutan aktifitas pada aplikasi.



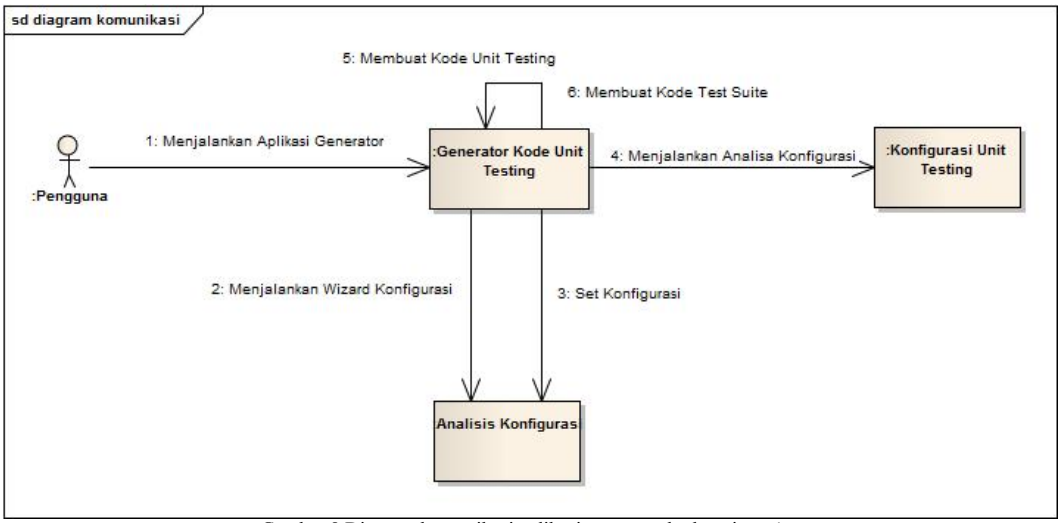
Gambar 5 *Usecase generator* kode unit *testing*



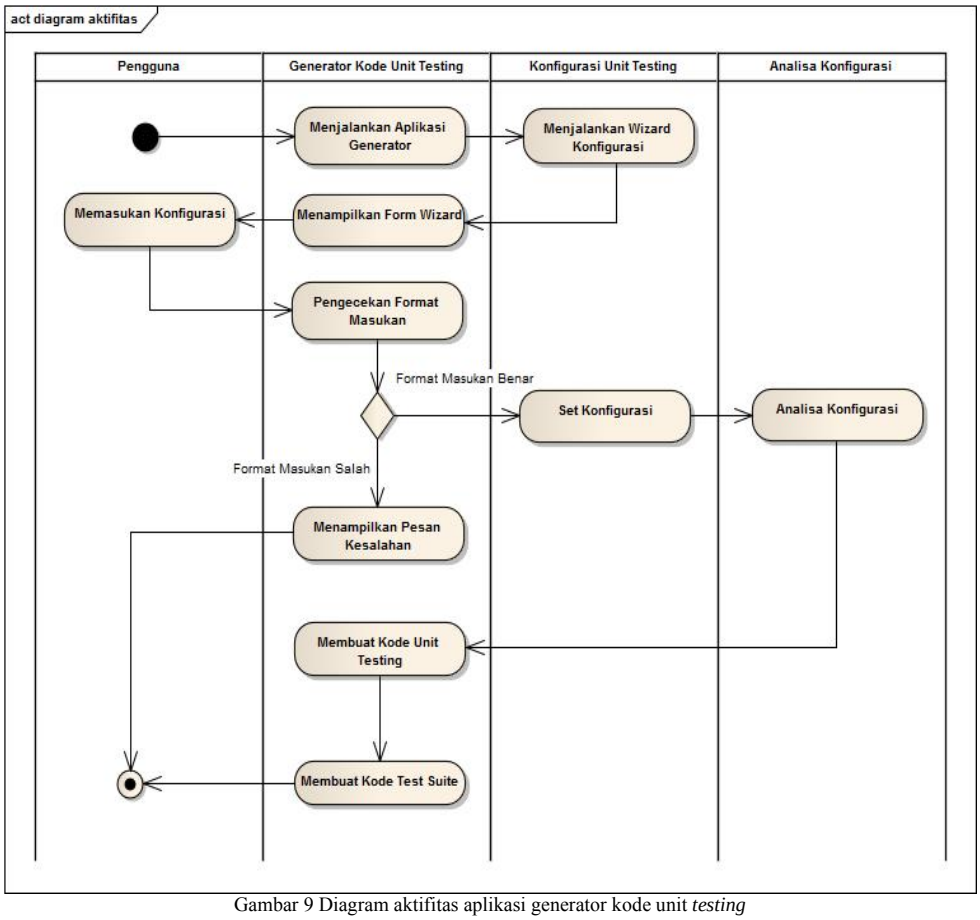
Gambar 6 Kelas diagram aplikasi generator kode unit *testing*



Gambar 7 Diagram sekuen aplikasi generator kode unit *testing*



Gambar 8 Diagram komunikasi aplikasi generator kode unit *testing*



Gambar 9 Diagram aktifitas aplikasi generator kode unit *testing*

IV. PENGUJIAN

Pengujian aplikasi generator *unit testing* ini menggunakan metode *black box* dan dilakukan berdasarkan beberapa skenario yang telah ditetapkan pada penelitian ini (Tabel I).

TABEL I
TABEL HASIL PENGUJIAN

Kriteria Pengujian	Hasil Pengujian
Apakah aplikasi generator dapat menghasilkan struktur direktori yang sesuai dengan yang telah dipaparkan pada bagian analisis dan perancangan?	Aplikasi telah menghasilkan struktur direktori yang seharusnya.
Apakah aplikasi generator dapat membaca data kasus uji dan menghasilkan kode serta *file unit testing* yang sesuai dengan struktur *unit testing* yang digunakan pada *framework* QUnit ?	Aplikasi telah menghasilkan kode dan *file unit testing* yang sesuai dengan struktur yang digunakan pada *framework* QUnit.
Apakah aplikasi generator dapat menghasilkan kode *unit testing* sesuai dengan jenis pengujian dan jenis *assertion*?	Aplikasi telah menghasilkan kode *unit testing* sesuai dengan jenis pengujian dan jenis *assertion*.
Apakah aplikasi generator dapat menyimpan *file* fungsi yang akan diuji pada direktori yang telah ditentukan?	Aplikasi telah menyimpan *file* fungsi pada direktori yang sesuai.
Apakah aplikasi generator dapat menghasilkan *file test suite* yang telah menyertakan *file* fungsi, *unit testing* serta *file-file* yang diperlukan dalam melakukan pengujian menggunakan *framework* QUnit?	Aplikasi telah menghasilkan *file test suite* yang sesuai.
Apakah *file unit testing* dan *test suite* yang dihasilkan oleh aplikasi generator dapat berjalan sesuai harapan?	*file unit testing* dan *test suite* yang dihasilkan dapat dijalankan sesuai dengan harapan.
Apakah *test suite* yang dihasilkan oleh aplikasi generator menampilkan hasil pengujian yang benar?	*Test suite* yang dihasilkan telah menampilkan hasil pengujian yang benar.

Berdasarkan hasil pengujian, maka hipotesa:

1. Generator kode *unit testing* akan membantu mempercepat penguji dalam membuat kode *unit testing* untuk pengujian fungsi-fungsi javascript yang lingkungan pengujiannya menggunakan *framework* QUnit.
2. Ada fungsi generik QUnit dan *test case* untuk fungsi-fungsi javascript kasus umum (*common case*) yang dapat diotomatisasikan.
3. Penambahan data kasus uji (*test case*) untuk fungsi javascript tertentu dapat dilakukan dengan mekanisme yang lebih memudahkan pengembang.

adalah benar.

Dari segi pengukuran waktu, ada penelitian ini mencoba membandingkan antara aktifitas pengujian yang dilakukan secara manual dengan aktifitas pengujian yang dilakukan dengan

menggunakan bantuan aplikasi ini. Tabel II adalah hasil pengukurannya.

TABEL II
TABEL HASIL PENGUKURAN

Aktifitas Pengujian	Manual	Menggunakan Aplikasi
Membuat Kode Mode Pengujian Unit untuk satu fungsi JavascriPT	5 Menit (Pemetaan Manual)	10 Detik (Pemetaan Otomatis)
Membuat Kode Mode Pengujian Unit untuk menguji lebih dari satu fungsi JavascriPT (Min. 3 Fungi)	15 Menit (Pemetaan Manual)	10 Detik (Pemetaan Otomatis)
Membuat Data Kasus Pengujian untuk satu Fungsi JavascriPT	10 Menit (Mengetik Manual)	3 Detik (Memilih TestCase Data yang tersedia)
Membuat Data Kasus Pengujian untuk menguji lebih dari satu fungsi JavascriPT (Min. 3 Fungi)	>25 Menit (Mengetik Manual)	8 Detik (Memilih TestCase Data yang tersedia)

Pengukuran waktu diatas adalah pengujian untuk fungsi Javascript Umum seperti: a. Fungsi validasi email, validasi username, pengecekan hanya angka, pengecekan hanya huruf, pengecekan hanya angka dan huruf, pengecekan format tanggal, pengecekan fungsi AJAX dengan Javascript dan JQuery.

V. KESIMPULAN

Pada penelitian ini terlihat bahwa sebuah fungsi javascript yang akan dihasilkan kode *unit testing*-nya dengan menggunakan aplikasi kode generator dapat dilakukan melalui dua cara yaitu dengan cara memasukkan nama fungsi dan jumlah parameter fungsi secara manual atau dengan cara mendeteksi nama fungsi dan jumlah parameter fungsi secara otomatis. Cara kedua jauh lebih cepat karena pengguna tidak perlu lagi memasukan nama fungsi dan jumlah parameter secara manual yang rentan terhadap kesalahan penulisan. Tetapi proses pedeteksian nama dan jumlah parameter sebuah fungsi tidak berjalan baik terhadap bentuk fungsi yang lebih kompleks seperti:

1. **Fungsi Bersarang (*Nested Function*)**. Pada javascript 1.2 dan ECMAScript v3, sebuah fungsi dapat dideklarasikan dalam sebuah fungsi.
2. **Fungsi *Constructor* (*Constructor Function*)**. Sejak javascript 1.1 dan ECMAScript v1, sebuah fungsi javascript dapat dibuat secara dinamis menggunakan fungsi *constructor*.
3. **Fungsi Literal (*Literal Function*)**. ECMAScript v3 dan javascript 1.2 mengimplementasi sebuah cara untuk membuat sebuah fungsi javascript dimana sebuah fungsi dapat dibuat tanpa atau dengan nama fungsi.
4. **Fungsi *Closure*. (*Closure Function*)**. Javascript 1.8 dan ECMAScript v4 mengimplementasikan sebuah cara untuk membuat sebuah fungsi dengan sebuah ekspresi javascript dan memperbolehkan untuk memiliki variabel bebas.

Proses identifikasi data kasus uji yang sering digunakan untuk fungsi-fungsi umum javascript dapat membantu pengguna dalam menghasilkan kode *unit testing* karena pengguna cukup memilih data kasus uji yang tersedia. Fasilitas untuk melakukan penambahan data kasus uji juga sangat membantu pengguna jika ingin menggunakan data kasus uji yang spesifik.

Secara umum, aplikasi generator ini dapat membantu pengguna dalam melakukan *unit testing*. Kode *unit testing* yang dihasilkan dapat dijaga konsistensinya dan mengurangi tingkat kesalahan penulisan kode *unit testing*. Selain itu, dengan menggunakan aplikasi generator, proses pembuatan kode *unit testing* dapat dilakukan dalam waktu yang lebih singkat jika dibandingkan dengan cara manual.

DAFTAR PUSTAKA

[1] JQuery Community Expert, *JQuery Cookbook.* O'Reilly, 2009.

[2] IEEE, *IEEE Std 610.12-1990 – IEEE Standard Glossary of Software Engineering Terminology*, Corrected Edition, in *IEEE Software Engineering Standards Collection*, The Institute of Electrical and Electronics Engineers, New York, 1991.

[3] Galin, Daniel. *Software Quality Assurance from Theory to Implementation*, Pearson Addison Wesley, 2004.

[4] Goldsmith, Robin F., *Software Development Magazine*, 4-part series, July-October, 2002.

[5] J. Myers, Glenford. *The Art of Software Testing 2nd Edition.* John Wiley & Sons, Inc, 2004.

[6] Hamill, Paul. *Unit Test Frameworks*. O'Reilly, 2004.

[7] Wells, Don. *Unit Testing Framework*. http:// www.extremeprogramming.org, diakses Desember 2011.

[8] Beck, Kent., *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley, 2000.

[9] JQuery Community Expert, *JQuery Cookbook.* O'Reilly, 2009.

[10] Schroeder, P. J. dan Rothe, D., *Teaching Unit Testing using Test-Driven Development*. Milwaukee School of Engineering, Milwaukee, 2005.

[11] Stock, Manfred., *Automatic Generation of JUnit Test-Harnesses.* Swiss Federal Institute of Technology Zurich, Swiss, 2007.

[12] T. Xie and N. David., *Tool-Assisted Unit-Test Generation and Selection Based on Operational Abstractions.* 2003.

[13] A. Karine., R. Xavier and M. Bertrand., *Test Wizard: Automatic test case generation based on Design by Contract™.* 2002.