

Pengembangan Kakas Bantu Pembangkitan Kasus Uji pada *Model-Based Testing* Berdasarkan *Activity Diagram*

Arlian Gutama¹, Achmad Arwan², Lutfi Fanani³

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: ¹gutama@student.ub.ac.id, ²arwan@ub.ac.id, ³lutfifanani@ub.ac.id

Abstrak

Pengujian merupakan salah satu tahap yang paling krusial dalam pengembangan perangkat lunak. Kompleksitas dalam pengujian sistem menyebabkan kebutuhan akan kakas bantu yang dapat menentukan kasus uji secara otomatis. *Activity diagram* merupakan salah satu UML *behavioural model* yang cocok untuk pengujian sistem, karena *activity diagram* dapat menggambarkan alur dari sebuah sistem secara keseluruhan. Oleh karena itu penelitian ini dimaksudkan untuk mengembangkan suatu kakas bantu pembangkitan yang dapat menentukan kasus uji pada *model-based testing* berdasarkan *activity diagram* secara otomatis. Teknik yang dipakai dalam pembangkitan kasus uji dengan membangun sebuah *dependency flow tree* (DFT) yang menampung informasi dari *file activity diagram* ArgoUML melalui bantuan sebuah *parser*. Kemudian DFT tersebut diproses dengan sebuah algoritme *depth first search* (DFS) yang sudah dimodifikasi untuk menelusuri setiap jalur dari kasus uji. Seluruh kebutuhan dari kakas bantu merupakan hasil elisitasi kebutuhan pada kajian pustaka pada penelitian terkait dan observasi pada beberapa *website* kakas bantu perangkat lunak. Kakas bantu ini telah diuji melalui pengujian unit menggunakan metode *white box* dengan teknik *basis path testing*, pengujian integrasi dengan pendekatan *big-bang*, pengujian validasi menggunakan metode *black box*. Kasus uji yang dihasilkan oleh kakas bantu memiliki tingkat akurasi sebesar 100%.

Kata kunci: kakas bantu, pembangkitan kasus uji, *activity diagram*, DFT, DFS

Abstract

Software testing is one of the most important stages in software development. The complexity of system testing leads to the need for tools that can determine automatic test case cases. The activity diagram is one of the UML behavioral models that is suitable for system testing, because the activity diagram can evaluate the flow of the complete system. Therefore this study proposes to develop a tool that can generate test cases based on activity diagrams for model-based testing. The technique used in generating test cases is by building a flow dependency tree (DFT) that collects information from the ArgoUML file activity diagram through the help of a parser. Then the DFT processes using the depth first search algorithm (DFS) which is ready to be modified for search each path of the test case. All requirements for tool are the result of library needs elicitation in related research and observations on a number of software tool websites. This tool has tested through unit testing with the white box basis path testing technique, integration testing with big-bang approach. Validation testing with black box testing technique. Test cases generated by this tools have an accurate rate of 100%.

Keywords: tool, test case generation, activity diagram, DFT, DFS

1. PENDAHULUAN

Pengujian merupakan salah satu tahap yang paling krusial pada proses pengembangan perangkat lunak. Pengujian mempunyai peran yang sangat penting dalam menentukan keandalan dan kualitas dari sebuah perangkat lunak. Lebih dari separuh waktu dan biaya dari pengembangan perangkat lunak dihabiskan

untuk pengujian (Jena, Swain dan Mohapatra, 2014).

Pengujian adalah proses menguji perangkat lunak yang bertujuan untuk mencari sebuah kesalahan. Hal dasar dari proses menguji perangkat lunak adalah membuat sebuah kasus uji. Pembuatan kasus uji adalah tahap yang paling sulit dalam menguji perangkat lunak

(Jena, Swain dan Mohapatra, 2014). Kompleksitas yang terkait dengan pengujian sistem telah menyebabkan kebutuhan untuk pembuatan kasus uji otomatis. Hal ini disebabkan pengujian sistem dengan kebutuhan yang skalanya besar secara manual menjadi tugas yang rumit, melelahkan dan memakan waktu (Oluwagbemi & Asmuni, 2015). Pembangkitan kasus uji adalah dasar dari setiap pelaksanaan untuk menghasilkan kasus uji otomatis. Kasus uji yang dihasilkan dengan benar tidak hanya mendeteksi kesalahan dalam sistem perangkat lunak, tetapi juga meminimalkan biaya tinggi dan waktu yang terkait dengan pengujian perangkat lunak (Li dan Lam, 2004).

Pengembangan perangkat lunak yang menggunakan pendekatan berorientasi pada objek, memanfaatkan bahasa pemodelan *unified modeling language* (Kurniawan, 2018). *Unified modeling language* (UML) merupakan standar *de-facto* yang digunakan untuk menganalisis dan memodelkan kebutuhan pengguna atau dikenal sebagai artefak desain. Dengan UML, pengembang perangkat lunak dapat dengan mudah menganalisis dan memvisualisasikan berbagai rancangan dari suatu sistem. UML memiliki berbagai macam diagram yang dapat digunakan untuk menjelaskan sebuah sistem (OMG, 2017).

Model-based testing (MBT) merupakan pendekatan pengujian berdasarkan pada model dari rancangan yang telah dibuat (Li, et al., 2017). Pada beberapa kasus *model-based testing* menggunakan UML *behavioral model* sebagai dasar pembuatan rancangan dari kasus uji (Pressman, 2010). Salah satu *behavioral model* yang menggambarkan aktivitas pada sistem adalah *activity diagram*. *Activity diagram* cocok untuk pengujian sistem karena mempunyai kapabilitas mendeskripsikan perilaku sistem yang dikembangkan secara efektif (Oluwagbemi & Asmuni, 2015).

Pada penelitian sebelumnya oleh Oluwagbemi dan Asmuni pada tahun 2015 dengan judul “*Automatic Generation Of Test Cases From Activity diagrams For UML Based Testing (UBT)*” ditemukan teknik untuk menghasilkan kasus uji secara otomatis dari *activity diagram* dengan membangun sebuah *dependency flow tree* (DFT) yang menyimpan semua informasi dari *file model activity diagram* melalui bantuan sebuah *parser*. Kemudian teknik tersebut menerapkan sebuah algoritme untuk menghasilkan kasus uji dari DFT yang

telah dibangun. Kasus uji yang dibuat mempertimbangkan empat kriteria cakupan utama yaitu *all-paths*, basis *pair paths*, *conditions* and *branches coverage criteria*.






Berdasarkan permasalahan yang telah diuraikan, penulis akan melakukan penelitian dengan judul “Pengembangan Kakas Bantu Pembangkitan Kasus Uji pada *Model-Based Testing* Berdasarkan *Activity diagram*”. Dengan adanya kakas bantu ini, diharapkan para pengembang perangkat lunak dapat dengan lebih mudah untuk mengetahui kasus uji dalam proses pengujian.



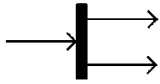
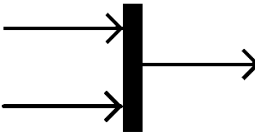
2. LANDASAN KEPUSTAKAAN

2.1 Activity Diagram

Activity diagram terdiri dari 9 elemen utama yaitu inisialisas (mulai), swimlanes, *activity*, *branch*, *guard*, *fork*, *join*, *merge* dan *end* (selesai). Semua elemen tersebut dapat diintegrasikan ke dalam *node* dan *edge*. Node mewakili proses *activity*, decision, swimlanes, *fork*, *merge*, *join* (Oluwagbemi dan Asmuni, 2015). Pada Tabel 1 dijelaskan mengenai fungsi dari beberapa notasi *activity diagram*.

Tabel 1. Notasi *Activity diagram*

Notasi	Deskripsi
<i>Start/ Initial Node</i> 	Digunakan untuk mewakili titik awal atau keadaan awal suatu kegiatan
<i>Activity</i> 	Digunakan untuk mewakili kegiatan proses
<i>Control Flow / Edge</i> 	Digunakan untuk merepresentasikan aliran kontrol dari satu aksi ke aksi lainnya
<i>Guard</i> [kondisi] 	Digunakan untuk menandai kondisi jalur setelah <i>decision</i> .
<i>Activity Final Node</i> 	Digunakan untuk menandai akhir dari semua aliran kontrol dalam aktivitas

<p><i>Decision Node</i></p> 	Digunakan untuk mewakili titik cabang bersyarat dengan satu <i>input</i> dan beberapa <i>output</i>
<p><i>Merge Node</i></p> 	Digunakan untuk mewakili penggabungan aliran. Ini memiliki beberapa <i>input</i> , tetapi satu <i>output</i> .
<p><i>Fork</i></p> 	Digunakan untuk mewakili aliran yang dapat bercabang menjadi dua atau lebih aliran paralel
<p><i>Join</i></p> 	Digunakan untuk mewakili aliran yang dapat bercabang menjadi dua atau lebih aliran paralel

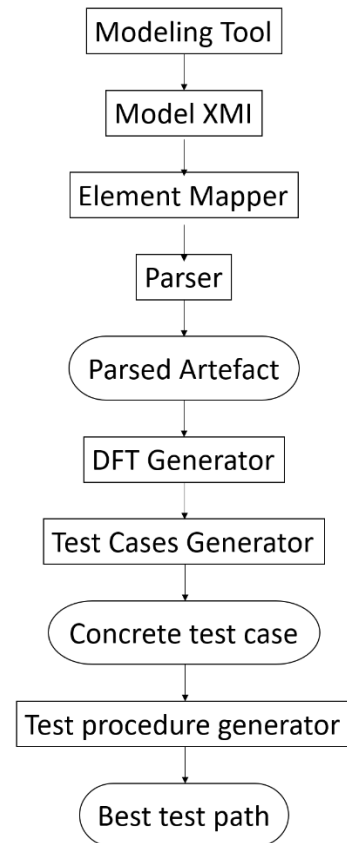
Dependency flow tree (DFT) adalah sebuah

Untuk menganalisa secara otomatis sebuah ekstraksi *activity diagram*, konsep *activity diagram* didefinisikan sebagai berikut (Oluwagbemi dan Asmuni, 2015) :

1. Sebuah *activity diagram* merupakan sebuah *tuple* $D = \{A, T, F, C, a_i, a_f\}$ dimana $A = \{a_1, a_2, \dots, a_n\}$ sebuah kumpulan dari *activity state*; $T = \{t_1, t_2, \dots, t_n\}$ merupakan sebuah kumpulan *transition state*; $C = \{c_1, c_2, \dots, c_n\}$ merupakan sebuah kondisi atau *guard*. F merupakan aliran hubungan antaran *activity* dan transisi. a_i merupakan *initial activity state*. Sementara a_f *final activity state*.
2. Sebuah kasus uji (KU) digenerasi dari *activity diagram* (AD), merupakan transversal dari urutan *activity* $\{a_1, a_2, \dots, a_n\}$, dimana a_i merupakan awal node dan a_n akhir node, sementara diantara a_i dan a_n merupakan *branch*, *fork*, *join*, *action*, *decision* atau *merge*.
3. Sebuah *curent state* (CS) untuk setiap transisi $*t$, $*t$ merepresentasikan *pre-set* dan *post-set* dari masing-masing t . Sebuah CS mewakili set transisi terkait dengan semua aliran keluar sebuah *edge*.
4. *Dependency flow tree* (DFT) adalah sebuah *graph* berarah yang berisi *node* dan *edge* dimana setiap *node* dan *edge* dalam grafik aktivitas memuat semua informasi yang telah diekstrak dari *file model activity diagram*. *Node* mewakili proses yang

meliputi keadaan *action*, *activity*, *decision*, *fork*, *join*, objek, dll. sedangkan *edge* mewakili urutan proses pada *activity diagram*.

2.2 Teknik Pembangkitan Kasus Uji



Gambar 1. Teknik pembangkitan kasus uji
Sumber: Oluwagbemi dan Asmuni (2015)

Teknik pembangkitan kasus uji diawali dengan membuat model *activity diagram* dari tool ArgoUML, kemudian model *activity diagram* disimpan dalam format XMI. Setelah itu *file XMI* diproses dengan *element mapper* untuk mengecek apakah format FileXMI sudah sesuai, kemudian *file XMI* akan diproses dengan sebuah Ekstraktor atau *Parser* yang akan menghasilkan sebuah *parsed artefact* dalam format *array*. Selanjutnya akan diproses DFT *Generator* yang akan menghasilkan sebuah struktur data *graph dependency flow tree* yang merepresentasikan sebuah *activity diagram*. Setelah itu akan diproses dengan *Test case generator* untuk menelusuri setiap *test case* pada DFT dan menghasilkan sebuah *concrete test case* yang berisi semua kemungkinan kasus uji dalam format variabel *array*. Selanjutnya diproses oleh *Test procedure generator* sehingga

menghasilkan sebuah *best test path*, yang merupakan jalur independen pada *activity diagram* yang ditelaah diproses.

2.3 Parser

Parser berurusan dengan proses penggalan artefak menjadi kasus uji konkret yang cocok untuk pelaksanaan tes. Untuk menghasilkan kasus uji komprehensif, penting untuk mengembangkan strategi ekstraksi artefak yang kuat yang mampu mendukung pembangunan artefak uji dari model UML untuk menghasilkan kasus uji pada *model-based testing*. *Parser* menggunakan serangkaian langkah korelasi selama proses ekstraksi untuk memastikan kesesuaian artefak dengan isi model UML dan elemen pada model. Hasil dari *parser* adalah sebuah variabel *array* yang akan diproses oleh DFT Generator (Oluwagbemi dan Asmuni, 2015).

2.4 DFT Generator

Generator *dependency flow tree* (DFT) yang bertujuan untuk menyimpan semua artefak yang diekstraksi sebelum dilakukan proses pembangkitan kasus uji. Elemen XMI berisi nama atribut dan variabel yang berguna untuk pembuatan kasus uji akan disimpan dalam DFT. DFT Generator bertanggung jawab untuk membangun DFT berdasarkan artefak yang diekstraksi. DFT dibangun berdasarkan jumlah *node* dan *edge* yang terkandung dalam *file XML*. Hasil dari DFT generator adalah sebuah struktur data *dependency flow graph* yang merepresentasikan dari suatu *activity diagram*.

2.5 Test Case Generator

Test Case generator bertanggung jawab untuk menelusuri setiap kasus uji dari sebuah *dependency flow tree*. Algoritme untuk pembangkitan kasus uji merupakan hasil modifikasi dari algoritma *depth first search* (Oluwagbemi dan Asmuni, 2015).

2.6 Algoritme Depth First Search (DFS)

Algoritme *Depth First Search* (DFS) adalah algoritme pencarian untuk menemukan *node* atau *simpul* yang belum dikunjungi dengan cara menelusuri *node* secara mendalam terlebih dahulu. Algoritme DFS menggunakan bantuan struktur data *Stack* yang berfungsi untuk menyimpan nilai dalam algoritme mencapai titik tertentu (Shaffer, 2009). Untuk *pseudocode*

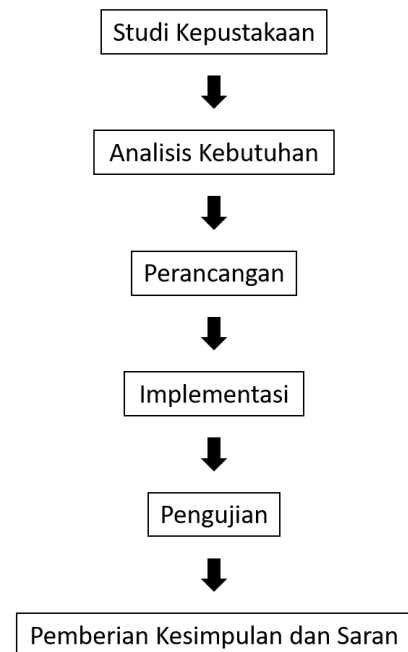
algoritme DFS bisa dilihat pada Gambar dibawah.

```
DFS(G,v) ( v is the vertex where the search starts )
Stack S := {}; ( start with an empty stack )
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
    u := pop S;
    if (not visited[u]) then
        visited[u] := true;
        for each unvisited neighbour w of u
            push S, w;
        end if
    end while
END DFS()
```

Gambar 2 Pseudocode algoritme DFS
Sumber: Heap (2002)

3. METODOLOGI PENELITIAN

Dalam bab metodologi penelitian dijelaskan terkait langkah - langkah yang hendak dilakukan dalam penelitian ini. Langkah – langkah dalam penelitian pengembangan kakas bantu ni, mengadopsi metode SDLC *waterfall model*. Hal ini dikarenakan penelitian ini merupakan implementasi dari penelitian sebelumnya. Kebutuhan sudah jelas dan jumlahnya tidak terlalu banyak. Sehingga tidak akan mengalami perubahan yang signifikan pada kebutuhan. Pada Gambar 3.1 menunjukkan alur pada metodologi penelitian ini..



Gambar 3. Diagram alir metode penelitian

Tahap pertama pada metodologi penelitian ini adalah studi kepustakaan. Pada tahap studi kepustakaan penulis mempelajari

kepastakaan yang menjadi penunjang penelitian yang dilakukan.

Tahap kedua adalah analisis kebutuhan. Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang harus ada pada sistem baik itu fungsional dan non fungsional. Pada tahap analisis kebutuhan penulis mengidentifikasi semua kebutuhan dan siapa saja pengguna yang terlibat dalam sistem. Elisitasi kebutuhan dilakukan dengan cara mengkaji penelitian yang sudah ada sebelumnya dan melakukan observasi pada *website* kakas bantu pengembangan perangkat lunak. Pada tahap ini penulis juga akan menjelaskan deskripsi sistem yang akan dibangun, kebutuhan sistem, karakteristik pengguna, serta melakukan pemodelan kebutuhan fungsional dengan *use case diagram* dan *use case scenario*. Sedangkan untuk menggambarkan aktifitas dan alur yang terjadi pada sistem menggunakan model *activity diagram*.

Tahap ketiga adalah perancangan. Setelah mendapatkan daftar kebutuhan fungsional dan non fungsional secara jelas tahap selanjutnya adalah perancangan. Perancangan yang dilakukan pada penelitian ini menggunakan pendekatan berorientasi objek. Pada pendekatan berorientasi objek sistem dimodelkan menggunakan pemodelan *unified modelling language* (UML). Pada tahapan perancangan ini dibuat beberapa diagram perancangan seperti *sequence diagram*, *class diagram*. Selain itu penulis juga melakukan perancangan antarmuka dan perancangan kode program. Selanjutnya adalah tahap ke empat yaitu implementasi, pada tahap ini penulis melakukan implementasi pada apa yang sudah dirancang sebelumnya. Untuk implementasi antarmuka menggunakan HTML, CSS dan Javascript, sedangkan untuk implementasi fungsi utama menggunakan bahasa PHP dengan *framework* Laravel versi 5.8.

Tahap kelima adalah pengujian. Tahap pengujian pada penelitian ini meliputi pengujian unit dengan menggunakan teknik *white box basis path testing*, pengujian integrasi dengan pendekatan *big-bang*, pengujian validasi kebutuhan fungsional menggunakan *use case testing*. Pengujian validasi pada dua kebutuhan non fungsional yaitu *compatibility* dan akurasi. Penghitungan akurasi kasus uji dilakukan dengan cara menguji kakas bantu dengan beberapa *activity diagram* kemudian mengecek jumlah jalur dan kebenaran dari jalur tersebut.

Tahap terakhir adalah pemberian kesimpulan dan saran. Pada tahapan terakhir ini peneliti menarik kesimpulan dari penelitian yang selesai dilakukan penarikan kesimpulan dilakukan setelah semua tahap sebelumnya telah dilakukan. Kesimpulan diambil guna memberikan jawaban terkait rumusan masalah yang telah ditentukan di awal penelitian. Di akhir penelitian ada pemberian saran untuk peneliti guna memperbaiki kekurangan yang ada pada penelitian ini dan sebagai pertimbangan untuk penelitian berikutnya.

4. ANALISIS KEBUTUHAN

Kebutuhan sistem didapatkan dari penjelasan *paper Automatic Generation Of Test Cases From Activity diagrams For UML Based Testing (UBT)* oleh oluwagbemi dan asmuni pada tahun 2015. Pada penelitian tersebut ditemukan kerangka kerja untuk pembangkitan kasus uji dari *activity diagram*. *Activity diagram* dibuat menggunakan kakas bantu ArgoUML (Oluwagbemi dan Asmuni, 2015).

Teknik yang digunakan untuk pembangkitan kasus uji pada penelitian tersebut dengan membangun sebuah *graph activity flow tree* (AFT) yang menyimpan semua informasi pada *activity diagram* dengan bantuan sebuah *parser*. Kemudian *activity flow tree* (AFT) diproses dengan algoritme *depth first search* (DFS) yang telah dimodifikasi sehingga dapat menghasilkan kasus uji. Selain itu terdapat contoh *prototype* yang dapat memudahkan dalam membantu menelusuri kebutuhan. Kemudian untuk kebutuhan lainnya diperoleh dengan melakukan observasi pada kakas bantu pengembangan perangkat lunak yang berbasis *website* seperti draw.io dan creately.com.

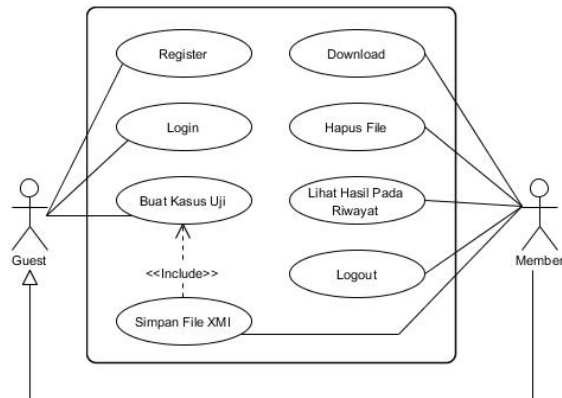
Dari elisitasi kebutuhan tersebut diperoleh 7 kebutuhan fungsional dan 1 kebutuhan non fungsional. Daftar kebutuhan fungsional pada kakas bantu ini tertera pada Tabel 2.

Tabel 2. Kebutuhan Fungsional

No.	Nama Fungsi	Deskripsi
1	Buat kasus uji	Kakas bantu harus dapat melakukan <i>parsing file</i> XMI yang diunggah <i>guest</i> atau member kemudian membuat DFT, menghasilkan kasus uji dan menampilkannya.

2	Lihat hasil	Kakas bantu harus dapat menampilkan hasil kasus uji dari histori unggah <i>member</i> .
3	Register	Kakas bantu dapat melayani proses pendaftaran Pengguna
4	Login	Kakas bantu dapat melakukan proses autentikasi Pengguna
5	Log out	Kakas bantu dapat melayani proses <i>member</i> keluar dari sistem
6	Download file	Kakas bantu dapat melayani proses unduh <i>file</i> XMI yang telah diunggah oleh <i>member</i> .
7	Hapus file	Kakas bantu dapat melayani proses hapus <i>file</i> XMI yang telah diunggah oleh <i>member</i> .
8	Simpan File XMI	Kakas bantu dapat menyimpan file XMI yang diunggah <i>member</i>

Pada Gambar 3 di bawah merupakan *use case diagram* yang memodelkan relasi antar aktor dengan kebutuhan fungsional, aktor dengan aktor lain dan kebutuhan fungsional dengan kebutuhan lain.



Gambar 4. Use case diagram

Untuk kebutuhan non fungsional pada kakas bantu yang dikembangkan dapat dilihat pada Tabel 3 di bawah.

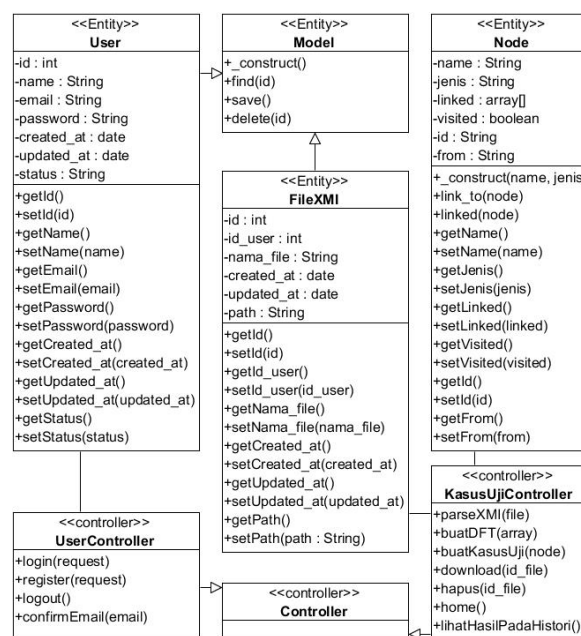
Tabel 3. Kebutuhan Non Fungsional

No	Parameter	Deskripsi
1	Compatibility	Fungsional pada kakas bantu pembangunan kasus uji berdasarkan <i>activity diagram</i> dapat dijalankan pada browser google chrome, mozilla firefox dan opera

2	Akurasi	Kasus uji yang dihasilkan kakas bantu harus memiliki tingkat akurasi sebesar 100%
---	---------	---

5. PERANCANGAN DAN IMPLENTASI

Pada bagian perancangan melakukan beberapa hal yaitu perancangan arsitektur *sequence diagram* dan *class diagram*. Selain itu pada bagian ini juga terdapat perancangan algoritme dan perancangan antarmuka pengguna. Hasil perancangan arsitektur *classs diagram* bisa dilihat pada Gambar 5 di bawah. Kelas Model dan Controller merupakan kelas bawaan dari kerangka kerja Laravel yang memiliki banyak fungsi yang tidak penulis cantumkan.



Gambar 5. Class diagram

Untuk perancangan Algoritme penulis akan menampilkan 3 algoritme utama dalam pembangunan kasus uji yaitu *parsing* XMI pada Gambar 6, buat DFT pada Gambar 7 dan buat DFT pada Gambar 8. Rancangan algoritme dibuat dalam bentuk *pseudocode*.

```

START
READ file xmi yang diupload
IF file xmi tidak sesuai format THEN
  SHOW pesan peringatan
  Break;
END IF
IF status pengguna adalah Member
  Save file XMI
END IF
FOR setiap state pada activity diagram
  ADD state to variabel array state
END FOR
FOR setiap state pada transition diagram
  ADD transition variabel array
  transition
END FOR
CALL FUNCTION DFT
END

```

Gambar 6. *Pseudocode parsing XMI*

```

INIT object node, variabel i;
FOR setiap variabel array state
  Add state to object node
  SET id node to i
  INCREMENT i;
END FOR
FOR setiap variabel array state
  IF cabang array state > 1 THEN
    FOR each node cabang
      SET node link to
      cabang node
    END for
  ELSE
    SET node link to cabang node
  END IF
END FOR
CALL function BuatKasusUji

```

Gambar 7. *Pseudocode buat DFT*

```

INIT elementStack[], decisionStack[], resultStack[],
result[], bestPath[], popElement
ADD node root to elementStack[]
WHILE elementStack[] tidak kosong
  CALL function pop of elementStack[]
  SET popElement to pop of elementStack[]
  Add popElement to result
  Add result to resultStack[]
  IF popElement belum dikunjungi
    FOR each cabang node
      CALL function push cabang to
      elementStack[]
    END FOR
  ELSE
    FOR each cabang node
      IF cabang node tidak ada pada result
        CALL function push cabang to
        elementStack[]
      END IF
    END FOR
  END IF
  IF popElement bercabang dan belum dikunjungi
    SET visited true
    CALL function push popElement to
    decisionStack[]
  END IF
  IF popElement tidak memiliki cabang
    Add result to bestPath
    WHILE end result = end decision stack
      CALL function pop result
    END WHILE
  END IF
END WHILE
SHOW halaman hasil

```

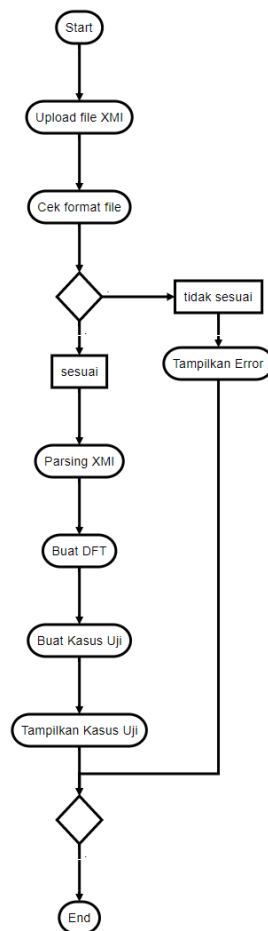
Gambar 8. *Pseudocode buat kasus uji*

Sedangkan untuk hasil implementasi bisa dilihat pada Gambar 9 dan Gambar 10 berikut

Gambar 9. Halaman utama *website kakas bantu*

Pembangkitan Kasus Uji Model-Based Testing Activity Diagram

Sebuah kakas bantu yang dapat menghasilkan kasus uji dari model activity diagram secara otomatis,



Jalur Independen

No	Jalur
1	Start → Upload file XMI → Cek format file → tidak sesuai → Tampilkan Error → End
2	Start → Upload file XMI → Cek format file → sesuai → Parsing XMI → Buat DFT → Buat Kasus Uji → Tampilkan Kasus Uji → End

Kasus Uji

No Jalur	Input Cek format file
1	tidak sesuai ,
2	sesuai ,

Gambar 10 Halaman hasil kasus uji

6. PENGUJIAN

Pada bagian ini penulis akan membahas mengenai proses pengujian terhadap aplikasi perangkat lunak yang telah dikembangkan. Terdapat 4 macam pengujian pada kakas bantu yaitu pengujian unit, pengujian integrasi dan

pengujian validasi baik pada kebutuhan fungsional maupun non fungsional.

Pada pengujian unit, penulis menggunakan teknik *white box basis path testing*. Pertama dengan melakukan pengecekan terhadap struktur kode sumber, kemudian membuat *flowgraph* untuk menentukan *cyclomatic complexity* dan

jalur independennya. Setelah itu menguji setiap jalur independen yang telah ditemukan. Pengujian unit yang dilakukan pada 3 fungsi yaitu parsing XMI, buat DFT dan buat kasus uji, menghasilkan nilai persentase valid sebesar 100%. Selanjutnya dilakukan pengujian integrasi dengan pendekatan *big-bang* menghasilkan 100% valid pada seluruh kasus uji.

Kemudian untuk pengujian validasi terdapat 2 jenis pengujian yaitu pengujian fungsional dan non fungsional. Pada pengujian validasi fungsional penulis menggunakan teknik *use case testing*. Dengan teknik ini penulis menentukan kasus uji berdasarkan *use case scenario*. Pengujian validasi dilakukan pada semua kebutuhan fungsional dengan hasil persentase valid 100%.

Pengujian validasi non fungsional *compatibility* dilakukan dengan cara menjalankan semua *website* kakas bantu pada beberapa *browser* yaitu google chrome, firefox dan opera. Hasil pengujian *compatibility* ditunjukkan pada Tabel 4 di bawah.

Tabel 4 Pengujian *compatibility*

Fungsi	Expected Result	Chrome	Firefox	Opera
Buat kasus uji	Fungsional dapat berjalan dengan lancar	Passed	Passed	Passed
Lihat hasil		Passed	Passed	Passed
Register		Passed	Passed	Passed
Login		Passed	Passed	Passed
Log out		Passed	Passed	Passed
Download file		Passed	Passed	Passed
Hapus file		Passed	Passed	Passed

Berdasarkan hasil pengujian *compatibility* pada sistem mendapat hasil 100% valid, sehingga sistem sudah memenuhi kebutuhan non fungsional.

Pengujian yang terakhir adalah pengujian validasi pada kebutuhan fungsional akurasi. Pengujian akurasi dimaksudkan untuk mengetahui tingkat akurasi kasus uji yang dihasilkan oleh kakas bantu. Pengujian dilakukan dengan cara mengecek jumlah jalur dan mengecek kebenaran jalur tersebut. Penulis menggunakan 3 data uji yaitu *activity diagram* sistem ATM, *activity diagram* kakas bantu pembangkitan kasus uji berdasarkan *activity diagram* (KUAD) dan *activity diagram scan*

assets. Hasil pengujian akurasi ditunjukkan pada Tabel 5 berikut.

Tabel 5. Pengujian Akurasi

Data Uji	Jumlah jalur berdasarkan jumlah region	Jumlah jalur dari kakas bantu	Tingkat Akurasi Kebenaran Jalur
ATM	4	4	100 %
KUAD	2	2	100 %
Scan Asset	2	2	100 %
Rata – rata akurasi kasus uji			100 %

Hasil pengujian akurasi menghasilkan rata - rata persentase 100 %. Hal tersebut menunjukkan bahwa kakas bantu yang dikembangkan penulis memiliki akurasi yang tinggi.

7. KESIMPULAN DAN SARAN

Kesimpulan pada penelitian pengembangan kakas bantu pembangkitan kasus uji berdasarkan *activity diagram* didefinisikan sebagai berikut :

1. Analisis kebutuhan dilakukan dengan mengkaji kepustakaan terkait pembangkitan kasus uji berdasarkan *activity diagram* dan melakukan observasi pada *website* kakas bantu pengembangan perangkat lunak yang sudah ada sebelumnya. Dari analisis kebutuhan menghasilkan 8 kebutuhan fungsionalitas dan 2 kebutuhan non fungsionalitas.
2. Perancangan pada penelitian ini menggunakan pendekatan berorientasi pada objek. Pada tahap ini dilakukan perancangan sistem digambarkan dengan *activity diagram*, perancangan arsitektur menggunakan *sequence diagram* dan *class diagram*, perancangan algoritme dan perancangan antarmuka pengguna yang digambarkan melalui *mock up*. Pada tahap implementasi berhasil menerapkan rancangan yang dibuat pada tahap sebelumnya. Dalam implementasi kasus uji berdasarkan *activity diagram* menggunakan bahasa pemrograman PHP dan *framework* Laravel versi 5.8.
3. Terdapat 3 tahapan dalam menguji kakas bantu, yaitu pengujian unit, pengujian integrasi dan pengujian validasi. Pengujian unit dengan menggunakan teknik *white box basis path testing* berhasil menguji semua

jalur dasar, pengujian integrasi dengan pendekatan *big-bang* dan pengujian validasi dilakukan dengan berhasil memvalidasi seluruh kebutuhan fungsional yang berjumlah 8 buah dan 2 buah kebutuhan fungsional.

Adapun berbagai saran untuk penelitian selanjutnya didefinisikan sebagai berikut :

Kakas bantu yang dikembangkan penulis saat ini hanya mampu melakukan proses pembangkitan kasus uji pada *file xmi activity diagram* dari tool ArgoUML saja. Untuk meningkatkan tingkat kefleksibelan dari kakas bantu penulis menyarankan pada penelitian selanjutnya untuk melakukan penyesuaian kakas bantu terhadap *file activity diagram* dari tools lainya atau mungkin dapat juga dengan menambahkan fungsi untuk perancangan *activity diagram* sendiri pada *website* kakas bantu.

REFERENSI

- Heap, D., 2002. *Depth First Search (DFS)*. [daring] Tersedia pada: <<http://www.cs.toronto.edu/~heap/270F02/node36.html>> [Diakses 29 Jan 2019].
- Jena, A.K., Swain, S.K. dan Mohapatra, D.P., 2014. A novel approach for test case generation from UML activity diagram. In: *Proceedings of the 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques, ICICT 2014*.
- Kurniawan, T.A., 2018. Pemodelan Use Case (UML): Evaluasi Terhadap beberapa Kesalahan dalam Praktik. *Jurnal Teknologi Informasi dan Ilmu Komputer*, 5(1), hal.77–86.
- Li, H. dan Lam, C.P., 2004. Software Test Data Generation using Ant Colony Optimization. hal.1–4.
- Oluwagbemi, O. dan Asmuni, H., 2015. Automatic Generation of Test Cases From Activity diagrams for UML Based Testing (UBT). 73(13), hal.37–48.
- Pressman, R.S., 2010. *Software Engineering A Practitioner's Approach*. 7th ed. *Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman*. New York: The McGraw-Hill.
- Shaffer, C. a, 2009. *A Practical Introduction to Data Structures and Algorithm Analysis Third Edition (Java)*. Building. Blacksburg: Prentice Hall.