PENGUJIAN PERANGKAT LUNAK DENGAN MENGGUNAKAN BEHAVIOR UML (STUDI KASUS : APLIKASI DARING TERBUKA TERPADU UB)

Drajad Muhammadi Latief[1], Aditya Rachmadi, S.T, M.TI[2], Denny Sagita Rusdianto S.Kom., M.Kom[3]
Fakultas Ilmu Komputer, Universitas Brawijaya Malang
Email : drajadmuhammadilatief@gmail.com[1], rachmadi.aditya@gmail.com[2], denny.sagita@ub.ac.id[3]

ABSTRAK

Penelitian ini bertujuan untuk melakukan pengujian aplikasi PDTT – UB menggunakan metode Behavior UML yang nantinya aplikasi tersebut akan digunakan sebagai media pembelajaran di Universitas Brawijaya Malang. Untuk mendapatkan *test case*, metode yang digunakan yaitu behavior UML, dimana UML yang digunakan adalah *Activity diagram* dan *Sequence diagram*. Dari diagram UML tersebut akan dibuat Model Flow Graph (MFG) yang didapat dari hasil analisis *Activity diagram* dan *Sequence diagram*. Setelah didapatkan MFG dari *Activity diagram* dan *Sequence diagram*, maka MFG akan dibuat jalur-jalur yang akan di analisis menjadi *test case* menggunakan metode *Test case* Generation untuk pengujian Aplikasi PDTT-UB.

Dengan menggunakan behavior UML, akan muncul beberapa test case yang tidak bisa didapat apabila menggunakan metode pengujian yang lain seperti blackbox testing karena pengujian Behavior UML sumber datanya adalah fase perancangan yaitu Activity diagram dan Sequence diagram sehingga test case yang didapatkan akan sesuai dengan fase perencanaan dan kebutuhan aplikasi. Dengan pengujian behavior UML ini, maka rekomendasi yang diberikan kepada developer aplikasi PDTT-UB tidak keluar dari kebutuhan aplikasi yang dibuat di fase perancangan.
Kata Kunci: Behavior UML, PDTT – UB, Model Flow Graph, Test case Generation

ABSTRACT

This research point to test PDTT-UB application using behavior UML that the application will be used as a method of learning in Brawijaya University. To get a *test case*, the method used is behavior UML which is used *Activity diagram* and *Sequence diagram*. From the UML Diagram will be made Model Flow Graph (MFG) is obtained from the analysis of the Activity diagram and Sequence diagram. Having obtained MFG of Activity diagram and Sequence diagram, the MFG will be made pathways to be a *test case* using *Test case* Generation methods for PDTT-UB.

By using Behavior UML, there will be some *test case*s that can't be obtained when using others methods of testing such as blackbox testing because in behavior UML, the data source will get from design phase, namely Activity diagram and Sequence diagram. So that *test case*s will be obtained in accordance with the planning phase and application needs. By testing Behahavior UML, then the results and recommendations goven to the developer of PDTT-UB doesn't come out the needs of applications made in design phase
Key words: Behavior UML, PDTT – UB, Model Flow Graph, *Test case* Generation

1. PENDAHULUAN

Pada era globalisasi saat ini perkembangan teknologi informasi sangat pesat. Dengan perkembangan tersebut semakin banyak pekerjaan yang menjadi praktis dengan dukungan software-software yang canggih. Semakin hari, kebutuhan akan software yang berkualitas juga semakin dibutuhkan. Untuk membuat software yang berkualitas salah satu cara yang di gunakan adalah memaksimalkan salah satu fase yang ada dalam *Software Development Lifecycle* (SDLC) yaitu fase Testing. SDLC adalah pendekatan bertahap untuk melakukan analisa dan membangun rancangan sistem dengan menggunkan siklus yang spesifik terhadap kegiatan pengguna. Software testing atau jika di artikan ke dalam Bahasa Indonesia adalah Pengujian Software merupakan hal yang penting untuk menentukan kualitas dari perangkat lunak. Dengan melakukan Sofware Testing maka bug atau error pada software dapat segera ditemukan dan diperbaiki.

Universitas Brawijaya merupakan salah satu institusi perguruan tinggi di Indonesia yang memberikan layanan kepada masyarakat dengan menyiapkan sumber daya manusia yang berkualitas baik. Untuk memperlancar proses belajar mengajar di UB terutama metode pembelajaran jarak jauh, maka bagian TIK dari Fakultas Ilmu Komputer membuat aplikasi Pembelajaran Daring Terbuka Terpadu (PDTT-UB) . Dengan adanya aplikasi ini, maka metode pembelajaran jarak jauh yang di usung Universitas Brawijaya akan dapat berjalan dengan baik.

Aplikasi Pembelajaran Jarak Jauh ini merupakan software baru di kalangan UB. Aplikasi ini kemungkinan masih ada fault, failure dan error didalamnya. Untuk meminimalisir error, fault dan failure tersebut maka perlu dilakukan software testing kepada Aplikasi Pembelajaran Jarak Jauh ini. Tujuan dari Testing sendiri adalah memastikan bahwa apa yang direncanakan di awal atau pada requirement dapat sesuai dengan hasil yang telah di

buat dalam bentuk aplikasi, proses-proses yang diinginkan dapat berjalan dengan baik sesuai *requirement* serta meminimalisir *error, fault* dan *failure* dalam aplikasi PDTT terutama pada modul administrator. Dalam modul ini perlu di uji dengan tepat karena merupakan modul vital yang menjadi sumber pada aplikasi PDTT. Banyak metode yang dapat di gunakan untuk melakukan pengujian pada Aplikasi PDTT ini seperti metode blackbox testing, whitebox testing, dan greybox testing.

Salah satu bagian dalam software testing yang pasti di pakai dalam metode pengujian apapun adalah *Test case*. "*Test case* adalah sekumpulan dokumen dari input test, kondisi yang akan dieksekusi, dan hasil yang diharapkan" (Galin, 2004). Membuat *Test case* yang baik sangat penting agar kelancaran saat pengujian dapat berjalan baik. Dalam pengujian Aplikasi PDTT ini belum terdapat *Test case* yang jelas sehingga perlu adanya *Test case* untuk pengujian Aplikasi PDTT. Membuat *test case* tidak mudah, karena perlu ketepatan dengan tujuan dibuatnya aplikasi PDTT. "*Test case* mungkin tidak layak untuk memverifikasi kebenaran untuk setiap kasus pada regresi yang besar" (McCabe, 1996). Namun setidaknya dengan *Test case* dapat meminimalisir error, fault dan failure yang ada, Semakin tepat *Test case* semakin tinggi pula kualitas Aplikasi PDTT ini. Salah satu cara pembuatan *test case* adalah mengacu pada Behavior UML Aplikasi PDTT. Dengan mengacu pada Behavior UML, *test case* yang dihasilkan tepat dan pengujian pada aplikasi PDTT dapat berjalan dengan baik sehingga kualitas software akan tinggi.

Kesulitan menentukan *Test case* yang tepat dapat diatasi dengan menggunakan metode behavior UML yang kelebihannya yang pertama adalah biaya design *test case* lebih kecil daripada biaya design *test case* manual seperti white box testing yang memiliki biaya design *test case* yang tinggi dan yang kedua adalah reliability dari test coverage tinggi (Swain & Mohapatra, 2010). Oleh karena itu diambil judul penelitian "Pengujian Perangkat Lunak Dengan Menggunakan Behavior UML (Studi Kasus : Aplikasi Pembelajaran Daring Terbuka Terpadu UB)".

2. LANDASAN KEPUSTAKAAN

2.1 APLIKASI PEMBELAJARAN DARING TERBUKA TERPADU UB (PDTT-UB)

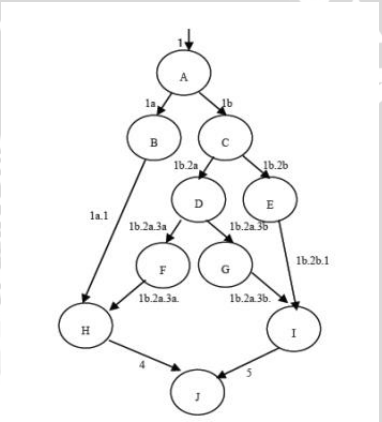
Aplikasi PDTT- UB adalah aplikasi yang digunakan untuk pembelajaran jarak jauh antara dosen dan mahasiswa untuk menambah model pembelajaran yang ada di Universitas Brawijaya. Awalnya aplikasi ini bernama PJJ (pembelajaran jarak jauh) yang awalnya akan digunakan hanya di Fakultas Ilmu Komputer (FILKOM), tetapi seiring berjalannya waktu seluruh fakultas di Universitas Brawijaya juga membutuhkan aplikasi tersebut sehingga muncul lah aplikasi PJJ yang skalanya adalah seluruh UB yang dinaman Aplikasi Pembelajaran Daring Terbuka Terpadu (PDTT) Universitas Brawijaya.

2.2 DEFINISI PENGUJIAN

Pengujian adalah suatu proses mengevaluasi sistem atau komponen dengan maksud untuk menemukan *error, fault, failure* sehingga aplikasi memenuhi persyaratan yang ditentukan atau tidak. Menurut ANSI / IEEE 1059 Standart, Pengujian dapat didefinisikan sebagai "suatu proses menganalisis item perangkat lunak untuk mendeteksi perbedaan antara kondisi yang ada dan diperlukan (cacat / kesalahan / bug) dan mengevaluasi fitur item perangkat lunak".

2.3 DEFINISI FLOW GRAPH

Flow Graph adalah gabungan dari beberapa titik (*node*s) dan garis (edges). Flow graph digunakan untuk menggambarkan alur dari suatu alogirtma atau proses UML seperti *activity diagram, sequence diagram* maupun UML yang lain. *Node* akan merepresentasikan keseluruhan pernyataan atau potongan proses, sedangkan edge merepresentasikan aliran kontrol..



Gambar 1 Flow Graph
(Swain & Mohapatra, 2010)

2.4 PENGUJIAN SOFTWARE DENGAN BEHAVIOR UML

Pengujian *software* dengan behavior UML adalah pengujian yang dilakukan berdasarkan UML seperti *activity diagram, sequence diagram*, class diagram, state machine diagram maupun diagram yang menggambarkan aplikasi tersebut. Dari Diagram-diagram UML akan dianalisis sehingga menghasilkan beberapa point yang akan menjadi *Test case*. Pengujian Behavior UML termasuk grey-box testing yang merupakan perpaduan dari black-box testing dengan white-box testing. Ada beberapa UML yang digunakan dalam behavior UML ini, diantaranya adalah *activity diagram* dan *sequence diagram*. "*Sequence diagram* merepresentasikan dengan lengkap pesan-pesan yang dikirim menuju objek" (Swain & Mohapatra, 2010). Ketika pesan dikirimkan menuju objek, objek akan melakukan operasi sesuai apa yang diminta. Setelah pesan diterima, operasi yang diminta akan di eksekusi. Dengan kata lain, sequence yang merupakan white-

box di dalam pengujian behavior UML. Sedangkan *activity diagram* mendeskripsikan alur yang terjadi dari masing-masing activity. "*Activity diagram* menekankan suatu aktivitas yang dilakukan object atau kumpulan object sehingga ini akan menjadikan *activity diagram* menjadi diagram yang tepat untuk merepresentasikan alur dari suatu aplikasi" (Swain & Mohapatra, 2010). Selain itu *activity diagram* juga menjelaskan *coverage criterion* yang digunakan untuk melengkapi test scenario. Dengan demikian *activity diagram* tidak menggambarkan alur yang ada di dalam sistem, melainkan hanya proses yang terjadi di luar sehingga *activity diagram* yang merupakan black-box di dalam pengujian behavior UML.

Proses behavior UML ada 3 (Swain & Mohapatra, 2010), yaitu :

a. Generate MFG dari *sequence diagram* dan *activity diagram*

MFG dibuat dengan menulusuri alur *activity diagram* dan *sequence diagram* dari awal sampai akhir, yang didalamnya ada kondisi, method, eksekusi, dan perulangan. Dari proses yang ada di dalam *activity diagram* dan *sequence diagram* tesebut di kumpulkan dan diberi label. Kumpulan proses yang ada di *activity diagram* adalah *Method activity table (MAT)* dan untuk *sequence diagram* adalah *Object Method Association Table (OMAT)*. Setiap proses di MAT dan OMAT menjadi *node* dan setiap *node* akan saling dihubungkan sesuai alur dari diagram oleh edge. Contoh dari MAT dapat dilihat pada gambar 2 dan untuk contoh dari OMAT dapat dilihat pada gambar 3

TABLE 1: Object Method Association Table (OMAT) for the Fig. 1

SYMBOL	OBJECT-METHOD ASSOCIATION
A	atm:Validate Amount
B	cstact: Withdraw
C	atm: Display Message
D	cstact: Return
E	atm: Dispense Cash
F	atm: Print Receipt
G	atm: Return

Gambar 2 Gambar Object Method Association Table (OMAT)
Sumber : (Swain & Mohapatra, 2010)

TABLE 2: Method Activity Table (MAT) for the Fig, 2

SYMBOL	ACTIVITY
A	Amount < Balance
B	Update Balance
C	Check for Overdraft
D	Check WithDraw Limit
E	Does not have Permission for Overdraft
F	With in Limit
G	Beyond Limit
H	Return True
I	Return False
J	Return

Gambar 3 Gambar Method activity table (MAT)
Sumber : (Swain & Mohapatra, 2010)

b. Generate *test sequence* dari MFG sesuai dengan *sequence diagram* dan *activity diagram*

Proses yang ada di MFG dijalankan sesuai dengan kondisi yang didalamnya ada *pre-condition*, *main scenario* dan *post-condition* fitur tersebut yang akan menghasilkan skenario-skenario jalur yang didalam jalur tersebut terdapat *edge*, *method* / *activity* dan *category partition* yang dinamakan *test sequence*

c. Generate *test case* dari test sequence sesuai jalur uji

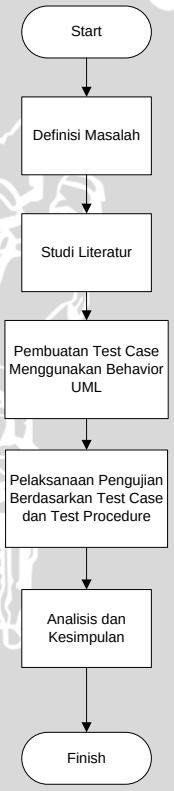
Untuk mendapatkan *test case*, yang pertama dilakukan adalah mencari jalur yang mungkin dalam MFG *sequence diagram* dan *activity diagram*. Dari setiap jalur di kombinasikan dengan test sequence sehingga muncul jalur yang sesuai dengan test sequence, jalur yang sesuai dengan test sequence tersebut yang menjadi *test case* yang dinamakan *test case generation*

3. METODOLOGI

3.1 JENIS PENELITIAN

Penelitian ini menggunakan metodologi implementatif pengembangan lanjut yang menggunakan metode pengembangan atau alogoritma yang tepat dan telah dilakukan kajian pustaka pada penelitian atau literatur terdahulu.

3.2 ALUR PENELITIAN



Gambar 4 Alur Penelitian

3.3 DEFINISI MASALAH

Pada tahap ini yang dilakukan adalah berdasarkan dari pengalaman tentang kesulitan menentukan *Test case* yang tepat untuk pengujian aplikasi yang ada di Universitas Brawijaya, dilakukan pencarian metode-metode lain untuk menentukan *Test case* yang tepat. Pengumpulan literatur serta beberapa contoh *Test case* di lakukan untuk melakukan pengujian terhadap aplikasi PDTT UB modul administrator.

Selama ini proses pembuatan *Test case* hanya dilakukan berdasarkan fitur-fitur yang ada sehingga terkesan ada beberapa hal yang tidak ter*cover* dari *Test case* tersebut. Pengujian pun tidak dapat dilakukan secara maksimal. Berawal dari hal tersebut, maka pengujian aplikasi PDTT UB akan di uji dengan menggunakan *Test case* yang di dapatkan dari metode behavior UML. Dari permasalahan tersebut akan di jadikan sebagai perumusan masalah oleh peneliti dalam pembuatan tugas akhir ini.

3.4 STUDI LITERATUR

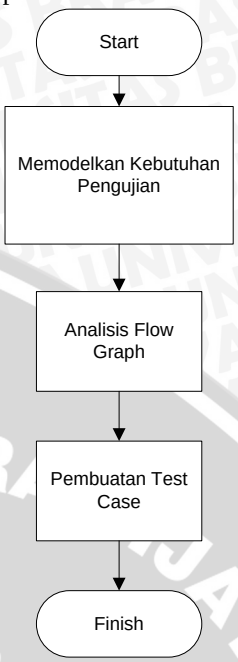
Melakukan Studi Literatur atau kajian pustaka yang berkaitan dengan Pengujian *Software* terutama metode behavior UML, UML secara umum dan secara khusus *Activity diagram, Sequence diagram* , (UML yang dipakai) sebagai dasar pembuatan *Test case*. Selain itu juga mempelajari Aplikasi Pembelajaran Daring Terbuka Terpadu (PDTT) UB modul administrator yang merupakan objek dari penelitian ini. Untuk paper yang lebih spesifik, dalam penelitian ini mengacu pada penelitian yang berjudul " *Test case Generation* from Behavioral UML Models" yang ditulis oleh Santosh Kumair Swain dan Durga Prasad Mohapatra pada September 2010.

Setelah melakukan studi literature, akan muncul apa saja yang diperlukan untuk melakukan penelitian ini yaitu objek yang diteliti dalam hal ini aplikasi PDTT-UB, kemudian modul yang diuji adalah modul administrator Karena menurut developer PDTT-UB ini adalah modul yang paling pokok dalam aplikasi ini. Setelah muncul objek dari penelitian, yang dilakukan adalah meminta sumber data dalam hal ini seharusnya adalah *activity diagram* dan *sequence diagram* dari fitur-fitur yang ada di aplikasi PDTT-UB. Namun, pada saat pengumpulan data, developer memberi tahu bahwa aplikasi PDTT-UB ini belum ada dokumennya Karena aplikasi diminta cepat jadi sehingga tidak ada dokumennya. Maka dari itu hal pertama yang dilakukan adalah membuat dahulu *Activity diagram* dan *Sequence diagram* dari masing-masing fitur yang ada di modul administrator aplikasi PDTT-UB.

3.5 PERBAIKAN MODEL

Dalam pembuatan *Test case* menggunakan behavior UML ini ada beberapa tahap yang harus dilakukan untuk mendapatkan *Test case*. Proses ini mengacu pada penelitian yang berjudul "*Test case Generation* from Behavioral UML Models" yang ditulis oleh Santosh Kumar Swain dan Durga Prasad Mohapatra. untuk lebih lebih jelasnya dapat dilihat pada gambar 5

Berdasarkan gambar 5, proses pertama adalah memodelkan kebutuhan pengujian. Untuk memenuhi kebutuhan pengujian dalam hal ini adalah *activity diagram* dan *sequence diagram* dari aplikasi PDTT-UB adalah observasi. Cara yang di gunakan untuk observasi adalah meminta source atau code dari aplikasi PDTT-UB kepada developer untuk di demo kan. Setelah melakukan demo aplikasi, proses selanjutnya adalah membuat *activity diagram* dam *sequence diagram* berdasarkan demo aplikasi dan source code dari aplikasi PDTT-UB tersebut.



Gambar 5 Alur Proses Metode Behavior UML

Activity diagram digunakan karena *activity diagram* menekankan suatu aktivitas yang dilakukan objek atau kumpulan objek sehingga ini akan menjadikan *activity diagram* menjadi diagram yang tepat untuk merepresentasikan alur dari suatu aplikasi (Swain & Mohapatra, 2010). *Sequence diagram* digunakan karena *sequence diagram* merepresentasikan dengan lengkap pesan-pesan yang dikirim menuju objek (Swain & Mohapatra, 2010).

Fase kedua dari gambar 3.1 adalah analisis flow graph. Setelah *Activity diagram* dan *sequence diagram* dibuat, selanjutnya adalah Generate MFG dari *sequence diagram* dan *activity diagram* dari setiap fitur yang hasilnya adalah MFG dari setiap fitur yang ada pada PDTT-UB modul administrator.

Fase ketiga dari gambar 5 adalah pembuatan *test case*. Pada fase yang pertama dilakukan adalah menggenerate MFG menjadi *test sequence*. Untuk mendapatkan *test sequence* dibutuhkan kondisi dari fitur yang akan di buat *test case*, Karena belum ada dokumen dari PDTT UB yang berisi kondisi dari masing-masing fitur, maka perlu dibuat dulu kondisi dari masing-masing fitur berdasarkan proses demo aplikasi PDTT UB. Setelah itu baru dilakukan generate MFG menjadi *test sequence*. Setelah itu proses selanjutnya adalah genereate MFG menjadi *Test case* menggunakan algoritma *test case generation* sehingga muncul *test case* dari masing-masing fitur. Setelah *test case* dari masing-masing fitur terbuat, selajutnya di tabelkan menggunakan tabel 1 untuk mempermudah pembacaan dan mempermudah proses pengujian

Tabel 1 Template Tabel *Test case* dan *Test Procedure*

No	Fitur	*Test case* ID	Input

3.6 PELAKSANAAN PENGUJIAN BERDASARKAN *TEST CASE* DAN TEST PROCEDURE

Fase ini adalah fase pengujian aplikasi. Pengujian dilakukan langsung ke aplikasi PDTT-UB sesuai dengan *Test case* dan *test procedure* yang ada. Kemudian hasil pengujian akan ditabelkan untuk mempermudah analisis selanjutnya.

3.7 ANALISIS DAN KESIMPULAN

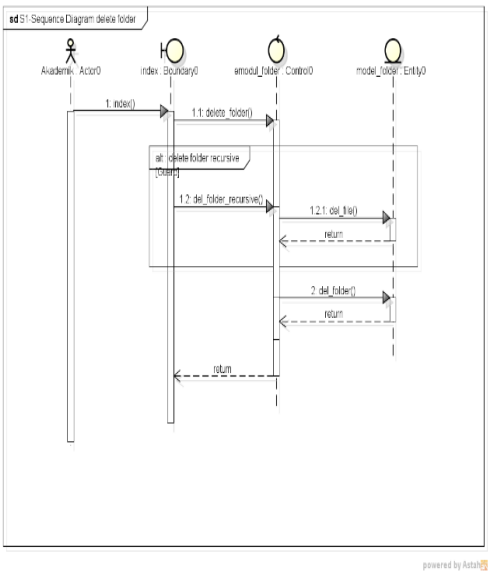
Setelah pengujian selesai, analisis hasil pengujian dilakukan dengan membaca berapa fitur yang valid dan tidak valid. Kemudian dijelaskan setiap fitur yang tidak valid beserta permasalahannya dan bagaimana seharunya yang dilakukan. Kemudian diambil kesimpulan yang berisi bagaimana cara pembuatan behavior UML dan bagaimana hasil ujinya.

4. PERANCANGAN

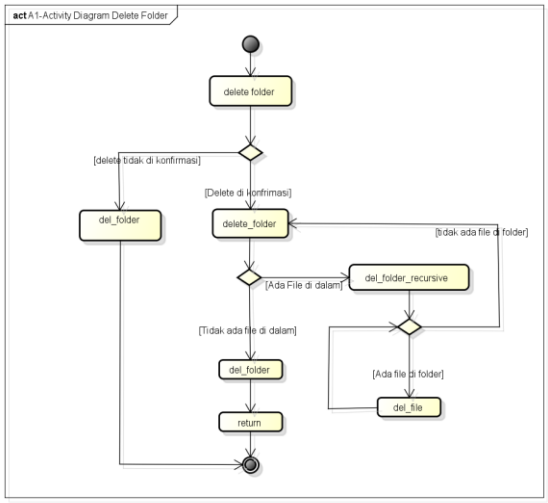
Dalam aplikasi PDTT-UB ini ada 30 fitur yang diujikan, tetapi yang di tampilkan dalam paper ini hanya 1 fitur yaitu fitur delete folder sebagai sample yang digunakan.

4.1 PERANCANGAN *ACTIVITY DIAGRAM* DAN *SEQUENCE DIAGRAM*

Untuk mendapatkan *Activity diagram* dan *Sequence diagram* dapat dilihat atau dirujuk dari dokumentasi perancangan aplikasi, tetapi karena aplikasi PDTT-UB ini sudah jadi dulu tanpa ada dokumentasi nya maka harus dilakukan pembuatan *Activity diagram* dan *Sequence diagram* terlebih dahulu. Pembuatan diagram ini dilakukan setelah observasi *source code* dan demo aplikasi PDTT-UB.



Gambar 6 *Sequence diagram* Delete Folder



Gambar 7 *Activity diagram* Delete Folder

4.2 ANALISIS *ACTIVITY DIAGRAM* DAN *SEQUENCE DIAGRAM*

Setelah mendapatkan *Activity diagram* dan *Sequence diagram*, proses selanjutnya pembuatan Object method associoation table untuk *Sequence diagram* dan Method Activity Table untuk *Activity diagram*

Object method association table

Symbol	Object Method
A	Delete
B	Delete_folder
C	Check_file
D	Ada file dalam folder
E	Tidak ada file dalam folder
F	Del_folder_recursive
G	Del_file
H	Return
I	Del_folder
J	Return
K	Return

Gambar 8 Object Method Association Table

Method Activity Table

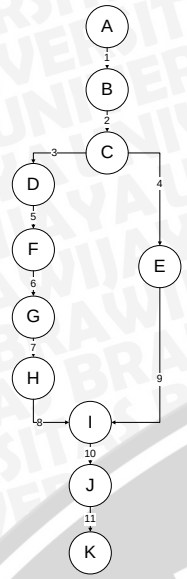
Symbol	Object Method
A	Delete Folder
B	Check konfirmasi
C	Delete di konfirmasi
D	Delete tidak di konfirmasi
E	Delete_folder
F	Check ada file di dalam folder
G	Ada file di dalam
H	Tidak ada file di dalam
I	Del_folder
J	Del_folder_recursive
K	Check masih ada file di dalam folder
L	Ada file di folder
M	Tidak ada file di folder
N	Return

Gambar 9 Method Activity Table

5. HASIL DAN PEMBAHASAN

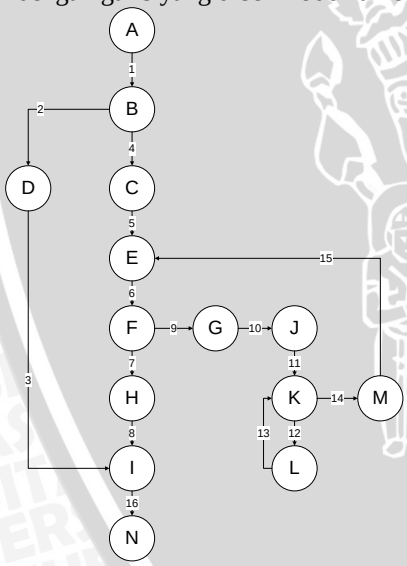
5.1 ANALISIS MODEL FLOW GRAPH

MFG *sequence diagram* delete folder dapat dilihat pada gambar 10. Method-method dari gambar 8 dinotasikan dengan bulatan (*node*) yang diberi kode alphabet sedangkan proses yang menghubungkan method 1 dengan method yang lain dalam proses menambah dan mengubah folder dinotasikan dengan garis yang diberi kode numeric



Gambar 10 MFG *Sequence diagram* Delete Folder

MFG *activity diagram* delete folder dapat dilihat pada gambar 11. Method-method dari gambar 9 dinotasikan dengan bulatan (*node*) yang diberi kode alphabet sedangkan proses yang menghubungkan method 1 dengan method yang lain dalam proses menambah dan mengubah folder dinotasikan dengan garis yang diberi kode numeric



Gambar 11 MFG *Activity diagram* Delete Folder

5.2 HASIL ANALISIS DAN TEST SEQUENCE

Pembuatan kondisi pada fitur delete folder ini berdasarkan hasil demo penggunaan aplikasi PDTT-UB sehingga untuk menjalankan fitur ini perlu ada beberapa hal yang harus dilakukan di aplikasi PDTT-UB yang di klasifikasikan menjadi 3 bagian yaitu *Pre condition*, *Main Scenario* dan *Post Condition*.

Pre condition

1. Akademik sudah login

Main Scenario

1. Akademik menghapus folder yang ada file di dalamnya
2. Akademik menghapus folder yang tidak ada file di dalamnya

Post Condition

1. Folder berhasil dihapus
2. Folder beserta file di dalamnya berhasil di hapus
3. Folder gagal dihapus

Test Sequence

1. *Test Sequence* 1

Edge Label	Message / Acitivity	Category Partition
1	Delete_folder	Null
2	Check_file	File = null
4 & 9	Del_folder	Null
10	Return	Null
11	Return	Null

2. *Test Sequence* 2

Edge Label	Message / Acitivity	Category Partition
1	Delete	Null
2	Check Konfirmasi	Konfirmasi = False
3	Delete tidak di konfirmasi	Null
16	Return	Null

3. *Test Sequence* 3

Edge Label	Message / Acitivity	Category Partition
1	Delete	Null
4	Check Konfirmasi	Konfirmasi = True
5	Delete_folder	Null
6 & 9	Check ada file di dalam folder	File di dalam folder = True
10	Del_folder_recursive	Null
11	Check masih ada file di folder	File di dalam folder = True
12	Del_file	Null
13	Check masih ada file di folder	File di dalam folder = False
14	Delete_folder	Null
15	Delete_folder	Null
16	Return	Null

Test case Generation

Dari analisis kondisi pengujian dan *Test Sequence* dapat dibuat *Test case* yang hasilnya sebagai berikut :

Test case 1 { input : folder tidak ada file didalamnya, output : Folder berhasil dihapus }

Test case 2 { input : folder tidak ada file dalamnya atau ada file didalamnya + konfirmasi = false , output : Folder gagal dihapus}

Test case 3 { input : folder ada file di dalamnya + konfirmasi = true , output : folder beserta file di dalamnya berhasil dihapus}

5.3 HASIL PENGUJIAN

Setelah masing-masing dari flow graph di generate menjadi beberapa *Test case* sesuai hasil analisanya, untuk mempermudah pembacaan, *Test case* akan disatukan dimana isinya dari fitur Manajemen folder terdapat 3 *Test case*, Delete folder terdapat 3 *Test case*, Manajemen file terdapat 5 *Test case*, delete file terdapat 2 *Test case*, Tambah modul 4 *Test case*, edit modul terdapat 10 *Test case* dan yang terakhir delete modul terdapat 3 testcase. Total dari 7 fitur yang ada pada modul admin aplikasi PDTT-UB terdapat 30 *Test case*.

Setelah mendapatkan tabel *Test case*, maka proses selanjutnya yang dilakukan adalah pelaksanaan pengujian. Dalam pelaksanaan pengujian ini, langkah-langkah yang dilakukan berdasar tabel *Test case* yang sudah dibuat. Nantinya hasil pengujian akan di bagi menjadi 2 bagian yaitu Valid dan Tidak Valid.

Valid : Hasil pengujian (realization) dengan apa yang diharapkan (expectation) sesuai

Tidak Valid : Hasil pengujian (realization) dengan apa yang diharapkan (expectation) tidak sesuai

6. KESIMPULAN

Berdasarkan pembahasan yang telah dilakukan pada penelitian ini, maka kesimpulan yang diperoleh dari penelitian ini adalah :

1. Pembuatan *Test case* pada aplikasi PDTT-UB dengan metode behavior UML teknik model flow graph (MFG) dapat dilakukan dengan cara membuat *Activity diagram* dan *Sequence diagram*, kemudian diagram tersebut di buat flow graph dengan metode MFG yaitu pengelompokan setiap proses yang ada di dalam diagram tersebut, setelah itu flow graph akan di generate menjadi *Test case* dengan metode *Test case* Generation.

2. Setelah melakukan pengujian berdasarkan *Test case* yang dibuat dengan metode behavior UML yang terkumpul didapat kan ada 18 *test case* yang mendapat label Valid dan 12 *test case* label tidak valid. *Test case* yang medapat label tidak valid tersebut berada pada fitur Delete folder terdapat 1 *test case* tidak valid, Manajemen file terdapat 2 *test case* tidak valid, Delete file terdapat 2 *test case* tidak valid dan Edit modul terdapat 8 *test case* tidak valid

DAFTAR PUSTAKA

Alter, S., 1992. *Information System : A Management Perspective.* s.l.:The Benjamin / Cummings Publishing : Company, Inc.

Beizer, B., 1990. *Software testing Techniques.* 2nd ed. New York: Van Nostrand Reinhold.

Biswal, B. N., 2010. *Test case Generation and Optimization of Objected-Oriented Software using UML Behavioral Models.* s.l.:s.n.

Davis, G. B., 2008. *Analisis dan Desain Sistem Informasi.* s.l.:s.n.

Fathansyah, I., 2001. *Basis Data.* Bandung: CV Informatika.

Galin, D., 2004. *Software Quality Assurance From Theory to Implementation.* England: Pearson Education Limited.

Graham, D., 2012. *Foundation of Software Testing ISTQB Certification.* England: Thomson.

Hartono, J., 1999. *Analisis dan Desain Sistem Informasi.* Yogyakarta: Andi.

Jogiyanto, 2001. *Sistem Teknologi Informasi.* Yogyakarta: Andi.

McCabe, T. J., 1996. *Structured Testing : A Testing methodology Using the Cyclomatic Complexity Metric.* MD 20899-0001 ed. Gaithersburg: National Institute Standards and Technology.

McLeod, R. J., 2008. *Sistem Informasi Manajemen.* Jakarta: Salemba Empat.

Swain, S. K. & Mohapatra, D. P., 2010. *Test case Generation* from Behavioral UML Models. *International Journal of Computer Applications,* Volume 6, p. 6.

Turban, McLeand & Wetherbe, 1999. *Information Technology for Management.* New York: John Wiley & Sons.