PENGUJIAN PERANGKAT LUNAK DENGAN MENGGUNAKAN MODEL BEHAVIOUR UML

Waskitho Wibisono, Fajar Baskoro

Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya
Jl. Raya ITS Gedung T. Informatika, Surabaya 60111
waswib@its-sby.edu, fajar@its-sby.edu

ABSTRAK

Pengujian perangkat lunak merupakan tahap keempat pada pengembangan perangkat lunak. Pengujian perangkat lunak dilakukan untuk mencari kesalahan perangkat lunak yang dikembangkan. Tahap analisis, desain dan implementasi perangkat lunak tidak menjamin bahwa perangkat lunak bebas kesalahan (fault free). Untuk mengurangi atau menghilangkan kesalahan pada perangkat lunak diperlukan suatu tahap pengujian untuk menemukan kesalahan-kesalahan yang ada pada perangkat lunak.

UML, Unified Modelling Language sebagai Bahasa Pemodelan Terpadu mempunyai perangkat untuk memodelkan perangkat lunak memvisualisasikan use case, statis, dan perilaku perangkat lunak di dalam sistem.

Pengujian perangkat lunak dengan menggunakan model behaviour UML dapat mengetahui kualitas perangkat lunak dalam sistem yang sedang dibangun.

Kata kunci : UML, pengujian, Behaviour.

1. PENDAHULUAN

Perangkat lunak dalam pengembangannya harus diuji karena proses analisis, perancangan dan pemrogramannya tidak bebas kesalahan. Pengujian dilakukan untuk mengetahui apakah perangkat lunak telah sesuai dengan keinginan. Pengujian diharapkan dapat menemukan kesalahan pada perangkat lunak.

Untuk dapat menemukan kesalahan pada perangkat lunak pengujian memerlukan skenario pengujian. Skenario pengujian secara efektif dibangun dari model behaviour UML perangkat lunak.

Pengujian perangkat lunak terdiri dari pengujian unit, pengujian integrasi dan pengujian sistem.

Pengujian sistem adalah pengujian berdasar spesifikasi / kebutuhan perangkat lunak. Pengujian ini biasanya dilakukan berdasarkan spesifikasi yang dianalisa secara informal dan manual. Pengujian ini tidak memiliki metode dan kriteria formal sehingga hasil pengujiannya bisa menjadi tidak konsisten dan rancu. Dukungan alat bantu untuk pengujian ini jarang ditemukan.

Adapun permasalahan yang dirumuskan adalah :

1. Kriteria formal apa saja yang bisa digunakan pada pengujian dan pembangkitan data uji berdasar spesifikasi *model behaviour UML*.
2. Bagaimana pengujian berdasar spesifikasi *model behaviour UML* dapat dilakukan.
3. Bagaimana data uji bisa dibangkitkan berdasar spesifikasi model *behaviour UML*..

Berdasarkan rumusan permasalahan yang ditetapkan diatas maka makalah ini dibuat dengan tujuan :

1. Mempelajari kriteria formal yang bisa digunakan pada pengujian dan pembangkitan data uji berdasar spesifikasi *model behaviour UML*.
2. Mempelajari metode pengujian dan pembangkitan data uji berdasar spesifikasi *model behaviour UML*.
3. Membangun perangkat lunak pembangkit data uji berdasar spesifikasi *model behaviour UML*.

2. PENGUJIAN PERANGKAT LUNAK

Pengujian perangkat lunak adalah proses untuk mencari kesalahan pada setiap *item* perangkat lunak, mencatat hasilnya, mengevaluasi setiap aspek pada setiap komponen (sistem) dan mengevaluasi fasilitas-fasilitas dari perangkat lunak yang akan dikembangkan.

Glen Myers menyatakan beberapa aturan yang dapat digunakan sebagai penjelasan tentang pengujian perangkat lunak:

1. Pengujian merupakan sebuah proses eksekusi program dengan tujuan utama untuk mencari kesalahan
2. Sebuah kasus pengujian dikatakan baik jika memiliki kemungkinan penemuan kesalahan yang tinggi
3. Pengujian yang berhasil adalah pengujian yang menemukan kesalahan

Berdasarkan ketiga pernyataan diatas dapat disimpulkan bahwa pengujian yang baik tidak hanya ditujukan untuk menemukan kesalahan pada perangkat lunak tetapi juga untuk dapat ditemukannya data uji yang dapat menemukan kesalahan secara lebih teliti dan cepat.
Gambaran abstrak pengujian perangkat lunak dapat diillustrasikan program *executable* P (digambarkan dengan disket) dengan himpunan data uji T, dieksekusi pada komputer. Hasil eksekusi itu kemudian dibandingkan dengan *requirement* yang diinginkan dan dievaluasi.
Perhatian utama penguji dalam hal ini adalah bagaimana menghasilkan data uji T yang mampu secara efektif menemukan kesalahan (*fault*) pada program, memberikan informasi tentang kualitas program dan memenuhi *requirement* atau kriteria tertentu.
Pengujian perangkat lunak mencakup beberapa hal, yaitu pengujian yang dilakukan dengan menggunakan teknik atau metoda tertentu dan pengujian berdasarkan strategi pengujian.

2.1. Metode Pengujian

Metode pengujian adalah cara atau teknik untuk menguji perangkat lunak. Metode pengujian berhubungan dengan perancangan data uji yang akan dieksekusi pada perangkat lunak yang dikembangkan. Metode pengujian diharapkan mempunyai mekanisme untuk menentukan data uji yang dapat menguji perangkat lunak secara lengkap (*completeness of test*) dan mempunyai kemungkinan tinggi untuk menemukan kesalahan (*high likelihood for uncovering error*)
Perangkat Lunak sendiri dapat diuji dengan dua cara yaitu :

a. Pengujian dilakukan dengan mengeksekusi data uji dan mengecek apakah fungsional perangkat lunak bekerja dengan baik. Data uji dibangkitkan dari spesifikasi perangkat lunak, yang dalam hal ini menjelaskan fungsional perangkat lunak. Cara pengujian ini disebut dengan *Black Box Testing.*
b. Pengujian dilakukan dengan mengenakan data uji untuk menguji semua elemen program perangkat lunak (data internal, lelaran (*loop*), logika keputusan, jalur). Data uji dibangkitkan dengan mengetahui struktur internal (kode sumber) dari perangkat lunak. Cara pengujian ini disebut dengan *White Box Testing.*

Metode pengujian *white-box testing* dan *black-box testing* mempunyai banyak jenis dan kriterianya. Penjelasan lebih lengkap untuk metode pengujian ini dapat dilihat pada lampiran.

Selain *black box testing* dan *white box testing* terdapat pendekatan lain dalam metode pengujian, metode tersebut adalah pengujian berdasar kode sumber dan pengujian berdasar spesifikasi.

2.2.1. Pengujian berdasar kode sumber

Pengujian berdasar kode sumber (*source code-based testing*) adalah pengujian yang data ujinya dibangkitkan berdasar kode sumber perangkat lunak. Pengujian ini adalah pengujian yang paling umum digunakan.
Pada pembangkitan data uji berdasar kode sumber (*source code-based test case generation*), kriteria pengujian diperlakukan pada perangkat lunak untuk menghasilkan *requirement* pengujian. Sebagai contoh, jika kriteria pengujian pencabangan digunakan, maka pembangkitan data uji harus melibatkan setiap pencabangan pada program.
Pengujian berdasar kode sumber ini adalah sama dengan *white-box testing*, hanya berbeda dalam pendekatannya saja. Gambaran abstrak proses pengujian berdasar kode sumber tampak pada gambar 2.3. berikut
Spesifikasi (dapat berupa spesifikasi informal atau formal) digunakan sebagai dasar untuk penulisan Program. Program kemudian digunakan untuk membangkitkan Data Uji dengan kriteria tertentu . Contoh kriteria pengujian adalah kriteria cakupan (*coverage criterion*) yaitu setiap pencabangan harus diuji paling tidak sekali.
Eksekusi Data Uji pada Program menghasilkan keluaran aktual yang akan dibandingkan dengan keluaran yang diharapkan (*expected output*). Keluaran yang diharapkan dihasilkan dari spesifikasi. *Code-based test case generation* menggunakan spesifikasi untuk membangkitkan kode sumber dan melakukan pengecekan keluaran pada program.

2.2.2. Pengujian berdasar spesifikasi

Pengujian berdasar spesifikasi (*specification-based testing*) adalah pengujian yang data ujinya dibangkitkan berdasar spesifikasi perangkat lunak.
Spesifikasi perangkat lunak selain berfungsi sebagai acuan teknis pengembangan perangkat lunak, khususnya spesifikasi dalam bentuk formal, juga merupakan alat yang dapat digunakan untuk pengujian perangkat lunak. Spesifikasi formal menjelaskan fungsi dari perangkat lunak yang terbentuk dalam suatu format tertentu sehingga dari spesifikasi ini dapat dilakukan proses otomasi untuk pembangkitan data pengujian.
Spesifikasi dapat juga digunakan sebagai basis untuk pengecekan keluaran untuk mengetahui apakah

keluaran dari perangkat lunak sudah sesuai dengan yang diinginkan.
Proses pembangkitan data uji berdasar spesifikasi perangkat lunak digunakan untuk menemukan kesalahan dari spesifikasi perangkat lunak sendiri. Jika langkah ini dilakukan maka masalah-masalah yang muncul dapat dieliminasi pada tahap awal pengembangan perangkat lunak yang pada akhirnya akan menghemat waktu dan sumber daya yang lain. Lebih dari itu pembangkitan data uji pada awal pengembangan perangkat lunak dapat meningkatkan efektivitas perencanaan dan utilisasi sumber daya.
Gambaran abstrak untuk *specification-based testing* tampak pada gambar 2.4.
Pada *specification-based testing* Spesifikasi selain digunakan untuk membuat Program juga digunakan untuk membangkitkan Data Uji. Dalam hal ini, spesifikasi adalah spesifikasi yang bisa dibangkitkan data ujinya dalam bentuk yang diformalkan.
Specification-based testing menggunakan spesifikasi formal sebagai masukan untuk pembangkitan data ujinya. Pembangkitan data uji dilakukan secara otomatis.
Karena spesifikasi formal mempunyai penjelasan yang lengkap, konsisten dan tidak rancu maka hasil pembangkitan data ujinya diharapkan dapat mencakup semua kemungkinan kesalahan dan memenuhi semua kelengkapan dari spesifikasi.
Data ujinya kemudian dieksekusi pada perangkat lunak dan diharapkan pada akhirnya perangkat lunak yang dihasilkan bersifat andal.
Salah satu kelebihan pembangkitan data uji berdasar spesifikasi adalah bahwa data uji dapat dibangkitkan pada tahap awal pengembangan perangkat lunak dan siap eksekusi sebelum program selesai dibangun. Sebagai tambahan, ketika data uji dibangkitkan akan ditemukan ketidak-konsistenan dan kerancuan pada spesifikasi, sehingga spesifikasi dapat diperbaiki sebelum program ditulis.

2.3. Strategi Pengujian

Strategi pengujian perangkat lunak merupakan proses integrasi metode perancangan kasus uji kedalam bentuk urutan langkah pengujian perangkat lunak.
Strategi pengujian perangkat lunak dapat dipandang sebagai urutan langkah seperti pada pengembangan perangkat lunak. Strategi pengujian perangkat lunak secara berurutan adalah pengujian unit (*unit testing*), *integration testing* dan *system testing*.

2.3.1. Pengujian Unit

Pengujian Unit (*Unit Testing*) adalah pengujian yang difokuskan pada unit terkecil dari program (modul). Pengujian ini didasarkan pada informasi dari deskripsi perancangan detil perangkat lunak. Pada umumnya pengujian ini dilakukan secara *white-box* dan *source code based testing* dengan melakukan pengecekan jalur khusus pada struktur kendali modul untuk meyakinkan kelengkapan cakupan dan deteksi maksimum kesalahan.
Adapun hal-hal yang diuji pada pengujian unit ini adalah sebagai berikut :

1. Antarmuka (*interface*) :
   a. Pengujian yang memastikan bahwa informasi yang berasal dari dan keluar dari modul yang diuji mengalir dengan benar.
   b. Pengujian untuk mencari kesalahan pada : penggunaan parameter pada modul baik jumlah maupun tipe datanya, urutan parameter, dan perubah global yang digunakan lintas modul.
   c. Jika modul menggunakan peralatan I/O eksternal maka pengujian harus ditambahkan untuk mencari kesalahan pada : penggunaan atribut, perintah *open/close* dan kondisi *end-of-file* pada file, I/O error handling dan informasi keluaran.
2. Struktur data lokal :
   a. Pengujian yang memastikan bahwa data yang tersimpan selama eksekusi modul terjaga integritasnya
   b. Pengujian untuk mencari kesalahan pada : penggunaan tipe data, inisialisasi atau nilai default, nama variabel, *underflow*, *overflow* dan *addresing exception*
3. Kondisi batas :
   a. Pengujian untuk memastikan bahwa modul beroperasi dengan benar pada batas-batas pemrosesan yang ditentukan
   b. Pengujian untuk mencari kesalahan pada : penggunaan struktur data pada batas-batas deklarasi, penggunaan perintah pengulangan pada batas pengulangannya dan penggunaan nilai maksimum dan minimum yang dibolehkan.
4. Jalur-jalur bebas (*independent paths*) :
   a. Pengujian untuk memastikan bahwa semua kemungkinan jalur kendali yang mungkin telah dieksekusi paling tidak satu kali
   b. Pengujian untuk mencari kesalahan pada : penghitungan (pemrosesan), pembandingan dan alur kendali.
5. Jalur penanganan kesalahan :
   a. Pengujian yang memastikan kondisi salah dapat diantisipasi dan ditangani dengan bersih (*antibugging*) dan tanpa ada pemberhentian.
   b. Pengujian untuk mencari kesalahan pada : respon kesalahan, pemberitahuan

kesalahan, penanganan kesalahan, kondisi kesalahan yang menyebabkan intervensi sistem sebelum penanganan kesalahan dilakukan dan respon kesalahan yang tidak memberikan informasi cukup untuk mencari sumber kesalahan.

Pengujian ini, karena dilakukan dengan metode *white-box,* mempunyai kriteria formal dalam pembangkitan data ujinya. Pengujian ini juga telah mempunyai banyak dukungan *tool* untuk pembangkitan data uji secara otomatis.

2.3.2 Pengujian Integrasi

Pengujian Integrasi (*Integration Testing*) adalah pengujian yang difokuskan pada gabungan unit-unit atau modul-modul yang membentuk kesatuan fungsional. Pengujian ini didasarkan pada informasi dari deskripsi perancangan awal perangkat lunak. Pengujian ini dilakukan untuk menemukan kesalahan antarmuka antar modul. Pengujian ini umumnya dilakukan oleh pengembang sendiri atau dilakukan antar pengembang. Pada umumnya pengujian ini dilakukan secara *white-box* dan *black-box.*

Pada pengujian integrasi dikenal istilah integrasi *non-incremental*, dan integrasi *incremental.* Integrasi *non-incremental* adalah proses integrasi yang menggunakan cara penggabungan langsung modul-modul yang terlibat. Program kemudian diuji secara keseluruhan. Hal ini bisa menyebabkan terjadinya kesalahan yang kompleks karena perkembangan program yang besar.

Sedang integrasi *incremental* adalah integrasi yang dilakukan secara bertahap. Pengujian juga dilakukan per segmen sehingga kesalahan dapat dengan mudah diisolasi dan diperbaiki.

2.3.3 Pengujian Sistem

Pengujian sistem adalah pengujian yang dilakukan pada sistem komputer (*computer-based system*) secara keseluruhan.

Pengujian ini umumnya dilakukan oleh pengembang bersamaan dengan pengembang lain, karena pengujian yang dilakukan berhubungan dengan elemen lain perangkat lunak. Pengujian ini dilakukan untuk mengantisipasi masalah-masalah antarmuka dan perancangan jalur penanganan kesalahan antar sistem pada perangkat lunak. Pengujian sistem dilakukan dengan mensimulasikan data salah atau data yang berpotensi salah pada antarmuka perangkat lunak. Pengujian sistem terdiri dari beberapa proses pengujan yang berbeda karakteristiknya. Pengujian ini mempunyai tujuan utama untuk melakukan validasi sistem secara keseluruhan dan melihat apakah proses integrasi dan kinerja masing-masing elemen sistem sesuai dengan yang dibutuhkan.

Pengujian ini juga berfokus pada validasi apakah perangkat lunak sudah sesuai dengan harapan pemakai.

Pengujian ini dilakukan secara *black-box* dan *specification-based testing.* Urutan pengujian ini dituangkan dalam perencanaan pengujian (*test plan*), yaitu dengan mendefinisikan prosedur pengujian yang kemudian dilanjutkan dengan menentukan data uji.

Pengujian sistem dirancang untuk meyakinkan bahwa :

- Semua kebutuhan fungsional perangkat lunak terpenuhi
- Kinerja perangkat lunak telah sesuai dengan kebutuhan
- Dokumentasi sudah benar
- Kebutuhan lain (*transportability, compatibility, error recovery, maintainability*) terpenuhi

Pengujian sistem untuk perangkat lunak yang dibuat secara khusus (*customized*) disebut dengan *acceptance test* yang dilakukan oleh pemakai dengan melakukan validasi dari spesifikasi kebutuhan. Pengujian ini biasanya dilakukan oleh pemakai secara informal ataupun secara sistematis selama perode waktu tertentu agar dapat ditemukan kesalahan kumulatif pada suatu periode waktu.

Sedangkan untuk produk perangkat lunak pengujiannya disebut dengan *Alpha Testing* dan *Beta Testing*. Pengujian ini dilakukan oleh *end user* (pemakai akhir). *Alpha Testing* adalah pengujian yang dilakukan oleh pemakai pada lingkungan pengembang, dalam hal ini lingkungan yang terkendali. *Beta Testing* adalah pengujian yang dilakukan oleh pemakai pada lingkungan pemakai sendiri, dimana lingkungan perangkat lunak tidak lagi dapat dikendalikan oleh pengembang.

Selain dari hal-hal diatas pengujian sistem juga meliputi :

1. Pengujian *Recovery* (*Recovery Testing*), pengujian ini memaksa perangkat lunak untuk gagal dalam berbagai cara dan mengecek apakah proses *recovery* dapat dilakukan dengan baik
2. Pengujian Keamanan (*Security Testing*), pengujian ini merupakan pengujian sistem proteksi yang diaplikasikan pada perangkat lunak. Perangkat lunak dipenetrasi oleh suatu rangkaian proses yang ilegal.
3. *Stress Testing*, pengujian ini memaksa perangkat lunak untuk bekerja abnormal baik dalam kuantitas, frekuensi dan volume datanya.

4. Pengujian Kinerja (*Performance Testing*), pengujian ini dilakukan untuk mengecek kinerja perangkat lunak pada waktu *run-time*. Pengujian seharusnya dilakukan pada setiap proses pengujian. Pengujian ini dilakukan dengan mengukur parameter-parameter sistem, seperti utilisasi sumber daya perangkat lunak, waktu respon dll.

Pengujian sistem adalah pengujian berdasar spesifikasi kebutuhan perangkat lunak. Pengujian ini biasanya dilakukan berdasarkan spesifikasi yang dianalisa secara informal dan manual. Pengujian ini juga tidak memiliki metode dan kriteria formal sehingga hasil pengujiannya bisa menjadi tidak konsisten dan rancu. Dukungan alat bantu untuk pengujian ini jarang ditemukan.

3. PENGUJIAN DARI SPESIFIKASI MODEL BEHAVIOUR UML

3.1. Model Diagram Sequence

Diagram Sequence digunakan untuk mengambarkan interaksi antar obyek dalam berkomunikasi saling mengirim message disusun berdasarkan urutan waktu. Diagram sequence ini bersama-sama dengan diagram collaboration biasa disebut juga dengan diagram interaksi, sebab keduanya berfungsi menggambarkan interaksi antar objek.

Diagram sequence disusun , bagian mendatar atas sebagai tempat objek-objek yang berinteraksi dan bagian vertikal menggambarkan urutan waktu objek-objek saling mengirim dan menerima messages.

Komponen yang membangun diagram sequence adalah :

1. Object
   Objek digambarkan didalam kotak. Isi kotak berupa nama objek kemudian diikuti dengan type objek. Tipe objek disesuaikan dengan tipe class yang telah didefinisikan di dalm diagram class.
2. Timeline/ Life line
   Timeline digambarkan berupa garis putus-putus menempel pada objek secara vertikal dari atas ke bawah.
3. Messages
   Messages berupa operasi yang ada di dalam objek disertai dengan parameter objek yang dikirimkan kepada objek lain.
4. Destroy
   Destroy menggambarkan bahwa pada waktu yang telah ditentukan objek yang didefinisikan sudah dihilangkan / di destroy keberadaannya di memory komputer.
5. Aktivasi
   Aktivasi digambarkan dengan persegi panjang. Aktivasi ini melambangkan kapan objek tersebut aktiv berperan di dalam sistem sampai objek tersebut pasif kembali.

Model diagram sequence digunakan untuk :

1. Memodelkan aliran kontrol / pesan-pesan yang dikirim dan diterima oleh masing-masing object dalam kurun waktu tertentu.
2. Memodelkan urutan pemrosesan
3. Memodelkan method yang menangani pesan pada masing-masing object

Memodelkan pesan/ messages apa saja yang digunakan untuk berinteraksi oleh masing-masing object

Dewasa ini kebutuhan perangkat lunak dengan tingkat kompleksitas tinggi dan kritis cukup meningkat. Perangkat lunak tersebut biasanya mempunyai deskripsi yang jelas, lengkap dan bahkan dalam bentuk spesifikasi formal. Pada kenyataannya deskripsi yang jelas tersebut kebanyakan hanya ada pada tingkat (*level*) unit, sedang pada tingkat sistem deskripsi hanya dilakukan secara informal.

UML merupakan notasi pemodelan yang cukup baik untuk menjelaskan perangkat lunak pada semua tingkat pengembangan perangkat lunak. Dan ini merupakan peluang yang dapat digunakan untuk penentuan data uji. Walaupun UML tidak terlalu formal tetapi deskripsi pada UML cukup teliti dan lengkap untuk menjelaskan perangkat lunak.

Statechart UML dipilih sebagai awal pembangkitan data uji berdasar spesifikasi karena statechart menjelaskan kelakuan (*behavior*) sistem.

Pengujian kelakuan perangkat lunak sebenarnya dapat dilakukan dengan menguji setiap metode (*method*) dari suatu obyek, karena kelakuan suatu obyek diimplementasikan dari metodenya. Tetapi pengujian setiap metode obyek hanya menguji sebagian kelakuan obyek bukan keseluruhan dari sistem [10].

3.2. Model Diagram Collaboration

Collaboration bermakna kerjasama antar bagian untuk melaksanakan fungsi tertentu yang dibebankan kepada masing-masing bagian objek.

Diagram Collaboration digunakan untuk mengambarkan susunan organisasi antar objek dalam berinteraksi dan bekerjasama untuk melaksanakan fungsi yang didefinisikan di dalam sistem software. Dalam diagram collaboration ini digambarkan objek apa saja yang bekerja sama dan message apa yang dikirim dan diterima untuk berkomunikasi dalam rangka melaksanakan fungsi yang didefinisikan.

Komponen yang membangun diagram collaboration adalah :
1. Object
2. Link
3. Messages

Kegunaan Diagram Collaboration

Diagram Collaboration digunakan untuk :
1. Memodelkan urutan interaksi antar obyek dalam bentuk susunan organisasi
2. Memodelkan aliran kontrol / pesan-pesan yang dikirim dan diterima oleh masing-masing object.
3. Memodelkan urutan pemrosesan
4. Memodelkan method yang menangani pesan pada masing-masing object

Memodelkan pesan/ messages apa saja yang digunakan untuk berinteraksi oleh masing-masing object

3.3. Model Diagram Activity

Diagram activity digunakan untuk mengambarkan urutan kerja masing-masing obyek dalam menyelesaiakan permasalahan tertentu sesuai dengan fungsi kerja masing-masing obyek. Diagram activity pada hakekatnya adalah flowchart yang menunjukkan aliran aktivitas ke aktivitas dalam memenuhi fungsi layanan yang didefinisikan didalam use case.

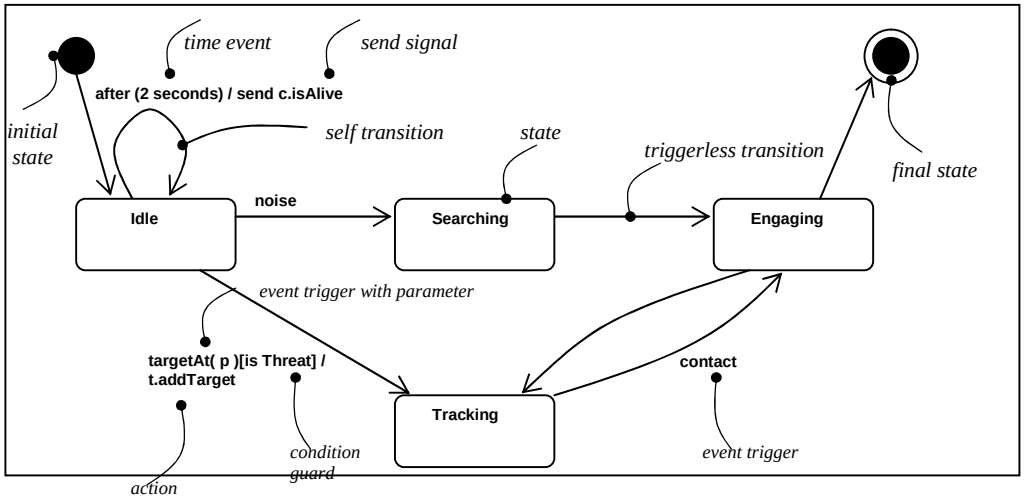
Diagram activity digunakan untuk :
1. Memodelkan aliran kerja object / workflow
2. Memodelkan opersional proses

3.4. Model Diagram Statechart

Statechart UML adalah diagram yang menggambarkan kelakuan sistem secara keseluruhan. Karena menggambarkan kelakuan sistem secara keseluruhan maka pembangkitan data uji berdasar statechart (state machine) dianggap menguji keseluruhan sistem. Statechart UML dibuat berdasar State Machine yang digunakan oleh David Harel .

Mesin Status (State machine) adalah mesin yang menggambarkan atau memodelkan kelakuan dari obyek individual. Mesin Status adalah kelakuan yang menggambarkan urutan status dari suatu obyek yang hidup pada suatu waktu (*lifetime*) karena respon dari *event*.

Mesin Status dan komponen-komponen pendukungnya secara konseptual dapat dijelaskan sebagai berikut :
1. **Mesin Status** : suatu *behavior* (kelakuan) yang menggambarkan urutan status (***state***) dari suatu obyek yang hidup pada waktu hidup (*lifetime*) nya karena respon dari ***event***.
2. Status (***state***) : kondisi atau situasi pada saat suatu obyek hidup yang memenuhi suatu kondisi, melakukan suatu aktivitas, atau menunggu suatu ***event.*** Pada statechart UML status digambarkan dengan kotak bersudut tumpul.
3. ***Event*** : spesifikasi suatu kejadian (*occurrence*) yang mempunyai alokasi ruang dan waktu. Didalam kontek Mesin Status, *event* adalah kejadian (*occurrence*) dari suatu pemicu (*stimulus*) yang memicu suatu transisi status. Pada statechart UML *event* digambarkan (dituliskan) sebagai teks yang menyertai transisi.
4. Transisi : adalah hubungan antara dua status yang menunjukkan bahwa obyek pada pada saat status pertama akan melakukan suatu aksi



Gambar 1. Contoh Mesin Status

tertentu dan masuk ke status kedua jika suatu *event* terjadi dan suatu kondisi tertentu dipenuhi. Pada statechart UML transisi digambarkan dengan anak panah berarah, dengan asal anak panah adalah status sumber (asal) dan anak panah tujuan adalah status target (tujuan).

5. Aktivitas : adalah eksekusi keluar yang *non atomic* pada mesin status. Pada statechart UML aktivitas digambarkan (dituliskan) sebagai teks yang menyertai *event* dan transisi.
6. Aksi : adalah komputasi atomik *executable* yang dihasilkan dari perubahan status atau mengembalikan suatu nilai. Pada statechart UML aktivitas digambarkan (dituliskan) sebagai teks yang menyertai *event* dan transisi.

State adalah kondisi atau situasi dari suatu obyek. Obyek pada kehidupannya dapat memenuhi beberapa kondisi, melakukan suatu aktivitas tertentu atau menungggu suatu *Event*. Suatu obyek dapat berada pada status tertentu pada suatu waktu yang terbatas.

4. PERANCANGAN PERANGKAT LUNAK

Pada perancangan pengujian model behaviour UML yang digunakan adalah Collaboration Diagram, Sequence Diagram, Activity Diagram dan Statechart Diagram.

Perancangan perangkat lunak dilakukan dengan tahapan sebagai berikut :

1. Perancangan format masukan dan keluaran
2. Perancangan aspek statis perangkat lunak, yaitu penentuan kelas dan hubungan antar kelasnya. Penentuan kelas meliputi data anggota dan fungsi anggota. Hubungan antar kelas digambarkan dengan class diagram.
3. Perancangan aspek dinamis, yaitu urutan penghidupan obyek, pemanggilan fungsi anggota dan pemusnahan obyek dilakukan dengan collaboration dan sequence diagram.

Pada perancangan aspek statis, penentuan struktur data dari anggota data kelas dilakukan dengan mencari representasi paling optimal untuk memenuhi spesifikasi kebutuhan perangkat lunak. Untuk representasi internal dari kelas banyak digunakan *multi-list*.

Tumpukan digunakan pada algoritma untuk mengubah ekpresi predikat dari *linked-list* ke bentuk struktur pohon. Struktur tumpukan menggunakan tipe tumpukan dari pointer yang menunjuk elemen dari pohon.

Perancangan aspek dinamis pada perangkat lunak digambarkan dengan collaboration dan sequence diagram. Diagram ini digunakan untuk menggambarkan organisasi obyek dan interaksi antar obyek dan menggambarkan urutan-urutan *message* serta aliran kendali.

Perangkat lunak mempunyai tiga sub program yang bisa dieksekusi. Program tersebut adalah : Program pembangkit file spesifikasi, Program pembangkit data uji dengan kriteria Full Predicate Coverage dan Program pembangkit data uji dengan kriteria Transition Pair. Masing-masing program dijelaskan dalam sub bab berikutnya.

Program ini adalah program yang bersifat interaktif yang mempunyai beberapa pertanyaan yang harus diisikan oleh pengguna. Pertanyaan tersebut adalah :

1. Apa deskripsi dari state machine file spesifikasi yang akan dibangkitkan ? Deskripsi ini diisi string yang boleh tidak diisi.
2. Berapa jumlah status (*state*) yang dipunyai state machine ? Jumlah ini harus diisikan nilai digit lebih besar dari 0 dan kurang dari 31. Status yang bisa ditangani oleh perangkat lunak terbatas untuk 30 status.
3. Status apa saja yang ada state machine ? Semua status yang ada pada state machine harus dituliskan disini. Status ke-1 diset sebagai status awal (*initial state*). boleh ada status yang tidak mempunyai nama

Kesulitan yang dihadapi pada saat perancangan file eksternal untuk keluaran yang dalam hal ini adalah spesifikasi uji adalah penentuan apakah file eksternal ini berisi spesifikasi uji yang memenuhi semua kriteria pembangkitan data uji atau berisi spesifikasi uji untuk satu kriteria saja.

Pemisahan atau tidaknya file eksternal spesifikasi uji untuk masing-masing kriteria ini berhubungan dengan pekerjaan selanjutnya yaitu otomatisasi pembangkitan *test script.*

Ada dua alternatif untuk memecahkan masalah ini :

1. Penggunaan sebuah file yang mencakup semua kriteria pengujian
2. Penggunaan beberapa file sesuai dengan jumlah kriteria yang diinginkan.

Pemisahan modul dan file eksternal keluaran akan memudahkan pembuatan laporan dan analisis hasil pengujian.

5. KESIMPULAN

Kesimpulan yang dapat diambil adalah :

Pembangkitan data uji berdasar spesifikasi statechart ini terbatas untuk transisi *enabled* dengan *change event*, sehingga dapat dikembangkan untuk transisi selain jenis tersebut.

Pembangkitan data uji berdasar spesifikasi ini terbatas pada spesifikasi statechart, sehingga dapat dikembangkan untuk spesifikasi yang lain (class diagram, collaboration diagram dll).

6. DAFTAR PUSTAKA

[1] Bahrami Ali, *Object Oriented System Development,,* Singapore, McGraw-Hill International Edition Singapore, 1999.

[2] Booch, G, Rumbaugh J., Jacobson I., *The Unified Modelling anguage User Guide.* Massachusetts., Addison Wesley Longman Inc, 1999.

[3] Larman C., *Appllying UML and Pattern An Introduction to Object Oriented Analysis and Design, New Jersey,* Prentice Hall PTR, 1998.

[4] Offut, A Jefferson. dan Abdulrazik, Aynur. *Generating test cases from UML specifications.* In Proceeding of the second IEEE International Conference on Unified Modeling Language (UML99), pages 416-429, Fort Collins, CO, IEEE Computer Society Press, October 1999.

[5] Offut, A Jefferson. dan Liu, Shaoying. *Generating test data from SOFL specifications.* The Journal of Systems and Software, 1999.

[6] Offut, A.Jefferson. Xiong, Yiwei. dan Liu, Shaoying. *Criteria for Generating Specification-based tests*. http://www.isse.gmu.edu

[7] Perry, William. *Effective Methods for Software Testing.* John Wiley & Sons, Inc., 1995.

[8] Pressman, Roger S., *Software Engineering – A Practitioner's Approach,* New York, McGraw-Hill Inc., 1997

[9] Valacich J.S. , George J.F. , Hoffer J.A., *Essentials of System Analysis and Design,* New Jersey, Prentice Hall, 2001.